Doctoral Dissertation

A Software Engineering Environment for Ada 2012 Programs

(Ada 2012 プログラムのためのソフトウェアエンジニアリング環境)

Bo Wang

Graduate School of Science and Engineering,
Saitama University

Supervisor: Professor Jingde Cheng

March 2017

Abstract

Programming language Ada is the only ISO standard, object-oriented, concurrent, real-time programming language. It is intended for use in large, long-lived applications where reliability and efficiency are essential and they are almost concurrent systems. Ada has gone through Ada 83, Ada 95, Ada 2005, and the last version of Ada, known as Ada 2012, is the world's premier programming language for engineering safe, secure and reliable software. Ada 2012 has a giant changes and extensions from Ada 2005, which can support many new features, such as contract-based programming, increased expressiveness, multicore support, container enhancements.

In general, Ada is always used to manipulate safety-critical systems, such as commercial and military aircraft avionics, air traffic control systems, railroad systems, medical devices, and so on. The systems are almost concurrent systems and they are vital to personal life safety, such that they are allowed any malfunction, otherwise personal life and assets might suffer catastrophic consequences. However, it is quite difficult to design, understand, test, debug, and maintain concurrent programs, the reason is that multiple control flow and data flow can exist simultaneously in concurrent systems, and execution or evaluation of statements are unpredictable and non-deterministic.

For solving the difficulties, only one tool cannot resolve all of difficulties, but rather we should have a set of software tools. Usually, in the set of software tools, they are not independent on each other. An engineering environment could integrate them together. Software Engineering Environment (SEE) is an engineering environment that integrates various tools and provides comprehensive facilities to designing, understanding, testing, debugging and maintaining software systems.

So far, some software engineering tools were developed for Ada programs, however, because Ada 2012 has large changes and extensions, they hardly support for software engineering activities of Ada 2012. Therefore, the development of supporting tools that can deal with Ada 2012 is an urgent study.

In order to support software development activities using Ada 2012, we proposed a software engineering environment for Ada 2012 programs. For development of SEE for Ada 2012 programs, we analyzed requirements for developing SEE of Ada 2012. Based on requirements, we designed an SEE for Ada 2012 programs. And then we confirmed core components of the SEE for Ada 2012 programs. Finally, we proposed methods to develop core components and gave implementations of them.

This work has following contributions. We proposed a software engineer environment for Ada 2012 programs and implemented its core components. We designed and implemented a definition-use net generator for Ada 2012 programs, which can automatically generate DUNs of target Ada 2012 programs. And then we designed and implemented a system dependence net generator for Ada 2012

programs, which can automatically generate SDNs of the target Ada 2012 programs including new types of program dependences and one interprocedural relation that we found and formally defined. We also designed and developed a run-time tasking deadlock detector for Ada 2012 programs, which can detect tasking deadlocks raised by all types of synchronization waiting relations in Ada 2012 programs, and we found some types of queue operation related tasking deadlocks in Ada 2012 programs. We also showed how to utilize Contract-based Programming (CBP) with Ada 2012 to solve challenges of future advanced software engineering. We proposed methods and solutions from five areas, such as security, continuity, reactive-ability, predictability, and anticipatable-ability. By CBP, we gave algorithms to develop an SSB-based system as a persistent computing system and an ARRS as an anticipatory system.

Structure of this thesis is as follows. Chapter 1 presents background, motivation, and purpose of this research. Chapter 2 presents a software engineering environment for Ada 2012 programs and gives the core components of software engineering environment. Chapter 3 presents definition-use nets of Ada 2012 programs. Chapter 4 presents a definition-use net generator for Ada 2012 programs. Chapter 5 elaborates system dependence nets of Ada 2012 programs. Chapter 6 shows the implementation of system dependence nets generator for Ada 2012 programs. Chapter 7 presents a tasking deadlock detector for Ada 2012 programs. Chapter 8 discusses contract-based programming for future computing with Ada 2012. Finally, concluding remarks is given in Chapter 9.

Acknowledgments

I would like to express special thanks first of all to my research supervisor Professor Jingde Cheng for his patience, invaluable guidance, and suggestions on all aspects of my academic life. In fact, Professor Jingde Cheng is not only a supervisor for my academic life, but also the most important tutor in my life. The things I learnt from him is not only limited in the academic field, but he taught me how to become a wise and good man.

I am very grateful to my thesis committee: Professor Norihiko Yoshida, Associate Professor Noriaki Yoshiura, Associate Professor Takashi Horiyama and Associate Professor Yuichi Goto for their support, invaluable advice and comments to my research.

I am grateful to Associate Professor Yuichi Goto for teaching me so much in my research and helping me in all respects. I would also like to thank other AISE lab members who have helped me in my research.

I also express my special thanks to Saitama University, and all of those teachers and staffs of the university who have helped me in my life studying abroad.

Finally, I would like to express my special thanks to my parents, who financed and support me to live and study in Japan. It would be impossible for me to pursue my doctoral degree in Japan without their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Programming language Ada is the only ISO standard, object-oriented, concurrent, real-time programming language. It is intended for use in large, long-lived applications where reliability and efficiency are essential and they are almost concurrent systems. Ada has gone through Ada 83, Ada 95, Ada 2005, and the last version of Ada, known as Ada 2012, is the world's premier programming language for engineering safe, secure and reliable software. Ada 2012 has a giant changes and extensions from Ada 2005, which can support many new features, such as contract-based programming, increased expressiveness, multicore support, container enhancements.

In general, Ada is always used to manipulate safety-critical systems, such as commercial and military aircraft avionics, air traffic control systems, railroad systems, medical devices, and so on. The systems are almost concurrent systems and they are vital to personal life safety, such that they are allowed any malfunction, otherwise personal life and assets might suffer catastrophic consequences. However, it is quite difficult to design, understand, test, debug, and maintain concurrent programs, the reason is that multiple control flow and data flow can exist simultaneously in concurrent systems, and execution or evaluation of statements are unpredictable and non-deterministic.

For solving the difficulties, only one tool cannot resolve all of difficulties, but rather we should have a set of software tools. Usually, in the set of software tools, they are not independent on each other. An engineering environment could integrate them together. Software Engineering Environment (SEE) is an engineering environment that integrates various tools and provides comprehensive facilities to designing, understanding, testing, debugging and maintaining software systems.

So far, some software engineering tools were developed for Ada programs, however, because Ada 2012 has large changes and extensions, they hardly support for software engineering activities using Ada 2012. Therefore, the development of supporting tools that can deal with Ada 2012 is an urgent issue.

1.2 Purposes and Objectives

In order to support software development activities using Ada 2012, we propose a software engineering environment for Ada 2012 programs. For development of SEE for Ada 2012 programs, we analyze requirements for developing SEE of Ada 2012. Based on requirements, we design an SEE for Ada 2012 programs. And then we confirm the core components of the SEE for Ada 2012 programs. Finally, we propose methods to develop the core components and give implementations of them. Meanwhile, another is about contract-based programming with Ada 2012. We show how to utilize Contract-based Programming (CBP) with Ada 2012 to solve challenges of future advanced software engineering. We propose methods and solutions from five areas, such as security, continuity, reactive-ability, predictability, and anticipatable-ability.

1.3 Structure of the Thesis

The rest of this thesis is organized as follows. Chapter 2 presents a software engineering environment for Ada 2012 programs and gives the core components of software engineering environment. Chapter 3 presents definition-use nets of Ada 2012 programs. Chapter 4 presents a definition-use net generator for Ada 2012 programs. Chapter 5 elaborates system dependence nets of Ada 2012 programs. Chapter 6 shows the implementation of system dependence nets generator for Ada 2012 programs. Chapter 7 presents a tasking deadlock detector for Ada 2012 programs. Chapter 8 discusses contract-based programming for future computing with Ada 2012. Finally, concluding remarks is given in Chapter 9.

Chapter 2

A Software Engineering Environment for Ada 2012 Programs

2.1 An Overview of the Software Engineering Environment

In general, Ada is always used to manipulate safety-critical systems, such as commercial and military aircraft avionics, air traffic control systems, railroad systems, medical devices, and so on. The systems are almost concurrent systems and they are vital to personal life safety, such that they are allowed any malfunction, otherwise personal life and assets might suffer catastrophic consequences. However, it is quite difficult to design, understand, test, debug, and maintain concurrent programs, the reason is that multiple control flow and data flow can exist simultaneously in concurrent systems, and execution or evaluation of statements are unpredictable and non-deterministic.

For solving the difficulties, only one tool cannot resolve all of difficulties, but rather we should have a set of software tools. Usually, in the set of software tools, they are not independent on each other. An engineering environment could integrate them together. Software Engineering Environment (SEE) is an engineering environment that integrates various tools and provides comprehensive facilities to designing, understanding, testing, debugging and maintaining software systems [5, 26].

SEE should be able to parse Ada 2012 programs from syntax and semantics, aid system engineers to make software design and specification, provide a basis software test coverage criteria for testing target programs, give a complexity metric to target programs, help engineers to understand the target program to maintain and reengineering, and should be able to provide a users with some tools to detect the existence of deadlocks in a concurrent program. Meanwhile, we faced some issues on contract-based programming with Ada 2012, which is important feature of Ada 2012. Although contract-based programming is important to assure the correctness of safety-critical systems and is heavily used in practice, in the using

of programs written on contract-based programming with Ada 2012, we will face the following problems: (1) How do we determine the contract between subprograms, to ensure the proper use of contracts? (2) For Ada 2012, how to establish a set of complete contract program-ming based protocols, namely, under international standards, which parts of pro-gramming language Ada 2012 can be used by contract-based run-time check; (3) For large Ada 2012 programs, how do we maintain the contract conditions? (4) When there are a large number of contracts in a system, how do we record the contract conditions in practice? (5) In view of the written contracts, how do we test them? (6) When we change a subprogram's contract, will this contract produce the ripple effect?

A software engineering environment should be an ideal solution to maintain and enhance Ada 2012, and aid developers to prevent and/or resolve the difficulties of software engineering. The software engineering environment consists of static tools and a dynamic tool. Static tools process some static analysis of control flow and data flow of Ada 2012 programs.

2.2 Requirement Analyses for SEE of Ada 2012 Programs

In order to solve the difficulties on software engineering activities of Ada 2012, here, we give some requirement analyses for SEE of Ada 2012 programs to develop an SEE.

R1: SEE must provide tools to support all features of Ada 2012 programs in design, development, understanding, and maintenance of software engineering.

R2: SEE must provide tools to capture correct information in the programs.

R3: SEE must provide tools to capture information in the programs as demand-based.

R4: SEE must provide tools to define software test coverage criteria and generate software test data.

R5: SEE must provide tools to grasp and represent dependent relationships in an Ada 2012 program.

R6: SEE must provide tools to understand the nature of behavior of software.

R7: SEE must provide tools to isolate the possible bug location, and pick up candidates of bugs.

R8: SEE must provide tools to grasp and represent multiple control flows and data flows in an Ada 2012 program.

R9: SEE must provide tools to detect the existence of deadlocks in a concurrent Ada 2012 program.

R10: SEE must provide tools to decompose slices of target programs to support software maintenance activities.

R11: SEE must provide tools to measure values of the complexity of a target concurrent Ada 2012 program.

R12: SEE must provide tools to visualize representations of grasping information.

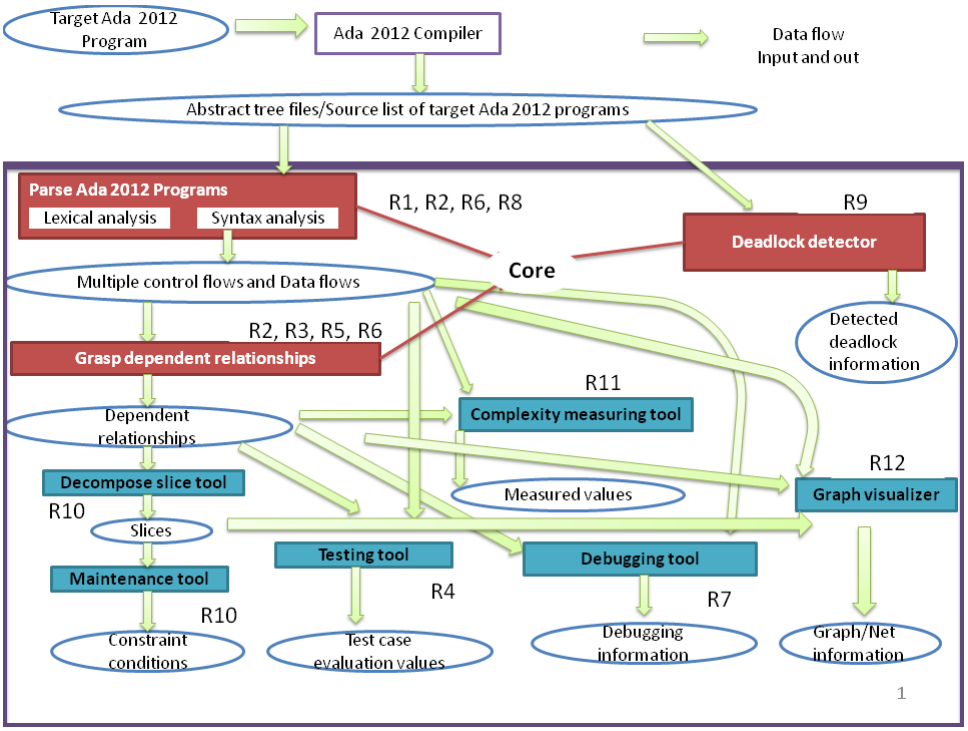

Figure 2.1: A Software Engineering Environment for Ada 2012 Programs

2.3 Designing an SEE for Ada 2012 Programs

To satisfy requirements, an SEE were designed, which integrated 9 tools for Ada 2012 programs. The SEE is separated into 2 types of tools, such as static tools and dynamic tools. To design, understand, test, debug, and maintain concurrent Ada 2012 programs, the tools were integrated to correlate as the following Figure 2.1. Some output results are considered as the input data of another tool and some necessary data is shared between tools.

In the SEE, there is a tool that it can parse target Ada 2012 programs, which can give lexical analyses and syntax analyses to generate multiple control flows, data flows, nondeterministic interaction, and some other required information.

A statement in a program cannot exist independently, but is dependent on others. Dependent relationships are dependences holding between statements in a program that is determined by control flows and data flows in the program.Therefore, we need a tool to compute and grasp dependent relationships.

A program slice is a method for automatically decomposing programs. It provides a reduced program by erasing irrelevant statements in the original program based on certain statements and a set of variables, called the "criterion", such that it is a tool to decompose slice tool. The tool could generate various slices of target Ada 2012 programs. When we have slices of target Ada 2012 programs, a tool uses the decomposition slices of target programs to support software maintenance activities and then constraint conditions.

When we have dependent relationships, sometimes, the user could calculate

coverage of a test-case with respect to the dependent relationships. The testing tool could give test-case evaluation values. Also, we need some debugging methods and tools to support bug localization, analysis, and correction in concurrent Ada 2012 program debugging. Dependent relationships are input the tool to generate debugging information. Meanwhile, the complexity measuring tool could measure the complexity of a program by using multiple control flows and data flows and dependent relationships. The tool could generate measured values of complexity of target Ada 2012 programs.

Only grasping, gaining and/or computing necessary information are not enough and sometimes impalpable. We need a graph visualizer tool to show visualization of graph/net information, such that we could use it to present control flows, data flows, dependent relationships, slices, and so on.

Tasking deadlocks is a serious and complex issue in concurrent Ada programs, such that it is inevitable to detect deadlocks of target Ada 2012 programs. The SEE of Ada 2012 programs should provide a tool to detect deadlocks and provide detected deadlock information.

2.4 Confirming the Core Components of SEE of Ada 2012

When a target Ada 2012 program has been parsed, we could gain all lexical and syntax information of target programs. By some analyses, control flows and data flows are computed. If there are control flows and data flows, we can do other software engineering activities as the figure above. Therefore, we consider that it is the most basic tool as static analysis tools of the SEE. Furthermore, grasping dependent relationships is another basic issue, such that it is essential software engineering activity, because we can carry on many software engineering activities by using dependent relationships, such as decompose slicing programs, testing, debugging, and complexity measuring. Meanwhile, we should provide a tool to detect deadlocks in Ada 2012 programs.

We consider the three tools should be the core components of SEE of Ada 2012 as above. It is critical to provide the core components for software engineering activities of Ada 2012.

Chapter 3

Definition-Use Nets of Ada 2012 Programs

3.1 Previous Researches

Cheng has proposed Nondeterministic Parallel Control Flow Net (CFN), and Nondeterministic Parallel Definition-Use Net (DUN) [15], both are arc-classified digraphs, in order to represent concurrent and/or distributed programs. The CFN represents multiple control flows in a concurrent program as well as the single control flow in a sequential program. The DUN is an extension of CFN, such that it represents multiple control flows, definitions and uses of variables, and inter-process synchronization and communication in a concurrent program. Also, Cheng gave formal definitions to CFN and DUN as follows.

Definition 1. A digraph is an ordered pair (V, A), where V is a finite set of elements, called vertices, and A is a finite set of elements of the Cartesian product $V \times V$, called arcs, i.e., $A \subseteq V \times V$ is a binary relation on V. For any arc $(v_1, v_2) \in A$, v_1 is called the initial vertex of the arc and said to be adjacent to v_2, and v_2 is called the terminal vertex of the arc and said to adjacent from v_1. A predecessor of a vertex v is a vertex adjacent to v, and a successor of v is a vertex adjacent from v. The in-degree of a vertex v, denoted in-degree (v) is the number of predecessors of v, and the out-degree of a vertex v, denoted out-degree (v), is the number of successors of v. A simple digraph is a digraph (V, A) such that $(v, v) \notin A$ for any $v \in V$.

Definition 2. An arc-classified digraph is an n-tuple $(V, A_1, A_2, \ldots, A_{n-1})$ such that every $(V, A_i)(i = 1, \ldots, n-1)$ is a digraph. A simple arc-classified digraph is an arc-classified digraph $(V, A_1, A_2, \ldots, A_{n-1})$ such that $(v, v) \notin A_i(i = 1, \ldots, n-1)$ for any $v \in V$.

Definition 3. A path in a digraph (V, A) or an arc-classified digraph $(V, A_1, A_2, \ldots, A_{n-1})$ is a sequence of arcs $(a_1, a_2, \ldots, a_l)$ such that the terminal vertex of a_i is the initial vertex of a_{i+1} for $(1 \leq i \leq l-1)$, where $a_i \in A(1 \leq i \leq l)$ or $a_i \in A_1 \cup A_2 \cup \ldots \cup A_{n-1}(1 \leq i \leq l)$, and $l(l \geq 1)$ is called the length of the path. If the initial vertex of a_1 is v_I and the terminal vertex of a_l is v_T, then the path is called a path from v_I to v_T, or $v_I - v_T$ path for short. A path in a digraph or an

arc-classified digraph is said to be simple if it does not include the same arc twice. A path in a digraph or an arc-classified digraph is said to be elementary if does not include the same vertex twice.

Definition 4. A deterministic control flow graph is a quadruple (V, A, s, t), where (V, A) is a simple digraph such that out-degree$(v) \leq 2$ for any $v \in V$, $s \in V$ is a unique vertex, called start vertex, such that in-degree $(s) = 0$, $t \in V$ is a unique vertex, called termination vertex, such that out-degree $(t) = 0$, and for any $v \in V(v \neq s, v \neq t)$, there exists a path from s to v and a path from v to t. Any arc $(v_1, v_2) \in A$ is called a control flow arc.

Definition 5. A nondeterministic control flow graph is a quadruple (N, V, A, s, t), where $N \subset V$ is a finite set of elements, called nondeterministic vertices, (V, A) is a simple digraph such that out-degree $(v) \leq 2$ for any $v \in (V - N)$ and $2 \leq$ out-degree $(v) \leq k$ for any $v \in N$, $s \in V$ is a unique vertex, called start vertex, such that in-degree $(s) = 0$, $t \in V$ is a unique vertex, called termination vertex, such that out-degree $(t) = 0$, and for any $v \in V(v \neq s, v \neq t)$, there exists a path from s to v and a path v to t. Any arc $(v_1, v_2) \in A$ is called a control flow arc.

A deterministic control flow graph can be regarded as a special case of nondeterministic control flow graphs where nondeterministic vertex set N is the empty set.

Definition 6. A nondeterministic parallel control flow net (CFN) is a 10-tuple $(V, N, P_F, P_J, A_C, A_N, A_{P_F}, A_{P_J}, s, t)$, where $(V, A_C, A_N, A_{P_F}, A_{P_J})$ is a simple arc-classified digraph such that $A_C \subseteq V \times V$, $A_N \subseteq N \times V$, $A_{P_F} \subseteq P_F \times V$, $A_{P_J} \subseteq V \times P_J$, $N \subset V$ is a set of elements, called nondeterministic selection vertices, $P_F \subset V(N \cap P_F = \phi)$ is a set of elements, called parallel execution fork vertices, $P_J \subset V(N \cap P_J = \phi$, and $P_F \cap P_J = \phi)$ is a set of elements, called parallel execution join vertices, $s \in V$ is a unique vertex, called start vertex, such that indegree $(s) = 0$, $t \in V$ is a unique vertex, called termination vertex, such that out-degree $(t) = 0$ and $t \neq s$, and for any $v \in V(v \neq s, v \neq t)$, there exists at least one path from s to v and at least one path from v to t. Any arc $(v_1, v_2) \in A_C$ is called a sequential control arc, any arc $(v_1, v_2) \in A_N$ is called a nondeterministic selection arc, and any arc $(v_1, v_2) \in A_{P_F} \cup A_{P_J}$ is called a parallel execution arc.

The nondeterministic parallel control-flow net differs primarily from the control flow graph (CFG for short) in that it has vertices to represent nondeterministic selection points in a concurrent program, vertices to represent those points in a concurrent program at which a control flow forks into several control flows that can progress in parallel, vertices to represent those points in a concurrent program at which several parallel control flows join into a single control flow, and nondeterministic selection and parallel execution arcs. Therefore, the CFG can be regarded as a special case of CFN where N, P_F, P_J, A_N, A_{P_F}, and A_{P_J} are the empty set, respectively.

Definition 7. Let u and v be any two vertices in a CFN. u forward dominates v iff every path from v to t contains u; u properly forward dominates v iff u forward dominates v and $u \neq v$; u strongly forward dominates v iff u forward dominates v and there exists an integer $k(k \geq 1)$ such that every path from v to t whose length is greater than or equal to k contains u; u is called the immediate forward dominator of v iff u is the first vertex that properly forward dominates v in every

path from v to t; u is called the last continuous forward dominator of v iff u is the vertex such that any vertex in the path from v to u forward dominates v but a successor of u does not forward dominate v.

Definition 8. A nondeterministic parallel definition-use net (DUN) is a 7-tuple $(N_C, \Sigma_V, D, U, \Sigma_C, S, R)$, where $N_C = (V, N, P_F, P_J, A_C, A_N, A_{P_F}, A_{P_J},$
$s, t)$ is a CFN, Σ_V is a finite set of symbols, called variables, $D : V \to P(\Sigma_V)$ and $U : V \to P(\Sigma_V)$ are two partial functions from V to the power set of Σ_V, Σ_C is a finite set of symbols, called channels, and $S : V \to P(\Sigma_C)$ and $R : V \to P(\Sigma_C)$ are two partial functions from V to the power set of Σ_C.

The nondeterministic parallel definition-use net differs primarily from the defini tion-use graph (DUG) in that it is defined based on the CFN rather than the CFG and each of its vertices corresponding statements concerned with interprocess synchronization and communication is classified with two sets of channels, i.e., the set of channels used at the corresponding statement for sending messages and the set of channels used at the corresponding statement for receiving messages. Therefore, the DUG can be regarded as a special case of DUN where N, P_F, P_J, A_N, A_{P_F}, A_{P_J}, Σ_C, S, and R are the empty set, respectively.

Note that the above definitions of CFN and DUN are graph-theoretical, and therefore, they are independent of any particular programming language [15].

3.2 Definition-Use Nets of Ada 2012 Programs

Here, we illustrate how to represent Ada 2012 programs by Definition-Use Nets as above definitions, such that convert mathematical symbols to actual graphics.

As above definitions, DUNs of Ada 2012 programs consist of vertices, types of vertices, labels of vertices, and arcs. We map all kinds of statement concerned with DUNs to vertices in Table 3.1, and then illustrate every representation of the vertex corresponding to the term Num of examples in Table 3.2.

3.2.1 Vertices of DUNs of Ada 2012 Programs

A vertex of definition-use nets for Ada 2012 represents a declaration, a definition, a control predicate corresponding to an expression, and a statement [7, 8, 40, 41], respectively. Every representation has some subordinates as the following Table 3.1. Every vertex represents take one type and one label corresponding to a description for Ada 2012 programs on account of Table 3.1.

3.2.2 Types of Vertices of DUNs of Ada 2012 Programs

As above definitions, there are five types of special vertices as nondeterministic selection vertices N_s, parallel execution fork vertices P_f, parallel execution join vertices P_j, start vertex Start, termination vertex Termination, and an ordinary types of the others, as the term Type in Table 3.1.

In Ada 2012, the nondeterministic selection vertex just appears as a select accept statement, or a conditional entry call statement, or a timed entry call

Table 3.1: Vertices of DUNs of Ada 2012 Programs

Representation	Subordinate	Type	Label	Example
Declaration	variable declaration	Ordinary	D(v)	A-2
	procedure body declaration	P_f	n/a	A-29
		P_j	n/a	A-31
		S	n/a	A-1
		T	n/a	A-31
	task body declaration	Ordinary	S(v)	A-9,14,18
		Ordinary	R(v)	A-30
Definition	aspect specification	Ordinary	U(v)	B-2
Control predicate	function call	Ordinary	U(v)	C-1
		Ordinary	U(v)	D-1
	in membership test	Ordinary	U(v)	E-1
	not in membership test	Ordinary	U(v)	F-1
	case expression	Ordinary	U(v)	G-1
	if expression	Ordinary	U(v)	G-4
	for all quantified expression	Ordinary	U(v)	H-1
	for some quantified expression	Ordinary	U(v)	I-1
Statement	assignment statement	Ordinary	D(v)	A-21
		Ordinary	U(v)	A-21
	while loop statement	Ordinary	U(v)	J-1
	for loop statement	Ordinary	D(v)	K-1
	procedure call statement	Ordinary	U(v)	L-1
	entry call statement	N_s	n/a	M-1
		N_s	n/a	N-1
		Ordinary	D(v)	A-15
		Ordinary	U(v)	A-10
		Oridinary	S(v)	A-10,15
		Oridinary	R(v)	A-11,15
	extend return statement	Oridinary	D(v)	O-1
	accept statement	N_s	n/a	A-19
		Ordinary	D(v)	A-20
		Ordinary	U(v)	A-26
		Ordinary	R(v)	A-20,24
		Ordinary	S(v)	A-22,26
	requeue statement	N_s	n/a	P-1
		Ordinary	D(v)	Q-1
		Ordinary	U(v)	R-1

Table 3.2: Examples of DUNs of Ada 2012 Programs

Num	Example	Type/Label
A-1	**procedure** Parent **is**	Start
A-2	X,Y:Integer :=1;	D(2)={X,Y}
A-3	**task** T0 **is**	
A-4	**entry** E1(I : **in** Integer);	
A-5	**entry** E2(O : **out** Integer);	
A-6	**end** T0;	
A-7	**task** T1; **task** T2;	
A-8	**task body** T1 **is**	
A-9	**begin**	S(9)={Parent}
A-10	T0.E1(Y);	U(10)={Y},S(10)={T0.E1}
A-11	Put_Line("This is Child T1.");	R(11)=T0.E1.END!
A-12	**end** T1;	
A-13	**task body** T2 **is**	
A-14	**begin**	S(14)={Parent}
A-15	T0.E2(X);	D(15)={X},S(15)={T0.E1}
A-16	**end** T2;	R(16)={T0.E2.END!}
A-17	**task body** T0 **is**	
A-18	**begin**	S(18)={Parent}
A-19	**select**	N_s
A-20	**accept** E1(I : **in** Integer) **do**	D(20)={I},R(20)={T0.E1}
A-21	Y:=I*I;	D(21)={Y},U(21)={I}
A-22	**end** E1;	S(22)={T0.E1.END!}
A-23	**or**	
A-24	**accept** E2(O : **out** Integer) **do**	R(24)={T0.E2}
A-25	O:=X;	U(26)={O}
A-26	**end** E2;	S(26)={T0.E2.END!}
A-27	**end** select;	
A-28	**end** T0;	
A-29	**begin**	P_f
A-30	Put_Line("This is Parent.");	R(30)={Parent}
A-31	**end** Parent;	$P_j, Termination$

Table 3.3: Examples of DUNs of Ada 2012 Programs (Continued)

Num	Example	Type/Label
B-1	**function** Sqrt (I : Integer) **return** Integer	
B-2	**with** Precondition => I = 0;	U(2)={I}
C-1	Function_Name(X);	U(1)={X}
D-1	**if** N = 1 **and** Y < 1 **then**	U(1)={N,Y}
E-1	**if** X **in** 0.5 .. Z | 2.0*Z .. 10.0 **then**	U(1)={X,Z}
F-1	**if** M **not in** Mon .. Fri **then**	U(1)={M}
G-1	Trans_fares :=(**case** Transport **is**	U(1)={Transport}
G-2	**when** Train | Metro => 140),	
G-3	**when** Bus => (**if** Mile <= 100 **then**	
G-4	(**if** Mile > 70 **then** 220 **else** 180);	U(4)={Mile}
H-1	B :=(**for all** I **in** M'**Range** => M(I)='C');	U(1) = {I}
I-1	B :=(**for some** I **in** K'**Range** => K(I)=10);	U(1) = {I}
J-1	**while** X < 10 **loop**	U(1)={X}
K-1	**for** I **in** Integer **range** 100 .. 200 **loop**	D(1)={I}
L-1	Procedure_Name(X);	U(1) = {X}
M-1	**select**	N_s
M-2	C.Rendezvous	
M-3	**else**	
M-4	Put_Line("Else");	
M-5	**end select**;	
N-1	**select**	N_s
N-2	C.Rendezvous	
N-3	**or**	
N-4	delay 5.0;	
N-5	**end select**;	
O-1	**return** R: Integer **do**	D(1)={R}
P-1	**requeue** Entry1 **with abort**;	N_s
Q-1	**requeue** E(O: **out** Integer);	D(1)={O}
R-1	**requeue** E(I: **in** Integer);	U(1)={I}

statement, or a requeue statement with abort as the Num A-19, M-1, N-1, P-1 in Table 3.2, respectively.

The parallel execution fork vertex represents a parallel execution branch starting vertex. For Ada 2012 programs, if there are a few tasks declared in a block, the block is called these tasks' parent block. After elaborating the declarative_part of the parent block, the tasks start to be activated. The initial part of the execution of the task body is referred to as the activation of the task, which consists of the elaboration of the declarative_part of the task body [37, 38, 39, 40]. Therefore, a parallel execution branch starting vertex always represents the statement immediately following the declaration part of the parent block, such that it is the reserved word **begin** of the parent block body. For example, the parallel execution branch starting vertex appears on line A-29 in the Parent example program of Table 3.2.

The parallel execution join vertex represents a parallel execution branch confluence vertex. In Ada 2012 programs, after all statements of the task body are completed, the parent block frees up the space of local variables and then exits. Even all the statements of the parent block body are earlier completed, it must wait for the completion of all the statements of the task body. In such a case, the task body is dependent on the parent block [37, 38, 39, 40]. Hence, a parallel execution branch join vertex always represents the statement immediately following the last statement of the parent block body, such that it is the statement with the reserved word **end** of the parent block, like the line A-31 of the above example program Parent.

The start vertex of DUNs for Ada 2012 programs represents the program unique start vertex, such that it is always the start vertex of the outermost parent block, if any. In Ada 2012 programs, it appears as the first statement of a procedure body, or a package body, for example, the line A-1 of the example program Parent.

Similarly, the termination vertex of DUNs for Ada 2012 programs represents the program unique termination vertex, such that it is always the termination vertex of the outermost parent block, if any. In Ada 2012 programs, it appears as the last statement of a procedure body declaration, or a package body declaration, i.e., it has the reserved word **end**. Note that sometimes the termination vertex and the parallel execution join vertex is the same vertex, but still unique vertex, for instance, the line A-31 of the example program Parent is also a termination vertex, moreover it is the unique termination vertex of the program.

3.2.3 Types of Labels of Vertices of DUNs of Ada 2012 Programs

The vertices of the DUNs are labeled with information about definitions and/or uses of the values of variables and inter-process synchronizations and communications as above definitions [37, 38, 39, 40, 42, 44].

D(v) is the set of all variables defined at v. In Ada 2012 programs, the set of variables defined represents as follows:

- a variable that appears on the left of an assignment statement, such as A-21 of Table 3.2

- a variable that is declared by a declaration statement excluding a constant declaration statement, such as A-2
- a variable that is actually referred to a formal parameter with **in** mode, or **in out** mode in an accept statement, such as A-20
- a variable that is actually referred to an actual parameter with **out** mode, or **in out** mode in an entry call statement without protected, such as A-15
- a variable that is actually referred to a loop parameter in a for loop statement, such as K-1
- a variable that is actually referred to a formal parameter with **out** mode, or **in out** mode in a requeue statement, such as Q-1
- a variable that appears in an extended return statement, such as O-1

U(v) is the set of all variables used at v. In Ada 2012 programs, the set of variables used represents as follows:

- a variable that appears on the right of an assignment statement, such as A-21 in Table 3.2
- a variable that appears in a function call expression, such as C-1
- a variable that appears in a procedure call expression without containing any parameters, such as L-1
- a variable appears in the control predicate of conditional branches, such as D-1
- a variable that is actually referred to a formal parameter with **out** mode, or **in out** mode, appears on the last statement represent end of an accept statement, such as A-26
- a variable that is actually referred to an actual parameter with **in** mode, or **in out** mode in an entry call statement without protected, such as A-10
- a variable that is actually referred to a formal parameter with **in** mode, or **in out** mode in a requeue statement, such as R-1
- a variable appears on the right of a Boolean expression, such that it appears on the right of the symbol => of an aspect specification, such as Precondition, Postcondition, Type_Invariant, Static_Predicate, Dynamic_Predicate, even Dispatching_Domain, such as B-2
- a variable appears in some expressions, such as a membership test expression, a case expression, an if expression, a for all quantified expression, a for some quantified expression, such as E-1, F-1, G-1, G-4, H-1, and I-1 in Table 3.2

S(v) is the set of symbols of the send messages functions at v.
R(v) is the set of symbols of the receive messages functions at v.
In Ada 2012 programs, after the activation of the task is completed, the statements of the parent block begin to execute. Therefore, the statement immediately

following the declarative_part of the task body, i.e., a vertex representing the reserved word **begin** of the task body, is labeled with send message function S(v), such as A-9, A-14, A-18 in Table 3.2, while the first statement of the parent block, i.e., the first statement under the **begin** of the parent block, such as A-30, is labeled with receive message function R(v), which is meaning that the child tasks complete the activations. The name of the symbol is the same as the parent block name.

Also, if there is an entry call to another task, the entry call statement is labeled with S(v) representing send message function, such as A-10, A-15 in Table 3.2, the accept statement corresponding to the entry call statement is labeled with R(v) representing receive message function, such as A-20 and A-24. Whereas, the vertex that represents **end** of the accept statement is labeled with S(v) sending message function, such as A-22 and A-26, to the next statement of the entry call statement, which is labeled with receiving message function R(v), such as A-11 and A-16. The symbol's name is the name of the entry call, representing that the rendezvous of this entry call is ending. The next statement can execute.

Furthermore, if an Ada 2012 program has some subprogram calls, i.e., function calls or procedure calls [40], we should extend the DUN with labels, such as:

- A_{in} represents the set of all parameters with mode in at v that is the subprogram call statement
- A_{out} represents the set of all parameters with mode out or in out at v that is the subprogram call statement
- F_{in} represents the set of all parameters with mode in at v that is the start vertex of the subprogram, i.e., callee function or procedure
- F_{out} represents the set of all parameters with mode out or in out at v that is the start vertex of the subprogram, i.e., a callee function or procedure

3.2.4 Arcs of DUNs of Ada 2012 Programs

In DUNs for Ada 2012 programs, the arcs represent several types of possible transfers of control between vertices. The two vertices have one or more types of transfers of control.

3.2.5 Types of Arcs of DUNs of Ada 2012 Programs

As above definition 2.6 and definition 2.7, there are three types of arcs, sequential control arc, nondeterministic selection arc, and parallel execution arc.

The sequential control arc (A_C) represents the control is transferred sequentially between a sequence of statements in a block, or a task, or a procedure.

The nondeterministic selection arc (A_N), always directs from the nondeterministic selection vertex to every select alternatives. There are at least two select alternatives corresponding to the selective statement, such that a nondeterministic selection vertex has at least two nondeterministic selection arcs adjacent to the select alternatives.

The parallel execution arc represents a control thread diverges from a parallel execution fork vertex to the vertices representing other blocks start, i.e., A_{P_F}; Or represents several control threads are confluence to a parallel execution join vertex, i.e., A_{P_J}.

Especially, synchronization channels represent a channel of $S(v)$ sending message function to $R(v)$ receiving message function. Though in essence they do not belong to control transfers, we still use arcs to depict them to express synchronization between vertices.

Also, for the scenario that there are some subprogram calls in an Ada 2012 program, the DUN should represent the interprocedural relationships between call sites and callee sites, such that the call arc A_{C_A} is connected from a function call statement/procedure call statement to the start statement of function body corresponding to its call/the start statement of procedure corresponding to its call.

An example of an Ada 2012 program is given as the following example program Example, and its DUN is shown in Figure 3.2. Program Example is a typical Ada 2012 program. The vertex number represents the line number of the actual program corresponding to the statement. The number following a variable expresses the number of the variable in a compilation unit. The same variable uses the same number. In this Example, procedure Example is the parent of task T and block Inner. Line 18 is fork vertex to the vertex line 9 of the beginning of the activation of task body T, labeling parallel execution arc. After the activation of task body T, the vertex line 11 is sending a message to the vertex line 19 of the execution of the beginning of the parent, labeling synchronization channel. The vertex line 16 is labeled with Join, expressing that control threads are confluence to the termination of the parent. Inside of each block, task T, procedure Example, and block Inner, control flow is just sequential transfers, such that they are labeled with sequential control arc.

```
1  with Ada.Text_IO,Ada.Integer_Text_IO,Ada.Float_Text_IO;
2  use Ada;
3  use Ada.Text_IO,Ada.Integer_Text_IO,Ada.Float_Text_IO;
4  procedure Example is
5     A: constant Integer := 10;  --first "A"
6     B: Integer:=5;       -- first "B"
7     C: Integer;          -- first "C"
8     task T;
9     task body T is
10       B:Integer :=50;
11    begin
12       New_Line;
13       Put("B in task is " );
14       Ada.Integer_Text_IO.Put(B);
15       New_Line;
16    end T;
17
18 begin
19    B := 20;
```

```
20     C := 30;
21     Inner: declare
22        B: Float;              -- second "B"
23        C: Float := 42.0;      -- second "C"
24     begin
25        B := 1.3;           -- local "B"
26        Integer_Text_IO.Put(A);  -- global "A"
27        Float_Text_IO.Put(B);
28        Float_Text_IO.Put(C);
29     end Inner;
30     B := 13;
31     C := 23;
32     Put(B);
33  end Example;
```

Here, we will illustrate how to represent Ada 2012 programs by Definition-Use Nets as above definitions. There has been given an example of an Ada 2012 program in the following program Example, and its DUN is depicted in Figure 3.2. Program Example is a representative Ada 2012 program. The vertex number represents the line number of the actual program corresponding to the statement. The number following a variable expresses the number of the variable in a compilation unit (The text of a program can be submitted to the compiler in one or more compilations. Each compilation is a succession of compilation units. A compilation unit contains either the declaration, the body, or a renaming of a program unit [8]). The same variable uses the same number. In this Example, procedure Example is the parent of task T1 and T2, and it has two subprograms, i.e., function Add and function expression ExpA. Line 67 is fork vertex to the vertices line 45 and line 61 corresponding to the beginning of the activation of task body T1 and T2 respectively, both labeling parallel execution arc. After the activation of task body T1 and T2, the vertex line 46 and 63 are sending a message to the vertex line 68 of the execution of the beginning of the parent, labeling synchronization channel, respectively. The vertices line 59 and line 65 are labeled with Join, expressing that control threads are confluence to the termination of the parent. For task T1, task T2, procedure Example, and subprogram function Add, control flow is just sequential transfers, such that they are labeled with sequential control arc in their own internal.

```
1  with Ada.Text_IO;
2  use Ada.Text_IO;
3  with Ada.Integer_Text_IO;
4  use Ada.Integer_Text_IO;
5  with Ada.Synchronous_Barriers;
6  use Ada.Synchronous_Barriers;
7
8  Procedure Example is
9
10    NT : constant :=2;
11    SB : Synchronous_Barrier (NT);
```

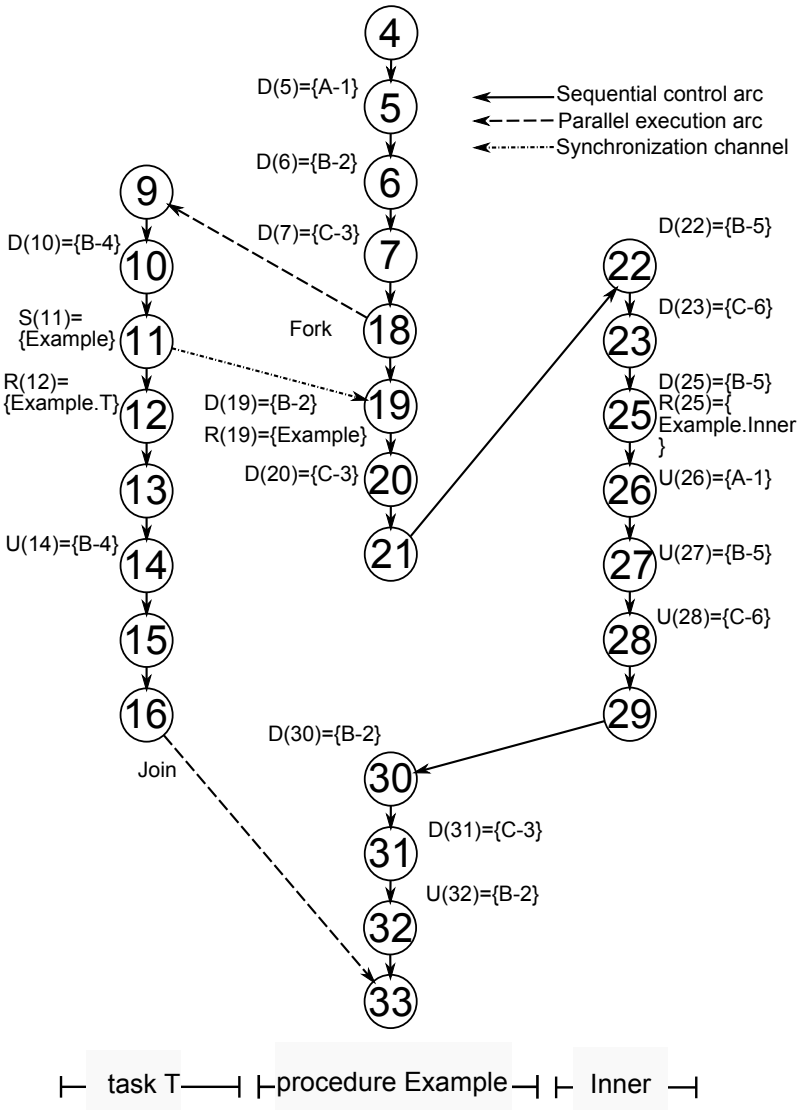

Figure 3.1: A DUN of the Example

```
12    Notified : Boolean := False;
13
14    subtype Single is Integer range 1..50;
15
16    subtype Double is Integer
17       with Dynamic_Predicate =>
18            Double mod 3 =0
19       and then
20            Double / 2 in Single;
21
22    function Add(Arg : in Double) return Integer
23       with Pre => Arg >=10,
24            Post => Add'Result <=100;
```

```
25
26     function Add(Arg : in Double) return Integer is
27     begin
28        return Arg+10;
29     end Add;
30
31     function ExpA(I: Integer) return Integer is
32        (if I = 0 then 1 else ExpA(I-1)*2);
33
34     X, Y: Integer;
35     Z:Integer:=0;
36     Bool :Boolean := False;
37
38     task T1 is
39        entry Start;
40        entry Quit;
41     end T1;
42
43     task T2;
44
45     task body T1 is
46     begin
47        while not Bool loop
48           Wait_For_Release(SB, Notified);
49           select
50              accept Start;
51              Put_Line("Y+Z=" & Integer'Image(Y+Z));
52           or
53              accept Quit;
54              Bool :=True;
55           or
56              terminate;
57           end select;
58        end loop;
59   end T1;
60
61   task body T2 is
62      K:Integer:=5;
63   begin
64      Z:=ExpA(K);
65   end T2;
66
67 begin
68    loop
69        Wait_For_Release(SB, Notified);
70        Get(X);
71        exit when X>100;
72        delay 0.5;
73        Y:=Add(X);
```

```
74          T1.Start;
75      end loop;
76      T1.Quit;
77  end Example;
```

In addition, There are two entry calls in the procedure Example, such as Start and Quit. The entry call statements 74 and 76, both are labeled with $S(v)$ representing sending messages, and the two accept statements corresponding to the two entry calls statement respectively, are labeled with $R(v)$ representing receiving messages, such as line 50 and line 53 in the task body T1. Whereas, the vertex that represents the finish of the accept statement is labeled with $S(v)$ sending messages, such as line 51 and line 54, to the next statement of the entry call statement, which is labeled with receiving message function $R(v)$, such as line 75 and line 77. The symbol's name is the name of the entry call, representing that the rendezvous of this entry call is ending. The next statement can execute.

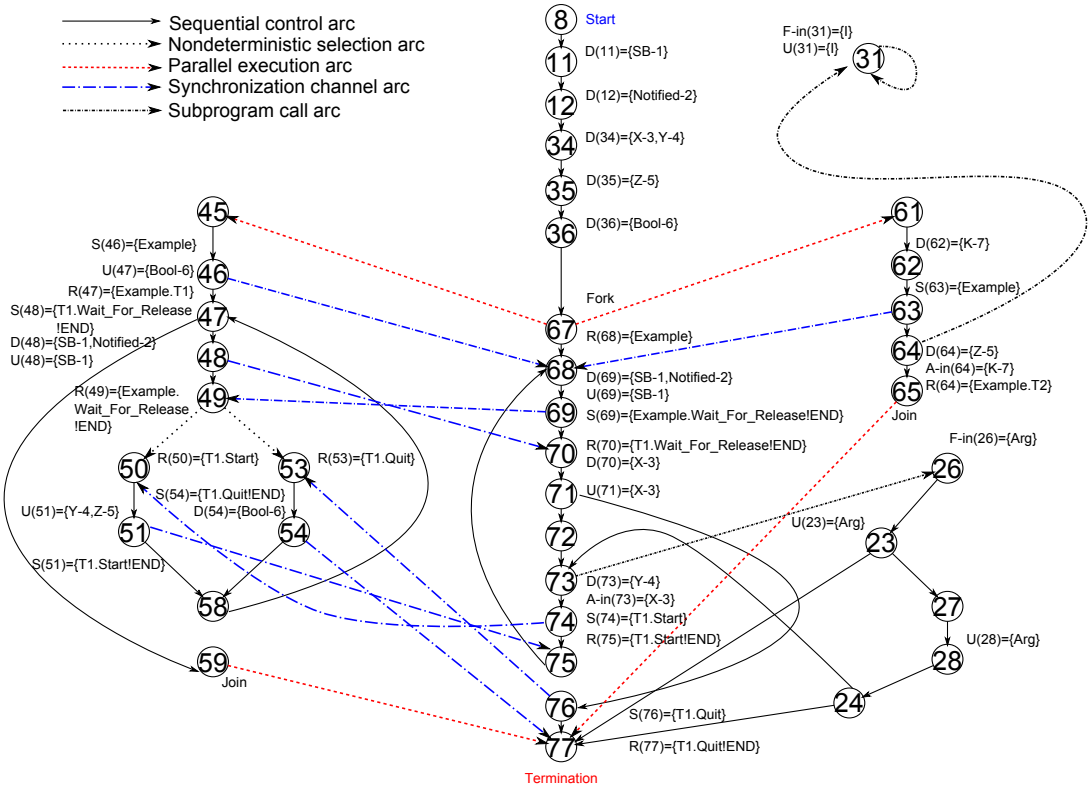

Figure 3.2: The DUN of the Program Example

3.2.6 New Concurrent Facilities of DUNs of Ada 2012 Programs

Ada 2012 introduces new facilities for concurrency, such as Enqueue, Dequeue, and Synchronous barriers and so on [40]. There are DUNs representing protected

object and requeue with abort for concurrent Ada 95 programs in [44]. By the DUN, new concurrent facilities of Ada 2012 are capable of being represented.

If a queue object has a bounded capacity and the number of existing elements in the queue equals the capacity, then a procedure call for Enqueue (implemented as overriding entry) will result in that the task calling Enqueue is blocked until storage becomes available. If a queue is empty, then a procedure call for Dequeue (implemented as overriding entry) will result in that the task calling Dequeue is blocked until an item becomes available. Enqueue waiting may be solved by other tasks calling Dequeue. Similarly, Dequeue waiting may be resolved by other task calling Enqueue [40].

Therefore, like a pair of entry call and accept statement, there is a synchronization channel from the vertex labeled with $S(v)$ representing Enqueue statement, to the vertex labeled with $R(v)$ representing the end of the Dequeue statement, and the symbol is the name of the queue; there is a synchronization channel from the vertex labeled $S(v)$ representing Dequeue statement to the vertex labeled with $R(v)$ representing the end of the Enqueue statement. For example, Figure 5.1 represents the DUN of the following program Task_Queue.

```
1  with Ada.Containers.Synchronized_Queue_Interfaces;
2  with Ada.Containers.Bounded_Synchronized_Queues;
3  with Ada.Text_IO;
4  use Ada.Text_IO;
5
6  procedure Task_Queue is
7     type Queue_Element is new String(1..0);
8     package String_Queues is new
9       Ada.Containers.Synchronized_Queue_Interfaces
10       (Element_Type => Queue_Element);
11    package String_Priority_Queues is new
12     Ada.Containers.Bounded_Synchronized_Queues
13     (Queue_Interfaces => String_Queues,
14     Default_Capacity => 1);
15
16    Q1, Q2 : String_Priority_Queues.Queue;
17    Elem : Queue_Element;
18
19    task T1;
20    task T2;
21
22    task body T1 is
23       begin
24          loop
25             delay 1.0;
26             Q1.Enqueue(Elem);
27             Put_Line("T1");
28       end loop;
29    end T1;
30    task body T2 is
```

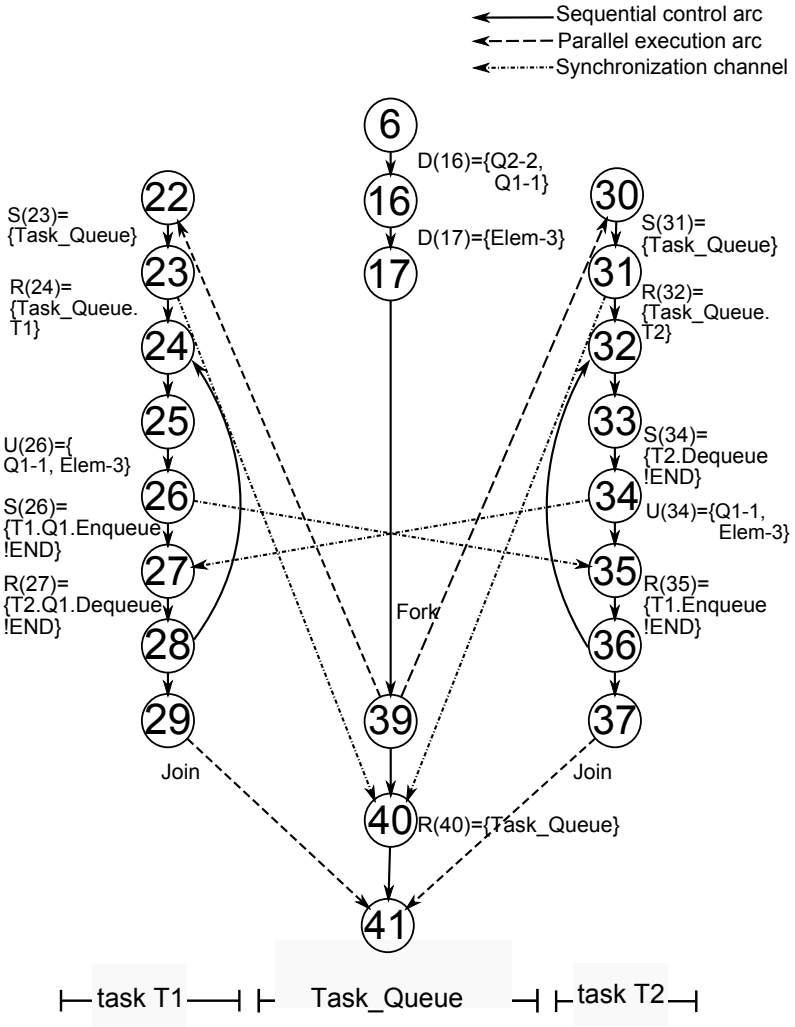

Figure 3.3: The DUN of the Task_Queue

```
31          begin
32              loop
33                  delay 1.0;
34                  Q1.Dequeue(Elem);
35                  Put_Line("T2");
36          end loop;
37      end T2;
38
39  begin
40      null;
41  end Task_Queue;
```

In this Task_Queue, procedure Task_Queue is the parent of task T1 and task T2. Line 39 is fork vertex to the vertex line 22 of the beginning of the activation of task body T1 as well as the vertex line 30 of the beginning of the activation of task body T2, labeling parallel execution arcs. After the activation of task body

T1, the vertex line 23 is sending a message to the vertex line 40 of the execution of the beginning of the parent, labeling synchronization channel, similarly, from line 31 to line 40, there is a synchronization channel. The vertex line 29 and line 37 is labeled with Join, expressing that control threads are confluence to the termination of the parent. Inside of each block, task T1, procedure Task_Queue, and T2, control flows are just sequential transfers, such that they are labeled with sequential control arc. Especially, an Enqueue statement is inside task body T1 the opposite to a Dequeue statement in task body T2. Both are a pair of queue related operations of Q1. As above, There are two synchronization channels about between two tasks. Line 26 is sending a message that is meaning the enqueue operation of Q1 is ending, to the line 35 that is receiving this message and the statement is starting to execute. In the same way, Line 34 is labeled with sending the message that the dequeue operation of Q1 is ending, while line 27 is receiving it and starting to execute.

On the other hand, Ada 2012 provides a language-defined package to synchronously release a group of tasks after the number of blocked tasks reaches a specified count value. Each call to Wait_For_Release blocks the calling task until the number of blocked tasks associated with the Synchronous_Barrier object is equal to Release_Threshold, at which time all blocked tasks are released. Barrier-release waiting is that tasks calling Wait_For_Release are blocked by other possible tasks calling it until the number of blocked tasks associated with the Synchronous Barrier object is equal to threshold [40].

Therefore, there is a synchronization channel from the vertex labeled with $S(v)$ representing Wait_For_Release statement, to the vertex labeled with $R(v)$ representing the end of the Wait_For_Release statement in another task bodies, and the symbol is the name of the task's Wait_For_Release; For example, Figure 3.4 represents the DUN of the following program Task_Barriers.

```
1  with Ada.Text_IO, Ada.Synchronous_Barriers;
2  use Ada.Text_IO, Ada.Synchronous_Barriers;
3
4  procedure Task_Barriers is
5    NT:constant:=2;
6    SB:Ada.Synchronous_Barriers.Synchronous_Barrier(NT);
7    Notified: Boolean:= False;
8    task T1;
9    task T2;
10
11    X:Integer:=0;
12    Y:Integer:=1;
13
14    task body T1 is
15    begin
16       loop
17          delay 1.0;
18          Wait_For_Release(SB, Notified);
19          X:=X+1;
```

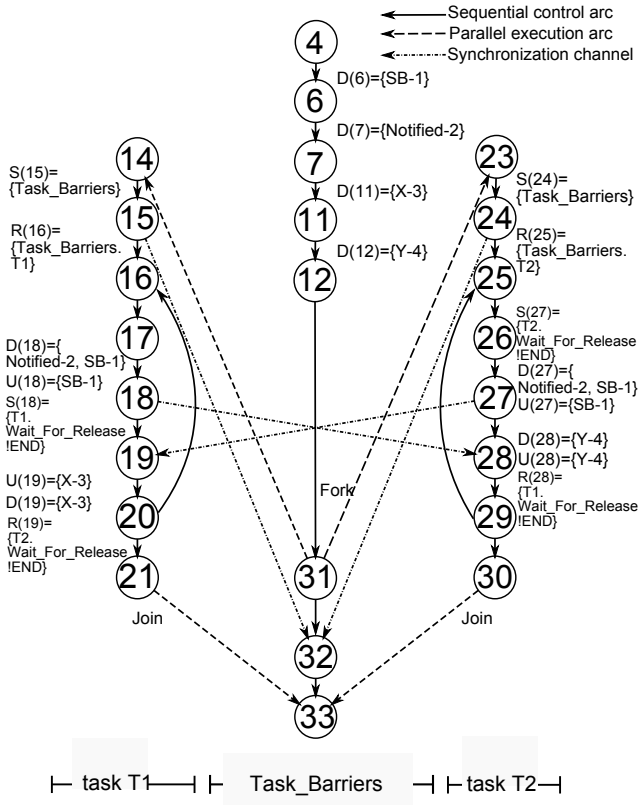

Figure 3.4: The DUN of the Task_Barriers

```
20          end loop;
21      end T1;
22
23      task body T2 is
24      begin
25          loop
26              delay 1.0;
27              Wait_For_Release(SB, Notified);
28              Y:=Y+1;
29          end loop;
30      end T2;
31  begin
32      null;
33  end Task_Barriers;
```

In this Task_Barriers, procedure Task_Barriers is the parent of task T1 and task T2. There are two synchronization channels about between two tasks. Line 18 is sending a message that is meaning the number of blocked tasks associated with the Schronous Barrier object is equal to the threshold, to the line 28 that is receiving this message and the statement is starting to execute. Here, the threshold is set to 2. In the same way, Line 27 is labeled with sending the message that the blocked tasks are released, while Line 19 is receiving it and starting to execute.

Chapter 4

A Definition-Use Net Generator for Ada 2012 Programs

4.1 Generation Method of DUNs of Ada 2012 Programs

4.1.1 Requirement Analyses of DUNs of Ada 2012 Programs

In order to generate a DUN for an Ada 2012 program automatically, we compute vertices, labels, and arcs as described in Section 3, such that they are regarded as the output of the DUN generator. For vertices, we just need to judge what statement the line number is. A DUN generator should satisfy the following requirements.

R1: the DUN generator should judge what statement every line represents in an Ada 2012 programs, as Table 3.1. A program consists of various statements. Each of them represents a declaration, or a definition, or a control predicate, or a parameter, or an execution statement, or an exception handler, or a path, as mentioned in section 3.1. This demands the DUN generator has to distinguish what every line represents, i.e., what every vertex represents.

R2: the DUN generator should find five types of special vertices, if any, that is, nondeterministic selection vertices, parallel execution fork vertices, parallel execution fork vertices, parallel execution join vertices, start vertex, and termination vertex, as described in section 3.2.

R3: the DUN generator should distinguish types of labels of vertices as the same as that section 3.3 described, to get information about the set of variables defined at a vertex D(v), the set of variables used at the vertex U(v), the set of the symbols of the send messages function S(v), and the set of symbols of the receive messages function R(v).

R4: the DUN generator should depict transfers of control between vertices, i.e., getting information about control flows, which are the ends of the arc.

R5: the DUN generator should distinguish types of arcs of DUNs as section 3.5 mentioned, such as sequential control arc, nondeterministic selection arc, parallel

execution arc, and even synchronization channels.

R6: the DUN generator should judge parent-child relationships among tasks.

4.1.2 Generation Algorithms of DUNs of Ada 2012 Programs

For the sequential arcs, they just appear in every task, block, procedure, such that they are identical with those in a sequential program, i.e., control flow graph. The algorithm is in [33]. We design algorithms to computer labels and other three types of arcs of DUNs in the following Algorithm 1, Algorithm 2, and Algorithm 3.

Algorithm 1 Compute labels

```
Input vertices
Output labels of vertices
for every verterx v do
    if v has variables then
        if the variables in D(v) then
            label D(v)=the variables'name {a variable definition}
        end if
        if the variables in U(v) then
            label U(v)=the varialbes'name {a variable use}
        end if
    end if
end for
```

Algorithm 2 Compute nondeterministic selection arcs

```
Input compilation units
Output nondeterministic selection arcs
for every task/block/procedure T do
    if T has select statements then
        for every vertex v as every select statement of T do
            connect a nondeterministic selection arc to from v to the first statement of every select alternative
            corresponding to v
        end for
    end if
    if T has requeue with abort then
        for every vertex v as every a requeue with queue of T do
            connect a nondeterministic selection arc from v to the entry call statement corresponding to v
            {requeue to the corresponding entry call}
            connect a nondeterministic selection arc from v to v itself {abort due to an abort or the expiration
            of a delay of the entry call}
        end for
    end if
end for
```

Algorithm 1 shows an algorithm to compute labels of vertices. The input is each of vertices, that is, every statement in an Ada 2012 programs. Algorithm 2 shows an algorithm to compute nondeterministic selection arcs. Algorithm 3 shows an algorithm to compute synchronization channel arcs. The input of Algorithm 2 and Algorithm 3 is a compilation unit. The text of a program can be submitted to the compiler in one or more compilations. Each compilation is a succession of compilation units. A compilation unit contains either the declaration, the body, or a renaming of a program unit [8]. A program unit is either a package, a task unit, a protected unit, a protected entry, a generic unit, or an explicitly declared

Algorithm 3 Compute synchronization channel arcs

```
Input compilation units
Output synchronization channel arcs
for every task/block/procedure T do
    if T has child then
        connect a synchronization channel arc from begin of T to the first statement of T's each child {FORK}
        label S(v) and R(v) to both ends of the arc, respectively
        connect the last statement of T's each child to the last statement of T {JOIN}
        label S(v) and R(v) to both ends of the arc, respectively
    end if
    if T has entry call to another task/block/procedure then
        for every vertex v as every entry call statement of T do
            connect a synchronization channel arc from v to accept statement corresponding to v
            label S(v) and R(v) to both ends of the arc, respectively
            connect a synchronization channel arc from the last statement of accept statement corresponding to
            v to next statement of v
            label S(v) and R(v) to both ends of the arc, respectively
        end for
    end if
    if T has queues then
        for every vertex v as every enqueue statement of a queue do
            connect a synchronization channel arc from v to next statement of the same queue's the dequeue in
            another task/block/procedure corresponding to v
            label S(v) and R(v) to both ends of the arc, respectively
        end for
        for every vertex v as every dequeue statement of the queue do
            connect a synchronization channel arc from v to next statement of the same queue's the enqueue in
            another task/block/procedure corresponding to v
            label S(v) and R(v) to both ends of the arc, respectively
        end for
    end if
    if T has Barriers then
        for every vertex v as every wait_for_release statement of a barrier do
            connect a synchronization channel arc from v to next statement of the same barrier's the
            wait_for_release in another task/block/procedure corresponding to v
            label S(v) and R(v) to both ends of the arc, respectively
        end for
    end if
end for
```

subprogram other than an enumeration literal. Certain kinds of program units can be separately compiled. Alternatively, they can appear physically nested within other program units [8]. Here, we should traverse every task/block/procedure to compute arcs.

4.2 Implementation of a DUN Generator for Ada 2012 Programs

The Ada Semantic Interface Specification (ASIS) [41] is an ISO standard that defines an interface between Ada environments. We developed an ASIS-based tool to cope with syntax and semantics of Ada 2012. The tool can gain considerable information, whose interfaces of ASIS installed as an Ada library. The functions of the DUN generator are to generate DUNs of compilation units in the Ada environment. Figure 4.1 shows a generation flow of DUNs for Ada 2012 programs.

In order to generate DUNs from Ada 2012 programs, there are six functional components in a definition-use generator. The core component, called Ada2DUN, is invoking other five components, DUN_Handler, Gela_Ids, Stacks, V_Strings, and

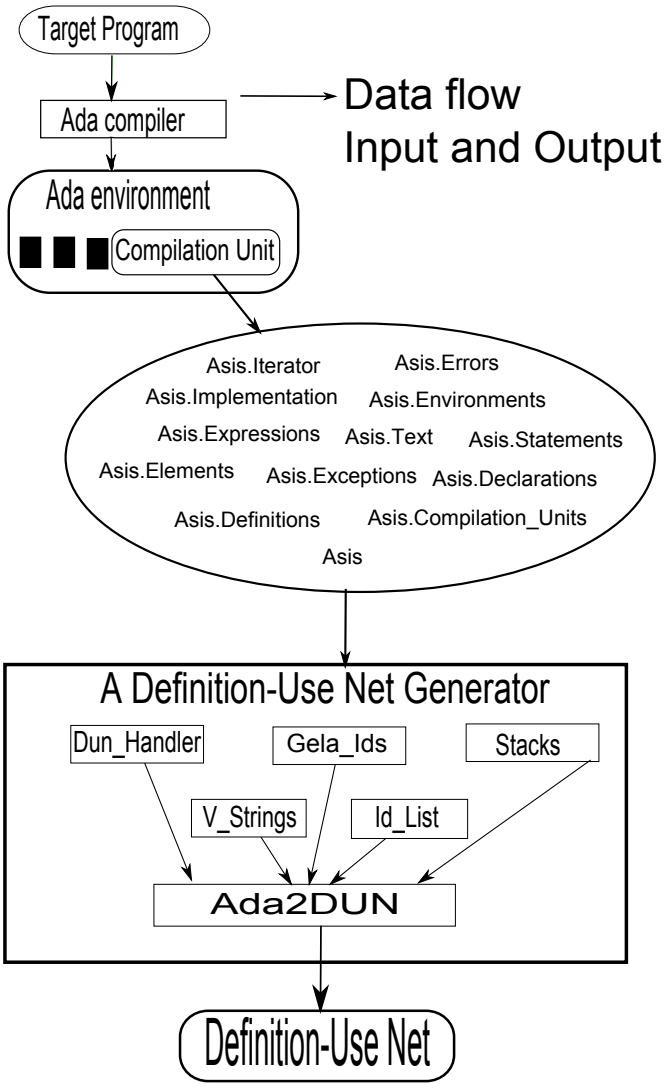

Figure 4.1: A Generation Flow of DUNs for Ada 2012 Programs

Id_List.

The component Ada2DUN can traverse a target Ada 2012 program to generate a DUN.

Component DUN_Handler encapsulates 77 functions and procedures to handle DUNs as above definitions and requirements.

The Gela_Ids [34] encapsulates a set of operations and queries that implement the ASIS Id abstraction. An Id is a way of identifying a particular Element, from a particular Compilation_Unit, from a particular Context. Ids can be written to files. Ids can be read from files and converted into an Element value with the use of a suitable open Context. By using Id, the Gela_Ids uses hash of an element concatenated with unit unique name.

Also, the Stacks encapsulates a generic stack handler package. The V_Strings encapsulates some strings handler functions. Finally, the Id_List supports some

functions for handling a list of Ids.

The generator of DUNs contains the following processes, i.e., an ASIS application must use the following steps [3]:

Step 1: Asis.Implementation.Initialize (...); −− Initialize ASIS;

Step 2: Asis.Ada_Environments.Associate (...); −− Associate ASIS;

Step 3: Asis.Ada_Environments.Open (...); −− Open a context;

Step 4: Get the name of the target unit for the user, and get a compilation unit of the name;

Step 5: Process elements contained in each compilation unit and generate DUN;

Step 6: Asis.Environments.Close (...); −− Close the context;

Step 7: Asis.Environments.Dissociate (...); −− Process the whole DUN;

Step 8: Asis.Implementation.Finalize (...); −− Finalize ASIS;

Step 9: Output the DUN as the text file.

4.3 Examples of a DUN Generator for Ada 2012 Programs

We give an output of this generator for the example program of Figure 1. The elapsed time is Clock 58156.807824882—58158.226458761 on Windows 7 with Intel Core i7-860 Processor (2.8GHz, 4 cores, 8threads), and 4 Gbyte memory.

A text output of this generator for the example program of Figure 3.2 as follows.

```
1: <line> b-8 <connect> 2
2: <line> b-11 <connect> 3 <def> Synchronous_Barrier ,
MAIN.Example5.SB-1
3: <line> b-12 <connect> 4 <def> Boolean ,
MAIN.Example5.Notified-2
4: <line> b-35 <connect> 5 <def> Integer ,
MAIN.Example5.Y-4 Integer ,MAIN.Example5.X-3
5: <line> b-36 <connect> 6 <def> Integer ,
MAIN.Example5.Z-5
6: <line> b-37 <connect> 7 <def> Boolean ,
MAIN.Example5.Bool-6
7: <line> b-68 <connect> 8 <fork> 32 21
8: <line> b-17 <receive> MAIN.Example5
9: <line> b-23 <Pre-connect> 11
10: <line> b-24 <Post-connect> 42
37: <line> b-69 <connect> 38
38: <line> b-70 <connect> 39
39: <line> b-71 <connect> 40
40: <line> b-72 <connect> 41 45
41: <line> b-73 <connect> 42
42: <line> b-74 <connect> 43 <use> Integer ,
MAIN.Example5.Add-7 <call> 9 <a-in>
43: <line> b-75 <connect> 44 <send> MAIN.Example5.T1.Start
```

```
44: <line> b-76 <connect> 37
45: <line> b-77 <connect> 46 <send> MAIN.Example5.T1.Quit
46: <line> b-78

11: <line> b-26 <connect> 12 <f-in> Double,
MAIN.Example5.Add.Arg-8
12: <line> b-27 <connect> 13
13: <line> b-28 <connect> 14 <receive> MAIN.Example5.Add
14: <line> b-28 <connect> 15 <return> Integer,
MAIN.Example5.Add-7
15: <line> b-29

16: <line> b-31 <connect> 17 <f-in> Integer,
MAIN.Example5.ExpA.I-10
17: <line> b-32 <connect> 18
18: <line> b-33 <connect> 19 <receive> MAIN.Example5.ExpA
<call> 16 <a-in>
19: <line> b-33 <connect> 20 <use> Integer,
MAIN.Example5.ExpA.ExpA-9 <return> Integer,MAIN.Example5.ExpA.ExpA-9
20: <line> b-34

21: <line> b-46 <connect> 22
22: <line> b-47 <connect> 23 <send> MAIN.Example5
23: <line> b-48 <connect> 24 31 <receive> MAIN.Example5.T1
24: <line> b-49 <connect> 25
25: <line> b-50 <s-connect> 26 28 31
26: <line> b-51 <connect> 27 <receive> MAIN.Example5.T1.Start
27: <line> b-52 <connect> 30
28: <line> b-54 <connect> 29 <receive> MAIN.Example5.T1.Quit
29: <line> b-55 <connect> 30
30: <line> b-59 <connect> 23
31: <line> b-60 <join> 46

32: <line> b-62 <connect> 33
33: <line> b-63 <connect> 34 <def> Integer,
MAIN.Example5.T2.K-11
34: <line> b-64 <connect> 35 <send> MAIN.Example5
35: <line> b-65 <connect> 36 <use> Integer,
MAIN.Example5.ExpA.ExpA-9 <receive>
MAIN.Example5.T2 <call> 16 <a-in>
36: <line> b-66 <join> 46
```

Here, We input an open souce of Ada 2012 programs. Ada Web Server (AWS) is an Ada-based web server, which can be embedded allowing your application to talk with all modern web browsers [4]. We chose Ada Web Server to eveluate the DUN generator, because Ada Web Server is open source and includes all features of Ada 2012 programs. We input 234 source files (167 specification ads files and 67 adb body files) of AWS as the target program, the DUN generator outputed 144 adt tree files, 83 DUN files, and 4328 vertex. We just give a snippet of the

output as following List 4.1.

Listing 4.1: A snippet output of the DUN generator for Ada Web Server

```
4289: <line> b-36 <connect> 4290 <f-in> ptr , Aliased-Attribute_Set ,
MAIN.AWS.LDAP.Thin.Item.Set-2115 int ,MAIN.AWS.LDAP.Thin.Item.Index-2116
4290: <line> b-41 <connect> 4291
4291: <line> b-42 <connect> 4292 <receive> MAIN.AWS.LDAP.Thin.Item
4292: <line> b-42 <connect> 4293 <return> chars_ptr ,MAIN.AWS.LDAP.Thin.Item-2109
4293: <line> b-43

4294: <line> b-49 <connect> 4295 <f-in>
Return_Code ,MAIN.AWS.LDAP.Thin.LDAP_API_ERROR.n-2117
4295: <line> b-50 <connect> 4296
4296: <line> b-51 <connect> 4297 <receive>
MAIN.AWS.LDAP.Thin.LDAP_API_ERROR
4297: <line> b-51 <connect> 4298 <return>
Boolean ,MAIN.AWS.LDAP.Thin.LDAP_API_ERROR-2088
4298: <line> b-52

4299: <line> b-58 <connect> 4300 <f-in>
Return_Code ,MAIN.AWS.LDAP.Thin.LDAP_API_RESULT.n-2118
4300: <line> b-59 <connect> 4301
4301: <line> b-60 <connect> 4302 <receive> MAIN.AWS.LDAP.Thin.LDAP_API_RESULT
<call> 4294 <a-in>
4302: <line> b-60 <connect> 4303 <use>
Boolean ,MAIN.AWS.LDAP.Thin.LDAP_API_ERROR-2088
 <return> Boolean ,MAIN.AWS.LDAP.Thin.LDAP_API_RESULT-2089
4303: <line> b-61

4304: <line> b-67 <connect> 4305 <f-in>
Return_Code ,MAIN.AWS.LDAP.Thin.LDAP_ATTR_ERROR.n-2119
4305: <line> b-68 <connect> 4306
4306: <line> b-69 <connect> 4307 <receive> MAIN.AWS.LDAP.Thin.LDAP_ATTR_ERROR
4307: <line> b-69 <connect> 4308 <return>
chars_ptr ,MAIN.AWS.LDAP.Thin.ldap_err2string-2112
4308: <line> b-70

4309: <line> b-76 <connect> 4310 <f-in>
Return_Code ,MAIN.AWS.LDAP.Thin.LDAP_NAME_ERROR.n-2120
4310: <line> b-77 <connect> 4311
4311: <line> b-78 <connect> 4312 <receive> MAIN.AWS.LDAP.Thin.LDAP_NAME_ERROR
4312: <line> b-78 <connect> 4313 <return>
Boolean ,MAIN.AWS.LDAP.Thin.LDAP_NAME_ERROR-2084
4313: <line> b-79

4314: <line> b-85 <connect> 4315 <f-in>
Return_Code ,MAIN.AWS.LDAP.Thin.LDAP_SECURITY_ERROR.n-2121
4315: <line> b-86 <connect> 4316
4316: <line> b-87 <connect> 4317 <receive> MAIN.AWS.LDAP.Thin.LDAP_SECURITY_ERROR
4317: <line> b-87 <connect> 4318 <return>
int ,MAIN.AWS.LDAP.Thin.ldap_simple_bind-2091
4318: <line> b-88

4319: <line> b-94 <connect> 4320 <f-in>
Return_Code ,MAIN.AWS.LDAP.Thin.LDAP_SERVICE_ERROR.n-2122
4320: <line> b-95 <connect> 4321
4321: <line> b-96 <connect> 4322 <receive> MAIN.AWS.LDAP.Thin.LDAP_SERVICE_ERROR
4322: <line> b-96 <connect> 4323 <return> int ,MAIN.AWS.LDAP.Thin.ldap_msgfree-2106
4323: <line> b-97

4324: <line> b-103 <connect> 4325 <f-in>
Return_Code ,MAIN.AWS.LDAP.Thin.LDAP_UPDATE_ERROR.n-2123
4325: <line> b-104 <connect> 4326
4326: <line> b-105 <connect> 4327 <receive> MAIN.AWS.LDAP.Thin.LDAP_UPDATE_ERROR
4327: <line> b-105 <connect> 4328 <return> int ,MAIN.AWS.LDAP.Thin.ldap_add_s-2097
4328: <line> b-106
```

Table 4.1: The Limitations of the DUN Generator

1	One Compilation unit	no special treatment
2	Multi-compilation units	convert them into one compilation unit
3	Ada standard library	no special treatment
4	Ada application library	put it on the same path as the target program

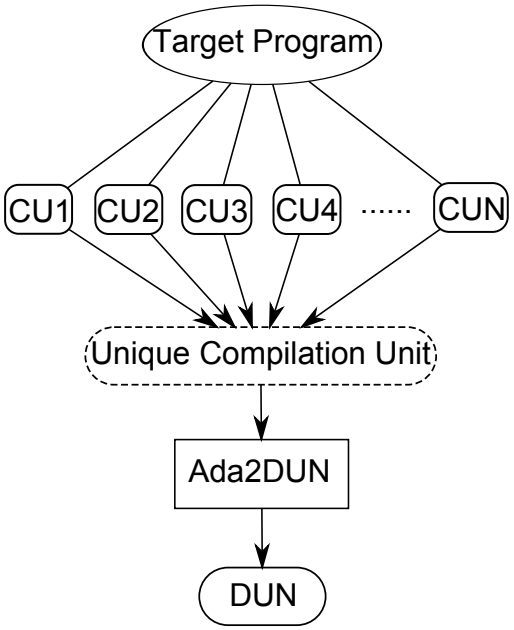

Figure 4.2: A Processing Flow of Compilation Units

4.4 Applications of the DUN Generator

The current DUN generator has two limitations. One is the case that the target program must be one compilation unit. If an Ada 2012 program has various compilation units, the authors have to transform them into one compilation unit, as shown in Figure 4.2. The other is that if the target program is taking some application libraries, which are not Ada standard libraries. The application libraries should be put on the same path as the target program. Table 4.4 shows the comparations of two limitations of the DUN generator. Future, two limitations should be resolved: One is that The DUN generator can process multi-compilation units directly, i.e., the multi-compilation units of the target program are inputted into the DUN generator together, which the merged unique compilation unit is abandoned; Two is that the application libraries can be put on both relative and absolute paths as same as Ada standard libraries.

The DUN of a concurrent program can provide a clear and precise basis for definitions of notions, descriptions of methods, and developments of tools in software engineering. Some applications dependent on the DUN are as follows.

First, we can use it as representation tools in software design and specification

for system engineers to exchange ideas and opinions.

Second, the DUN provides a basis for defining software test coverage criteria and generating software test data.

Third, having the DUN as representations of concurrent programs, many well-know complexity metrics of sequential programs can be redefined for concurrent programs based on the representation [14].

Fourth, the DUN can be used as program understanding tools in software maintenance and reengineering.

Finally, but perhaps most importantly, the DUN provides a clear and precise basis for control flow and data flow analysis of programs, especially for program dependence analysis. Program dependences are determined by control flow and data flow in the program [33, 48]. Since capturing program dependences between statements of a program is indispensable to many software development activities. Furthermore, Task Dependence Net [11, 17, 42] (TDN) is also an arc-classified digraph to explicitly represent the basic program dependences in concurrent Ada programs. Each task can be viewed as a sequential program. There are control and data dependences in each task like ordinary sequential programs. The selection dependences are also in each task. When two or more tasks synchronize and/or communicate with each other, there is a dependence relationship between tasks. Moreover, the paper [53] has presented some new program dependences for Ada 2012, such as precondition dependence, postcondition dependence, predicate dependence, expression dependence, and task-barriers dependence. We can capture various program dependences for representing a data flow for generating program dependences from the DUN in Ada 2012 programs.

Chapter 5

System Dependence Nets of Ada 2012 Programs

5.1 Previous Researches

Based on the CFN and DUN of a concurrent program, Cheng has further proposed Process Dependence Net (PDN) [11, 12], which is arc-classified digraph to represent various program dependences in concurrent programs explicitly. Corresponding to the PDN, Task Dependence Net (TDN) [11, 42, 17] is a version for the PDN of Ada programs, depending on the definitions of Cheng. System Dependence Net [55] is a formal model which can present the program dependences and interprocedural relations in a concurrent program with multiple procedures, such that it is extended from PDN.

5.2 Program Dependences and Interprocedural Relations

5.2.1 Program Dependences

By means of constructing formal representation of DUNs of Ada 2012 programs, we can analyze, identify, and define various types of primary program dependences and interprocedural relations in the program [12]. In this section, based on concurrent Ada 2012 programs, we define five primary program dependences, such as control dependence, data dependence, selection dependence, communication dependence, and synchronization dependence, present four types of interprocedural relations, and then give definitions of new types of program dependences in Ada 2012 programs.

Definition 9. Let (V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_{P_F}, A_{P_J}, A_{C_A}, s, t) be the CFN of a concurrent program, and $u \in V$, $v \in (V - (N \cup P_F \cup P_J \cup F_{in} \cup F_{out}))$ be any two vertices of the net. u is directly strongly control dependent on v iff there exists a path $P = (v_1 = v, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n = u)$ from v to u such that P does not contain the immediate forward dominator of v and there exists no vertex v' in P such that the path from v' to u does not contain

the immediate forward dominator of v'. u is directly weakly control dependent on v iff v has two successors v' and v'' such that there exists a path $P = (v_1 = v, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n = u)$ from v to u and vertex $v_i (1 < i \leq n)$ in P strongly forward dominates v' but does not strongly forward dominate v''.

Not that according to the above definition, if u is directly strongly control dependent on v, then u is also directly weakly control dependent on v, but the converse is not necessarily true.

Informally, if u is directly strongly control dependent on v, then v must have at least two successors v' and v'' such that if the branch from v to v' is executed then u must be executed, while if the branch from v to v'' is executed then u may not be executed. If u is directly weakly control dependent on v, then v must have two successors v' and v'' such that if the branch from v to v' is executed then u is necessarily executed within a fixed number of steps, while if the branch from v to v'' is executed then u may not be executed or the execution of u may be delayed indefinitely. The difference between strong and weak control dependences is that the latter reflects a dependence between an exit condition of a loop and a statement outside the loop that may be executed after the loop is exited, but the former does not.

Definition 10. Let (V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_{P_F}, A_{P_J}, A_{C_A}, s, t) be the CFN of a concurrent program, and $u \in V$, $v \in N$ be any two vertices of the net. u is directly selection dependent on v iff (1) there exists a path $P = (v_1 = v, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n = u)$ from v to u such that P does not contain the immediate forward dominator of v, and (2) there exists no vertex $v_i (1 < i < n)$ in P such that the path from v_i to u does not contain the immediate forward dominator of v_i.

Informally, if u is directly selection dependent on v, then v must have some successors such that if the branch from v to one of the successors is executed then u must be executed, while if another branch is executed then u may not be executed.

The difference between the direct (strong or weak) control dependence and the direct selection dependence is that the former defines a kind of program dependence holding between the control predicate of a conditional branch statement and a statement whether it is executed is determined by the truth value of the control predicate, but the latter defines a kind of program dependence holding between a nondeterministic selection statement and a statement whether it is executed is determined by the nondeterministic selection.

Definition 11. Let $(N_C, \Sigma_V, D, U, \Sigma_C, S, R)$ be the DUN of a concurrent program, where N_C is the CFN (V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_{P_F}, A_{P_J}, A_{C_A}, s, t) of the program, and u and v be any two vertices of the net. u is directly data dependent on v iff there is a path $P = (v_1 = v, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n = u)$ from v to u such that $(D(v) \cup U(u)) - D(P') \neq \phi$ where $D(P') = D(v_2) \cup \cdots \cup D(v_{n-1})$.

Informally, if u is directly data dependent on v, then the value of a variable computed at v has direct influence on the value of a variable computed at u.

Definition 12. Let $(N_C, \Sigma_V, D, U, \Sigma_C, S, R)$ be the DUN of a concurrent program, where N_C is the CFN (V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_{P_F}, A_{P_J}, A_{C_A}, s, t) of the program, and u and v be any two vertices of the net. u is directly synchronization dependent on v iff any of the following conditions holds:

(1) $(v, u) \in A_{P_F} \cup A_{P_J}$, i.e., (v, u) is a parallel execution arc,
(2) $S(v) = R(u)$, or
(3) there exists a vertex v' such that v' directly synchronization dependent on v, u is the last continuous forward dominator of v', and $S(v'') = \phi$ and $R(v'') = \phi$ for any vertex v'' (excluding v') in the path from v' to u.

Informally, if u is directly synchronization dependent on v, then the start and/or termination of execution of v directly determines whether or not the execution of u starts and/or terminates.

The difference between the direct (strong or weak) control dependence and the direct synchronization dependence is that the former is irrelevant to the execution timing of a program but the latter is intrinsically relevant to the execution timing.

Definition 13. Let $(N_C, \Sigma_V, D, U, \Sigma_C, S, R)$ be the DUN of a concurrent program, where N_C is the CFN (V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_{P_F}, A_{P_J}, A_{C_A}, s, t) of the program, and u and v be any two vertices of the net. u is directly communication dependent on v iff there exist two vertices v' and v'' such that u is directly data dependent on v', $R(v') = S(v'')$, and v'' is directly data dependent on v.

Informally, if u is directly communication dependent on v, then the value of a variable computed at v has direct influence on the value of a variable computed at u by an inter-process communication.

The difference between the directly data dependence and the direct communication dependence is that the direct data dependence is irrelevant to communication channels of a program but the direct communication dependence is intrinsically relevant to the channels.

Here, We give some examples to introduce program depdences. For example, in Figure 3.2, vertices 47, 48, 49, and 58 are directly strongly (weakly) control dependent on vertex 47; vertex 59 is directly weakly control dependent on vertex 47 but not directly strongly control dependent on 47; vertices 68, 69, 70, 71, 72, 73, 74, and 75 are directly strongly (weakly) control dependent on vertex 71; vertices 76 and 77 are directly weakly control dependent on vertex 71 but not directly strongly control dependent on 71. Vertices 50, 51, 53, and 54 are directly selection dependent on vertex 49. Vertices 47 is directly data dependent on both vertices 36 and 54; vertices 48 and 69 are directly data dependent on vertex 11; vertex 64 is directly data dependent on vertex 62; vertices 71 and 73 are directly data dependent on vertex 70; vertex 51 is directly data dependent on vertex 35. vertices 45, 61, 62 are directly synchronization dependent on vertex 67; vertex 68 is directly synchronization dependent on both vertices 46 and 63; vertex 70 is directly synchronization dependent on vertex 48; vertex 49 is directly synchronization dependent on 69; vertex 53 is directly synchronization dependent on 76; vertex 50 is directly synchronization dependent on 74; vertex 75 is directly synchronization dependent on 51; vertex 77 is directly synchronization dependent on vertices 54, 59, and 65. vertex 51 is directly communication dependent on vertices 64 and 73.

5.2.2 Interprocedural Relations

To handle interprocedure calling and parameter passing issues, Horwitz et al. modeled System Dependence Graph (SDG) [36]. In Ada programs, there are four types of interprocedural relations [44, 55]:

- if statement v is calling a subprogram, the start statement u of the subprogram is said to be call-related with v. For example, in Figure 3.2, vertex 31 is call-relationed with vertices 64 and 31; vertex 26 is call-relationed with vertex 73
- the formal parameter u labeled with F_{in}, is said to be parameter-in-related with the actual parameter v corresponding to u. For example, in Figure 3.2, vertex 31 is also parameter-in-related with vertices 64 and 31; vertex 26 is call-relationed with 73
- the actual parameter u is said to be parameter-out-related with the formal parameter v labeled with F_{out} corresponding to u
- the parameter/statement u is said to be returned-value-related with a return statement v corresponding to the function call, such that a returned value from v directly affects the variables assigned at u. For example, in Figure 3.2, vertex 64 is also returned-value-related with vertex 31; vertex 73 is returned-value-related with vetex 24

5.2.3 New Types of Program Dependences in Ada 2012 Programs

On account of changes and extensions introduced by Ada 2012, some new types of program dependences have been found, such as precondition dependence, postcondition dependence, predicate dependence, expression dependence, task-barriers dependence.

- Precondition dependence, postcondition dependence, and predicate dependence come from extentions about "Contract-based programming" [7, 8, 40]. In Ada 2012, a precondition is an obligation on the caller to ensure that it is true when the subprogram is called and it is a guarantee to the implementer of the body that it can be relied upon on entry to the body [7]. A postcondition is an obligation on the implementer of the body to ensure that it is true on return from the subprogram and it is a guarantee to the caller that it can be relied upon on return [7]. Similarly, Ada 2012 introduced assertions for types and subtypes with predicates.
- Expression dependence originates from expression functions, which can parameterize an expression without the formality of providing a function body. In this way, we can express a if, case, quantification expression, even a function in one statement that is an expression.

- Task-barriers dependence is considered from a package of Ada.Synchron ous_barriers to make the tasks given to be waited for by using a discriminant and to be released together [40]

However, there are no definitions about the new program dependences in the paper [53], whereby predicate dependences are relevant to types and belong to dependent types [6], and thus they cannot be defined by control flow or data flow, such that there is no predicate dependece in any DUNs. Here, we will give formal definitions of the other four types.

Definition 14. Let $(N_C, \Sigma_V, D, U, \Sigma_C, S, R)$ be the DUN of a concurrent program, where N_C is the CFN (V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_{P_F}, A_{P_J}, A_{C_A}, s, t) of the program, and u and v be any two vertices of the net. u is directly precondition dependent on v iff (1) there exists a path $P = (v_1 = F_{in}, v_2), \ldots, (v_{n-1}, v_n = A_{in})$, (2) v in P is the immediate forward dominator of v_1, and the t of the net is the immediate forward dominator of v, and (3) vertex u in P is directly strongly control dependent on v.

Informally, if u is directly precondition dependent on v, then v must have two successors such that if the branch from v to one of the successors is executed then u must be executed, that is meaning that the requirement of precondition meets, which is true on entry, whereas, on the other branch, v raises a program exception on preconditon, the control flow is running to termination, u is never executed. For example, in Figure 3.2, vertices 27, 28, 24, and 73 are directly precondition dependent on vertex 23.

The differences between the direct (strong or weak) control dependence and the direct precondition dependence is that the former is irrelevant to runtime checks, and must appear in the control flow but the latter is intrinsically relevant to the runtime checks optionally by means of a switch.

Definition 15. Let $(N_C, \Sigma_V, D, U, \Sigma_C, S, R)$ be the DUN of a concurrent program, where N_C is the CFN (V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_{P_F}, A_{P_J}, A_{C_A}, s, t) of the program, and u and v be any two vertices of the net. u is directly postcondition dependent on v iff (1) there exists a path $P = (v_1 = u, v_2), \ldots, (v_{n-1}, v_n = v)$, $u = A_{in}$, and (2) u is the immediate forward dominator of v, and the t of the net is the immediate forward dominator of v.

Informally, if u is directly postcondition dependent on v, then v must have two successors such that if the branch from v to one of the successors is executed then u must be executed, that is meaning that the requirement of postcondition meets, which is true on return, whereas, on the other branch, v raises a program exception on postconditon, the control flow is running to termination, u is never executed. For example, in Figure 3.2, vertex 73 is directly postcondition dependent on vertex 24.

The differences between the direct precondtion dependence and the direct postcondition dependence is that the former is an obligation on the caller to ensure that it is true when the subprogram is called and it is a guarantee to the implementer of the body that it can be relied upon on entry to the body, while the latter is an obligation on the implementer of the body to ensure that it is true on return from the subprogram and it is a guarantee to the caller that it can be

relied upon on return. And furthermore, there exit three vertices v', u and v in a path $P = (v_1, v_2), \ldots, (v_{n-1}, v_n)$ of the DUN, if v' is precondition dependent on v and postcondition dependent on u, then u is directly precondition on v and u is a forward dominator of v.

Definition 16. Let $(N_C, \Sigma_V, D, U, \Sigma_C, S, R)$ be the DUN of a concurrent program, where N_C is the CFN (V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_{P_F}, A_{P_J}, A_{C_A}, s, t) of the program, and v be a vertex of the net. v is directly expression dependent on itself v iff (1) there exists a vertex v, (2) $F_{in}(v) = U(v)$, and (3) $(v, v) \in A_{C_A}$, i.e., (v, v) is a subprogram call arc.

Informally, if v is directly expression dependent on itself v, then v must be an expression function with some control predicates, to process some recursive loop operations. For example, in Figure 3.2, vertex 31 is directly expression dependent on itself.

Definition 17. Let $(N_C, \Sigma_V, D, U, \Sigma_C, S, R)$ be the DUN of a concurrent program, where N_C is the CFN (V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_{P_F}, A_{P_J}, A_{C_A}, s, t) of the program, and u and v be any two vertices of the net. u is directly task-barriers dependent on v iff (1) there exist two different paths P_1 and P_2, and three different vertices u', w, and w', (2) u is the immediate forward dominator of u' in P_1 and w is the immediate forward dominator of w' in P_2, (3) u' and w' are directly data dependent on v, and (4) u is directly synchronization dependent on w' and w is directly synchronization dependent on u'.

Informally, if u is directly task-barriers dependent on v, then v must define a specified count value. After the number of blocked tasks reaches the value, synchronously release a group of tasks. Each call to Wait_For_Release blocks the calling task until the number of blocked tasks associated with the Synchronous Barrier Object is equal to Release Threshold, at which time all blocked tasks are released. For example, in Figure 3.2, vertices 49 and 70 are directly task-barriers dependent on vertex 11, representing that line 49 and line 70 can be executed when the number of synchronous barrier object is equal to Release Threshold difined on line 11.

In Ada 2012 programs, after the activation of the task is completed, the statements of the parent block begin to execute. Therefore, the statement immediately following the declarative_part of the task body, i.e., a vertex representing the reserved word **begin** of the task body, is labeled with send message function S(v), such as A-9, A-14, A-18 in Table 3.2, while the first statement of the parent block, i.e., the first statement under the **begin** of the parent block, such as A-30, is labeled with receive message function R(v), which is meaning that the child tasks complete the activations. The name of the symbol is the same as the parent block name.

5.3 System Dependence Nets of Ada 2012 Programs

From the above, based on graph-theoretical, when we presents the definitions of various types of program dependences and interprocedural relations, we can con-

struct a formal model (SDN) to dipict dependence-based program representation explicitly.

The SDN of a concurrent Ada 2012 program is a kind of arc-classified digraph to represent program dependences and interprocedural relations, such that each type of arc explicitly denotes a type of program dependence or interprocedural relation, as well as each node indicates a statement at both ends of the arc.

In Ada 2012 programs, the SDN is defined as an arc-classified digraph, as (V_{DUN}, Con, Dat, Sel, Syn, Com, Pre, Pos, Exp, Tas, Cal, P_{in}, P_{out}, Ret), where

- $V_{DUN} \in (V \cap A_{in} \cap A_{out} \cap F_{in} \cap F_{out})$ to represent the node set of DUN
- Con is the set of control dependence arcs such that any 2-tuple $(u, v) \in Con$ if and only if u is directly control dependent on v
- Dat is the set of data dependence arcs such that any 2-tuple $(u, v) \in Dat$ if and only if u is directly data dependent on v
- Sel is the set of selection dependence arcs such that any 2-tuple $(u, v) \in Sel$ if and only if u is directly selection dependent on v
- Syn is the set of synchronization dependence arcs such that any 2-tuple $(u, v) \in Syn$ if and only if u is directly synchronization dependent on v
- Com is the set of communication dependence arcs such that any 2-tuple $(u, v) \in Com$ if and only if u is directly communication dependent on v
- Pre is the set of precondition dependence arcs such that any 2-tuple $(u, v) \in Pre$ if and only if u is directly precondition dependent on v
- Pos is the set of postcondition dependence arcs such that any 2-tuple $(u, v) \in Pos$ if and only if u is directly postcondition dependent on v
- Exp is the set of expression dependence arcs such that any 2-tuple $(u, v) \in Exp$ if and only if u is directly expression dependent on v
- Tas is the set of task-barriers dependence arcs such that any 2-tuple $(u, v) \in Tas$ if and only if u is directly task-barriers dependent on v
- Cal is the set of call relation arcs such that any 2-tuple $(u, v) \in Cal$ if and only if u is call related with v
- P_{in} is the set of parameter-in relation arcs such that any 2-tuple $(u, v) \in P_{in}$ if and only if u is parameter-in-related with v
- P_{out} is the set of parameter-out relation arcs such that any 2-tuple $(u, v) \in P_{out}$ if and only if u is parameter-out-related with v
- Ret is the set of returned-value relation arcs such that any 2-tuple $(u, v) \in Ret$ if and only if u is returned-value-related with v

As thus, by using various types of arcs, the SDN of an Ada 2012 program can explicitly represent control dependence, data dependence, selection dependence, synchronization dependence, communication dependence, precondition dependence, postcondition dependence, expression dependence, task-barriers, call-relation, parameter-in-relation, parameter-out-relation, and returned-value-relation, respectively. Figure 5.1 shows the SDN of the Ada 2012 program Example, corresponding to its DUN in Figure 3.2.

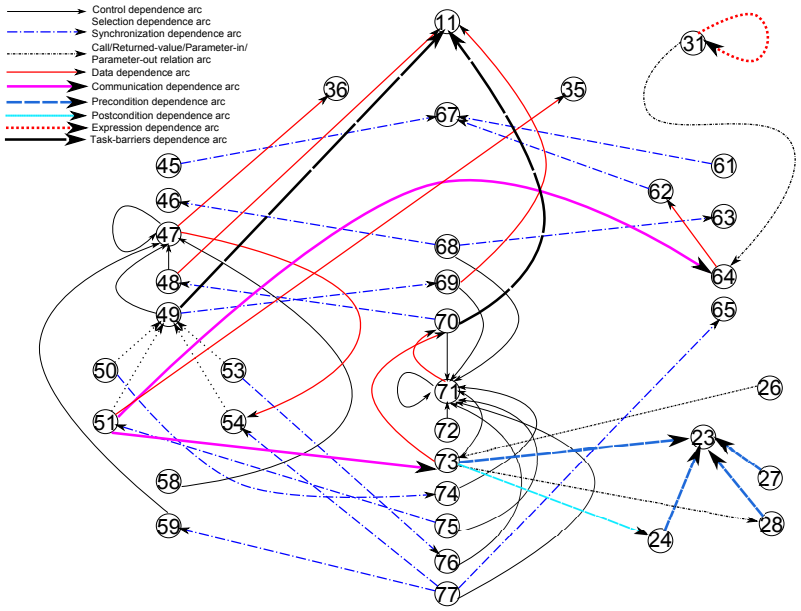

Figure 5.1: An SDN for the Program Example

Algorithm 4 Compute forwared-dominator tree

Input definition-use net
Output forward-dominator tree
if vertex v only has one successor **then**
  the forward-dominator of v is the successor
else
  find all the passes from v to the end vertex
  find the common path of all the passes from v to the end vertex
  the first vertex of the common path is the forward-dominator of v
end if

Chapter 6

A System Dependence Net Generator for Ada 2012 Programs

6.1 Generation Method of SDNs for Ada 2012 Programs

We can capture various program dependences in Figure 6.1, representing a data flow for generating program dependences from the DUN in Ada 2012 programs.

By means of inputing DUNs of target programs, we can get various types of program dependences. We propose the algorithms to compute forwared-dominator tree, control dependence, data dependence, synchronization dependence, communication dependence, selection dependence, precondition dependence, postcondition dependence, and expression dependence for Ada 2012 programs.

6.2 Examples of an SDN Generator for Ada 2012 Programs

Here, we show an text output of the SDN generator corresponding to the input file of the DUN of Example program of Figure 1.

```
1: <line> b-8
2: <line> b-11
3: <line> b-12
4: <line> b-35
5: <line> b-36
6: <line> b-37
```

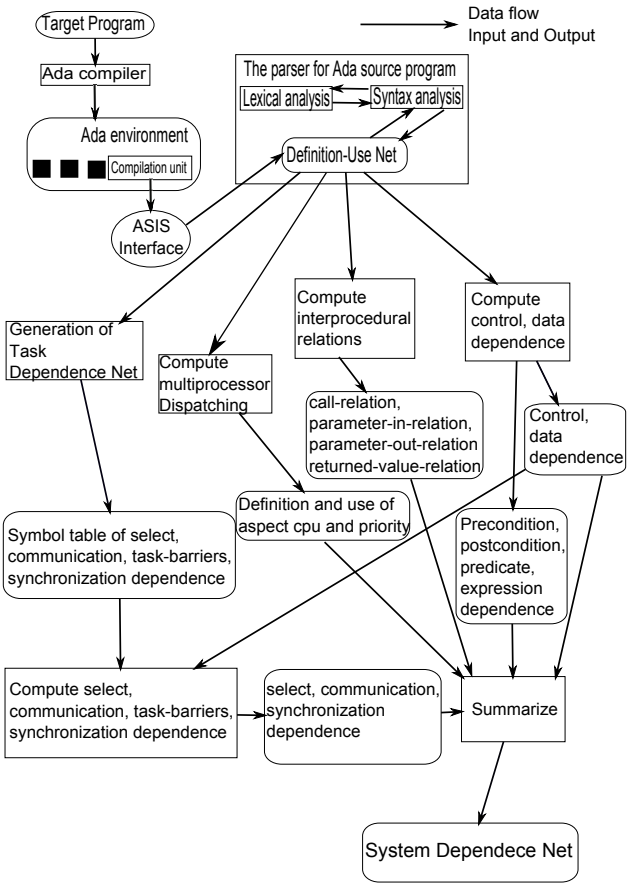

Figure 6.1: A Dada Flow for Generating Program Dependences from DUNs of Ada 2012 Programs

Algorithm 5 Compute control dependence

Input definition-use net
Output control dependence arcs
Make the forward-dominator tree
for each vertex in the DUN **do**
  if the control flow begin from the vertex v is at least two **then**
    make the parent list of the vertex v based on the forward dominator
    for each vertex u in the branch of the vertex **do**
      make the parent list of the vertex u based on the forward dominator
    end for
    if the vertex v is in the parent list of vertex u or the any parent of vertex v is in the parent list of vertex u **then**
      vertex u is control dependent on (vertex u + parent list of vertex v - parent list of vertex v)
    end if
  end if
end for

```
7: <line> b-68
8: <line> b-17  <sync> 22 34
9: <line> b-23  <call> 42
10: <line> b-24
11: <line> b-26  <pre> 9
12: <line> b-27  <pre> 9
13: <line> b-28  <pre> 9
14: <line> b-28  <pre> 9
```

Algorithm 6 Compute data dependence

Input definition-use net
Output data dependence arcs
repeat
  for each vertex v and its successor u **do**
    if the variable x is defined in v **then**
      push v into chain[u][x]
    else
      push chain[v][x] into chain[u][x]
    end if
  end for
until the array chain is changed during the block above
for each vertex w **do**
  for each vertex y used in w **do**
    draw the arc from y to chain[y][w] to show the data dependence
  end for
end for

Algorithm 7 Compute synchronization dependence

Input definition-use net, control dependence arcs
Output synchronization dependence arcs
for each vertice v contains fork/join arc(s) in DUN **do**
  for each vertex u of the successor of the fork/join arcs **do**
    use connectsync (u, v) to draw synchronization dependences
  end for
end for

for each vertex v contains the set of sendings **do**
  for each node u that has the same channel in their receivings **do**
    if the set of sendings of v equals the set of receivings of u **then**
      use connectsync (u, v) to draw synchronization dependences
    end if
  end for
end for

connectsync (u, v)
draw the arc that shows u synchronization depends on v
while u has only one successor **do**
  u' = the successor of u
  if u has the set of sendings or receivings **then**
    break
  end if
  draw the arc that u' depends on v
end while

Algorithm 8 Compute communication dependence

Input definition-use net, control dependence arcs
Output communication dependence arcs
for each vertex v that has set of sendings **do**
  for each vertex u that the set of receivings equals to the set of sendings of v **do**
    for each vertex startnode that v depends on **do**
      for each vetex endnode that depends on u **do**
        draw the arcs that show endnode depends on startnode
      end for
    end for
  end for
end for

```
15: <line> b-29  <pre> 9
16: <line> b-31  <call> 18 35
17: <line> b-32
18: <line> b-33  <param-out> 19
19: <line> b-33
20: <line> b-34
21: <line> b-46  <sync> 7
```

Algorithm 9 Compute selection dependence

Input definition-use net, control dependence arcs
Output selection dependence arcs
for each nondeterministic vertex v **do**
  for each vertex u that control depends on v **do**
    add the arc that shows u selection depends on v
    delete the arc that shows u control depends on v
  end for
end for

Algorithm 10 Compute precondition dependence

Input definition-use net, control dependence arcs
Output precondition dependence arcs
for each vertex v in the DUN **do**
  if v is the immediate forward dominator of a vertex F_{in} **and** t is the immediate forward dominator of v **then**
    if u strong control depends on v **then**
      add the arc that shows u precondition depends on v
      delete the arc that shows u control depends on v
    end if
  end if
end for

Algorithm 11 Compute postcondition dependence

Input definition-use net, control dependence arcs
Output postcondition dependence arcs
for each vertex labeled with $A_{in}(u)$ **do**
  if u is the immediate forward dominator of v **then**
    if t is the immediate forward dominator of v **then**
      add the arc that shows u postcondition depends on v
    end if
  end if
end for

Algorithm 12 Compute expression dependence

Input definition-use net
Output Expression dependence arcs
for each vertex v **do**
  if $F_{in}(v) = U(v)$ **then**
    if $(v, v) \in \mathrm{A}_{C_A}$ **then**
      add the arc that shows v expression depends on v
    end if
  end if
end for

```
22: <line> b-47
23: <line> b-48  <control> 40 <sele> 25
24: <line> b-49  <control> 25 40 <sele> 25
25: <line> b-50  <control> 25 40 <sele> 25
26: <line> b-51  <control> 40 <sele> 25 <sync> 43
27: <line> b-52  <control> 40 <sele> 25 <sync> 43
28: <line> b-54  <control> 40 <sele> 25 <sync> 45
29: <line> b-55  <control> 40 <sele> 25 <sync> 45
30: <line> b-59  <control> 40 <sele> 25 <sync> 43 45
31: <line> b-60
32: <line> b-62  <sync> 7
33: <line> b-63  <sync> 7
34: <line> b-64
35: <line> b-65  <param-out> 19
```

```
36: <line> b-66
37: <line> b-69  <control> 40
38: <line> b-70  <control> 40
39: <line> b-71  <control> 40
40: <line> b-72  <control> 40
41: <line> b-73  <control> 40
42: <line> b-74  <control> 40 <post> 10
43: <line> b-75  <control> 40
44: <line> b-76  <control> 40
45: <line> b-77
46: <line> b-78  <sync> 31 36
47: <line> b-31  <control> 16  <param-in> 48 49
48: <line> b-33  <control> 18
49: <line> b-65  <control> 35
50: <line> b-74  <control> 42
```

6.3 Applications of the SDN

6.3.1 Slicing

The most direct and important use of SDN will be program slicing. The explicit representation of the various program dependences in programs makes SDN ideal for program slicing.

Program slicing was first proposed by Weiser in 1981 [51, 52]. It is a method for automatically decomposing programs. It provides a reduced program by erasing irrelevant statements in the original program based on certain statements and a set of variables, called the "criterion". The reduced program is called a "slice" based on the criterion. He claimed that experienced programmers always create a "slice" in their mind to find the suspect which causes the error. And with program slicing technology, creating the "slice" based on certain criterion can be done automatically and helps programmers by narrowing down the possible scope of the cause of error. During about 30 years's researches, the most popular approach [13] to process program slices are based on SDN. Program slicing based on SDN will be simplified as a reachability problem on a digraph.

6.3.2 Testing

Testing is the process that executes the program with the intention of finding errors. Since SDN can represent the data flow properties of the program, the dependence-coverage of programs can be found by using SDN.

6.3.3 Understanding and Maintenance

To understand a program, we always intent to find which variable in which statement might affect a variable of interest. With a certain slice of the program, the set of the statements and the variables which affect the variable of interest can be easily found. As we mentioned above, to create the slice, we have to build the SDN of the program.

In program maintenance, the problem which is called "ripple effect" [54] is well-known as: whether changing a code in a program will affect the behavior of

other codes of the program, which may cause new error(s). With the slice of the changing code, we can find all the codes in the program that might be affected. It is obviously useful in program maintenance to build slices. As we mentioned above, to build slices, the SDN will come in handy.

6.3.4 Complexity Measurement

Software metrics have many uses in software engineering, such as program understanding, debugging, testing, analysis, maintenance, and so on [14]. Based on SDN, we can define a set of metrics for measuring the complexity of programs from several viewpoints. For example, the metric defined by the sum of all program dependences between statements can be used to measure the complexity of corresponding program. The metric defined by the sum of communication dependences can be used to measure the complexity of concurrency in corresponding program. And the proportion of the communication dependences in a program can be used to measure the degree of concurrency in corresponding program.

Chapter 7

A Tasking Deadlock Detector for Ada 2012 Programs

7.1 Tasking Deadlock Issues

Tasking deadlocks is a serious and complex issue in concurrent Ada programs. A task is said to be blocked in an execution state of a concurrent Ada program, if it is waiting at some synchronization points in its thread of control for synchronization with one or other tasks or even itself. This waiting state will be kept until that either the synchronization has occurred or the task is aborted. A tasking deadlock in a concurrent Ada program is a situation where some tasks form a circle of synchronization waiting relations at some synchronization points that cannot be resolved by the program itself (including the behaviors of other tasks), and hence can never proceed with their computation by themselves [9, 10]. A synchronization waiting relation between tasks is such relation that to synchronize with the other task or tasks, a task is blocked until the synchronization takes place, unless the synchronization waiting has a deadline.

7.2 Previous Researches

Synchronization Waiting Relations in Ada Programs

Ada 95 defines eight types of synchronization waiting relations, i.e., activation waiting relation, finalization waiting relation, completion waiting relation, acceptance waiting relation, entry-calling waiting relation, protection waiting relation, protected-entry calling waiting relation, and suspension waiting relation [16, 22, 38]. In Ada 2005, there is no new synchronization waiting relations [32, 39]. Since Ada 2012, four new operations are extended [7, 8, 40], such as procedure Enqueue and Dequeue defined in Annex A.18.27 [40], procedure wait_for_release defined in Annex D.10.1 [40], corresponding to enqueue waiting relation, dequeue waiting relation, barrier-release waiting relation [32], respectively, and procedure Suspend_Until_True_And_Set_Deadlines defined in Annex D.10 [40], which does not cause new synchronization waiting relation, because of temporary blocking,

i.e., no continuous blocking [32]. So far, there are eleven types of synchronization waiting relation as follows. We also proposed them in [10, 16, 22, 32, 45].

- Activation waiting (Ada 83 or later)

  A task that created some new tasks and initiated their activations in its own body or the body of a block statement executed by it or a subprogram, which may be unprotected or protected, called by it or a protected entry body is blocked until all of these activations complete.

- Finalization waiting (Ada 83 or later)

  Each task depend on one or more masters, and when a master is finalized, it must wait for the termination of any tasks dependent on the master. A block statement or a subprogram executing master is also blocked until the finalization of the master has been performed.

- Completion waiting (Ada 83 or later)

  A task which depend on some master and is blocked at a selective accept statement with an open terminate alternative, must wait for completion together with other dependents of the master considered that are not yet completed.

- Acceptance waiting (Ada 83 or later)

  A task executing an accept statement or a selective accept statement with some open accept alternative but no open delay alternatives and no else part is blocked until a caller of the corresponding entry is selected.

- Entry-calling waiting (Ada 83 or later)

  A task that issued a simple entry call on an entry is blocked until the corresponding rendezvous has finished or the call is canceled by a requeue with abort. Similarly, if a task issued a timed entry call, a conditional entry call or a entry call as the triggering statement of an asynchronous select on an entry and the corresponding rendezvous has started, then it is blocked until the rendezvous has finished or the call is canceled by a requeue with abort.

- Protection waiting (Ada 95 or later)

  A task that issued a call on a protected procedure or a simple entry call on a protected entry of a protected object is blocked until a new protected action can be started on the protected object.

- Protected-entry-calling waiting (Ada 95 or later)

  A task that issued a simple entry call on an protected entry is blocked until the execution of the corresponding entry body has finished. Similarly, a task that issued a timed entry call, a conditional entry call, and the corresponding protected action has started, and the execution of the corresponding entry body has started is blocked until the execution of the corresponding entry body has finished.

- Suspension waiting (Ada 95 or later)

  A task calling the procedure Suspend_Unit_True of a suspension object is blocked until the state of the suspension object becomes true.

- Enqueue waiting (Ada 2012)

  A task calling Enqueue is blocked when a queue in implemented Synchronized_Queue_Interfaces is full until another task calls Dequeue for this queue.

- Dequeue waiting (Ada 2012)

  A task calling Dequeue is blocked when a queue in implemented Synchronized_Queue_Interfaces is empty until another task calls Enqueue for this queue.

- Barrier-release waiting (Ada 2012)

  Tasks calling Wait_For_Release are blocked by other possible tasks calling it until the number of blocked tasks associated with the Synchronous_Barrier object is equal to threshold.

7.3 Queue Operation Related Tasking Deadlocks in Ada 2012 Programs

A tasking deadlock in a concurrent Ada program is situation where some tasks form a circular waiting relation at some synchronization points that cannot be resolved by the program itself (including the behavior of other tasks), and hence these tasks can never proceed with their computation by themselves [9, 10]. Tasking deadlocks is a serious and complex issue in concurrent Ada programs [22].

To detect, avoid and resolve Ada tasking deadlocks, it is indispensable to identify all types of tasking deadlocks. A task synchronization waiting relation between tasks is a relation such that to synchronize with the other task or tasks, a task is blocked until the synchronization takes place, unless the synchronization waiting has a deadline or is breaked by another task.

Cheng proposed a way to completely classify all types of tasking deadlocks by different combinations of various synchronization waiting relations between executing tasking objects [9, 10]. According to this way of classification, various combinations of synchronization waiting relations concerning synchronization waiting tasks may lead to various types of tasking deadlocks [9, 10, 16, 22, 32, 45]. And there is a method and a tool to detect tasking deadlocks at run-time in [22, 45].

Ada 2012 defined four new operations those can block a task, i.e., procedure Enqueue and procedure Dequeue in the package Synchronized_Queue_Interfaces in Queue container, procedure Wait_For_Release in the package Synchronous_Barriers and procedure Suspend_Until_True_And_Set_Deadlines in the package Ada.Synchro nous_Task_Con
trol.EDF [7, 8, 40]. Moreover, [32] showed some types of tasking deadlocks concerning these operations. On the other hand, since the days of Ada 95 [38], a requeue statement has come available, which can be used to complete an accept_statement or entry_body, while redirecting the corresponding entry call to a new (or the same) entry queue [38, 39, 40]. The requeue statement are permitted using procedure renamed an entry, as same as Enqueue and Dequeue in Ada 2012 [40]. Therefore, though the requeue is different from Enqueue and Dequeue in the queue container, they are implemented essentially as same as entry [40]. They are together called queue operations. There may be some queue operation related tasking deadlocks.

7.3.1 Queue Operations

The Enqueue and Dequeue in generic package Containers.Synchronized_Queue_Inte rfaces defined in Annex A.18.27 [40]. From it, four different queue containers are derived, such as A.C.Unbounded_Synchronized_Queues in A.18.28, A.C.Bounded_S ynchronized_Queues in A.18.29, A.C.Unbounded_Priority_Queues in A.18.30, A.C. Bounded_Pri
ority_Queues in A.18.31 [7, 8, 40].

A.18.28 representing task-safe queue is unbounded, whose Enqueue can never block, because the capacity of the queue is unbounded; A.18.29 is bounded, and thus Enqueue might cause blocking, if the capacity of the queue is full; Similarly, A.18.30 provides task-safe queue, and Enqueue can never block, while Enqueue in A.18.31 might cause blocking, because of bounded capacity of the queue. If the capacity of the queue is empty, the Dequeue of these four queue containers might cause blocking, because of a dequeue waiting relation. Note that A.18.30 and A.18.31 provide procedure Dequeue_Only_High_Priority, which is not blocking, returns immediately. Unlike Dequeue, it is not synchronization waiting relation [40]. Thus, in this article, we do not discuss the Dequeue_Only_High_Priority of queue operations in Ada 2012.

On the other hand, though Enqueue and Dequeue are defined in Ada 2012, there has been introduced a requeue statement and a requeue with abort statement since Ada 95 [38]. A requeue_statement presents the completion of an accept statement or entry body, while redirecting the corresponding entry call to a new (or the same) entry queue [38, 39, 40]. The statement's form is

```
requeue entry_name_or_procedure_name [with abort];
```

The entry body or accept statement, that has a requeue, is exited, as long as the entry call has been redirected to another entry queue, or the same entry queue. Requeue must is within the body of an accept statement, or within entry body, which is a protected entry. With abort is option, which allows to cancel the call, due to an abort or the expiration of a delay [38, 39, 40], if any.

Ada 83 allows renaming an entry as a procedure [37]. Like this,

```
procedure P(Para: in Item) renames T.E;
```

Ada 2012 provides a new feature [40] that we use a requeue statement using an entry renamed as a procedure as following:

```
requeue P;
```

As mentioned above, the requeue is just redirecting an entry call to a new (or the same) entry queue. If there are not any entry calls in this entry queue, the entry call requeued can reach the accept statement corresponding to it. The sequence of statements, if any, of the accept statement is executed by the called task, while the calling task remains suspended, called rendezvous [37, 38, 39, 40]. But the accept statement or entry body that has the requeue statement is completed.

However, if there are another entry calls in this queue, the entry call requeued is queued to waiting until the accept statement corresponding to this entry removes other call in front of it from the queue, i.e., the call requeued must wait until the call in front of it rendezvous. The calls are processed in the order of arrival [37, 38, 39, 40]. And if the requeue statement redirects an entry call within a protected body, then the calling task must wait until protected entry call is finalized. Therefore, a requeue statement cannot cause a synchronization waiting relation, such that blocking a task directly. But it can indirectly cause tasking deadlock.

We consider that the Requeue is also a queue operation, though it is different from Enqueue and Dequeue in Queue Container, as above. Because procedure Enqueue and Dequeue in A.18.27 (The Generic Package Containers. Synchronized_Queue_Interfaces) are implemented by entry in essence [40], as same as Requeue. Three operations are called queue operations. This article is focused on queue operation related tasking deadlocks in Ada 2012 programs.

7.3.2 Examples of Queue Operation Related Tasking Deadlocks in Ada 2012 Programs

Seven example programs are presented in this section. We present each program by its Ada 2012 code and Task-Wait-For Graph corresponding to it. A program has a temporary circle with requeue statements of queue operations. Though it does not lead to a tasking deadlock, but an infinite requeue circle. All of others have queue operation related tasking deadlocks.

Task-Wait-For Graph

A Task-Wait-For Graph (TWFG) at time t (this time may be a physical time in an interleaved implementation of Ada or a virtual time in a distribed implementation of Ada) is a kind of arc-classified digraph to represent tasking waiting state in an execution of an Ada program [10]. The TWFG explicitly represents various types of waiting relations in an execution of an Ada program. The notion of TWFG was originally proposed for classification and detection of tasking deadlocks in Ada 83 programs [9, 10] and was extended to deal with tasking deadlocks in Ada 95 programs [16]. In a TWFG, vertices present tasking objects. An executing tasking object in an execution state of a concurrent Ada 2012 program is any of the following: a task whose activation has been initiated and whose state is not terminated, a block statement that is being executed by a task, a subprogram that is being called by a task, a protected subprogram that is being called by a task, a protected object on which a protected action is undergoing, and a suspension object that is being waited by a task. Arcs indicate synchronization waiting relations which are binary relations between tasking objects [9, 10]. In a TWFG of an Ada 2012 program, there are 11 types of arcs: activation waiting arc, finalization waiting arc, completion waiting arc, acceptance waiting arc, entry-calling waiting arc, protection waiting arc, protected-entry-calling waiting arc, suspension waiting arc, enqueue waiting arc, dequeue waiting arc, and barrier-release waiting arc, respectively, corresponding to an activation waiting relation, finalization waiting relation, completion waiting relation, acceptance waiting relation, entry-calling waiting relation, protection waiting relation, protected-entry-calling waiting relation, suspension waiting relation, enqueue waiting relation, dequeue waiting relation, and barrier-release waiting relation [9, 10, 16, 22, 32, 45].

Various types of arcs are shown by $\rightarrow_{Act}$, $\rightarrow_{Fin}$, $\rightarrow_{Com}$, $\rightarrow_{Acc}$, $\rightarrow_{EC}$, $\rightarrow_{Pro}$, $\rightarrow_{PEC}$, $\rightarrow_{Sus}$, $\rightarrow_{Enq}$, $\rightarrow_{Deq}$, and $\rightarrow_{BR}$, denoting activation waiting arc, finalization waiting arc, completion waiting arc, acceptance waiting arc, entry-calling waiting arc, protection waiting arc, protected-entry-calling waiting arc, suspension waiting arc, Enqueue waiting arc, Dequeue waiting arc, and Barrier-release waiting arc, respectively [9, 10, 16, 22, 32, 45].

Though the TWFGs corresponding to enqueue and dequeue waiting relations have been proposed in [32], there is no representation of the requeue of queue operations. It needs to present it that might cause a tasking deadlock, even though it is not synchronization waiting relation. The arc $\rightarrow_{Req}$ represents the entry call requeued is waiting to rendezvous an accept statement corresponding to it in another task. Especially, if the requeue statement is within a protected entry body, the TWFG requires additional representation of tasking objects, i.e., entry body.

Example Program 1: Infinite Requeue Circle without Tasking Deadlocks

This program has no synchronization waiting relations. Between two entry bodies in the protected body, two requeue statements form a circle representing an infinite

requeue action that cannot be resolved by the program itself. Nevertheless, there are no tasking deadlocks, because the circle in Figure 7.1 is not continuous but temporary, that is, it allows to have just one entry calling at time t, i.e.,

$$E_1 \rightarrow_{R_{eq}} E_2$$

$$E_2 \rightarrow_{R_{eq}} E_1$$

Example Program 1

```
with Ada.Text_IO;
use Ada.Text_IO;
procedure Infinite_Requeue_Circle is
  task T1;
  task T2;

  protected E2E is
    entry E1;
    entry E2;
  end E2E;

  protected body E2E is
    entry E1 when True is
    begin
      requeue E2;
    end E1;

    entry E2 when True is
    begin
      delay 1.0;
      put_line("!");
      requeue E1;
    end E2;
  end E2E;

  task body T1 is
  begin
    E2E.E1;
  end T1;

  task body T2 is
  begin
    E2E.E2;
  end T2;

begin
  null;
end Infinite_Requeuee_Circle;
```

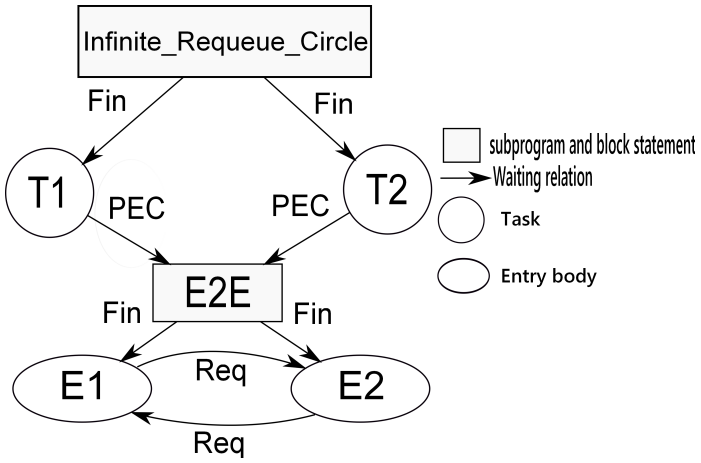

Figure 7.1: TWFG of Example Program 1 for an Infinite Requeue Circle without Tasking Deadlocks

Example Program 2: Tasking Deadlocks with Requeue Operations

Two requeue statements cause two synchronization waiting relations indirectly, i.e., there is a continuous circle that cannot be resolved by the program itself. The program might have a queue operation related tasking deadlock, which is circular entry calling during the activation of a task in Figure 7.2:

$$T_1 \rightarrow_{R_{eq}} T_2 \rightarrow_{R_{eq}} T_1$$

Example Program 2

```
with Ada.Containers.Synchronized_Queue_Interfaces;
with Ada.Containers.Bounded_Synchronized_Queues;

procedure Req_Req is
  type Queue_Element is new String(1..0);
  package String_Queues is new Ada.Containers.Synchronized_Queue_Interfaces
          (Element_Type => Queue_Element);
  package String_Priority_Queues is new Ada.Containers.Bounded_Synchronized_Queues
          (Queue_Interfaces => String_Queues,
           Default_Capacity => 1);

  Q1 : String_Priority_Queues.Queue;
  Elem : Queue_Element;

  task T1 is
    entry E1;
  end T1;

  task T2 is
    entry E2;
  end T2;

  protected E2E is
    entry E1;
    entry E2;
  end E2E;

  protected body E2E is
    entry E1 when True is
    begin
      requeue T2.E2;
```

```
    end E1;

    entry E2 when True is
    begin
      requeue T1.E1;
    end E2;
  end E2E;

  procedure P1 renames E2E.E1;
  procedure P2 renames E2E.E2;

  function GET return Integer is
  begin
    P2;
    return 0;
  end GET;

  task body T1 is
  begin
    P1;
    accept E1;
  end T1;

  task body T2 is
    I :Integer :=GET;
  begin
    accept E2;
  end T2;

begin
  null;
end Req_Req;
```

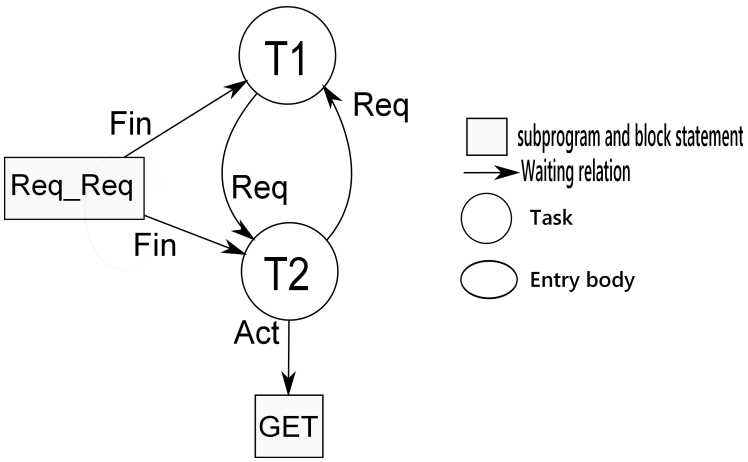

Figure 7.2: TWFG of Example Program 2

Example Program 3: A Tasking Deadlock with Requeue and Enqueue Operations

In this program requeue and enqueue operations form a circle between two entry bodies, which indirectly result in a tasking deadlock between tasks, when P2 is requeued to entry-calling wait for the finalization of entry body E1 in protected body E2E, while E1 is blocking to wait to enqueue an element to the queue that

is full.The synchronization waiting circle in Figure 7.3 is represented as follows:

$$E_2 \rightarrow_{R_{eq}} E_1 \rightarrow_{E_{nq}} E_2$$

Example Program 3

```
with Ada.Containers.Synchronized_Queue_Interfaces;
with Ada.Containers.Bounded_Synchronized_Queues;

procedure Req_Enq is
  type Queue_Element is new String(1..0);
  package String_Queues is new Ada.Containers.Synchronized_Queue_Interfaces
          (Element_Type => Queue_Element);
  package String_Priority_Queues is new Ada.Containers.Bounded_Synchronized_Queues
          (Queue_Interfaces => String_Queues,
           Default_Capacity => 1);

  Q : String_Priority_Queues.Queue;
  Elem : Queue_Element;

  protected E2E is
    entry E1;
    entry E2;
  end E2E;

  protected body E2E is
    entry E1 when True is
    begin
      Q.Enqueue(Elem);
    end E1;

    entry E2 when True is
    begin
      Q.Enqueue(Elem);
      requeue E2E.E1;
    end E2;
  end E2E;

  task T1;
  task T2;

  procedure P1 renames E2E.E1;
  procedure P2 renames E2E.E2;

  task body T1 is
  begin
    delay 0.1;
    P1;
  end T1;

  task body T2 is
  begin
    P2;
  end T2;

begin
  null;
end Req_Enq;
```

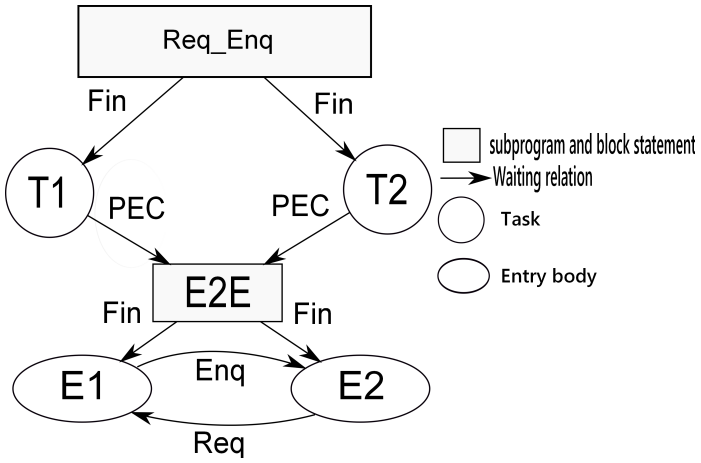

Figure 7.3: TWFG of Example Program 3

Example Program 4: A Tasking Deadlock with Requeue and Dequeue Operations

Program Req_Deq forms a circle between two entry bodies by requeue and dequeue operations, which indirectly result in a tasking deadlock between tasks, when P2 is requeued to entry-calling wait for the finalization of entry body E1 in protected body E2E, the entry body E2 is completed by requeue statement. However, the Enqueue below this requeue statement is unreachable code, namely the queue is still empty, resulting in that E1 is blocking to wait to dequeue an element. The synchronization waiting circle in Figure 7.4 is shown as follows:

$$E_2 \rightarrow_{R_{eq}} E_1 \rightarrow_{D_{eq}} E_2$$

Example Program 4

```
with Ada.Containers.Synchronized_Queue_Interfaces;
with Ada.Containers.Bounded_Synchronized_Queues;

procedure Req_Deq is
  type Queue_Element is new String(1..0);
  package String_Queues is new Ada.Containers.Synchronized_Queue_Interfaces
          (Element_Type => Queue_Element);
  package String_Priority_Queues is new Ada.Containers.Bounded_Synchronized_Queues
          (Queue_Interfaces => String_Queues,
           Default_Capacity => 1);

  Q : String_Priority_Queues.Queue;
  Elem : Queue_Element;

  protected E2E is
    entry E1;
    entry E2;
  end E2E;

  protected body E2E is
    entry E1 when True is
    begin
      Q.Dequeue(Elem);
    end E1;

    entry E2 when True is
    begin
```

```
      requeue E2E.E1;
      Q.Enqueue(Elem);          --never reach,such that the queue is empty
    end E2;
  end E2E;

  task T1;
  task T2;

  procedure P1 renames E2E.E1;
  procedure P2 renames E2E.E2;

  task body T1 is
  begin
    delay 0.1;
    P1;
  end T1;

  task body T2 is
  begin
    P2;
  end T2;

begin
  null;
end Req_Deq;
```

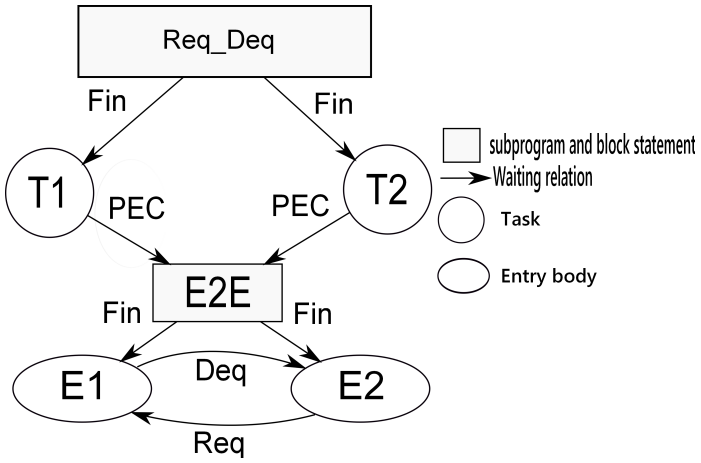

Figure 7.4: TWFG of Example Program 4

Example Program 5: Tasking Deadlocks with Enqueue Operations

This program has enqueue waiting relations of the same queue. The two enqueue waiting relations form a circle at time t that cannot be resolved by the program itself, when the Enqueue of Q of T1 is enqueue waiting, while the Dequeue of Q of T2 is enqueue waiting, causing an enqueue operaiton related tasking deadlock. The synchronization waiting circle in Figure 7.5 is followed by:

$$T_1 \rightarrow_{E_{nq}} T_2 \rightarrow_{E_{nq}} T_1$$

Example Program 5

```
with Ada.Containers.Synchronized_Queue_Interfaces;
with Ada.Containers.Bounded_Synchronized_Queues;
```

```
procedure Enq_Enq is
  type Queue_Element is new String(1..0);
  package String_Queues is new Ada.Containers.Synchronized_Queue_Interfaces
          (Element_Type => Queue_Element);
  package String_Priority_Queues is new Ada.Containers.Bounded_Synchronized_Queues
          (Queue_Interfaces => String_Queues,
           Default_Capacity => 1);

  Q: String_Priority_Queues.Queue;
  Elem : Queue_Element;

  task T1;
  task T2;

  task body T1 is
  begin
    loop
      Q.Dequeue(Elem);
      Q.Enqueue(Elem);
      Q.Dequeue(Elem);
    end loop;
  end T1;

  task body T2 is
  begin
    loop
      Q.Enqueue(Elem);
      Q.Enqueue(Elem);
      Q.Dequeue(Elem);
    end loop;
  end T2;

begin
  null;
end Enq_Enq;
```

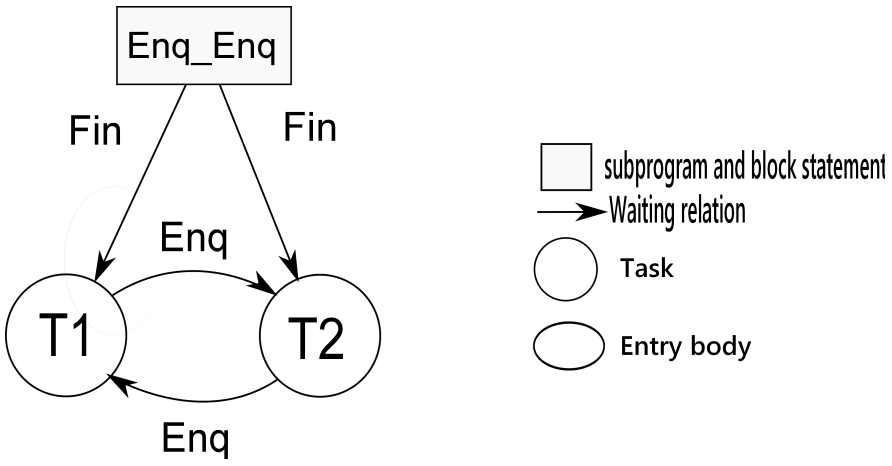

Figure 7.5: TWFG of Example Program 5

Example Program 6: Tasking Deadlocks with Dequeue Operations

This program has dequeue waiting relations of different queues. The two dequeue waiting relations form a circle that cannot be resolved by the program itself, when the Dequeue of Q1 of T1 is dequeue waiting, while the Dequeue of Q2 of T2 is dequeue waiting, causing a dequeue operaiton related tasking deadlock. The

synchronization waiting circle in Figure 7.6 is given as follows:

$$T_2 \rightarrow_{D_{eq}} T_1 \rightarrow_{D_{eq}} T_2$$

Example Program 6

```
with Ada.Containers.Synchronized_Queue_Interfaces;
with Ada.Containers.Bounded_Synchronized_Queues;

procedure Deq_Deq is
  type Queue_Element is new String(1..0);
  package String_Queues is new Ada.Containers.Synchronized_Queue_Interfaces
          (Element_Type => Queue_Element);
  package String_Priority_Queues is new Ada.Containers.Bounded_Synchronized_Queues
          (Queue_Interfaces => String_Queues,
           Default_Capacity => 1);

  Q1, Q2 : String_Priority_Queues.Queue;
  Elem : Queue_Element;

  task T1;
  task T2;

  task body T1 is
  begin
    loop
      Q2.Enqueue("");
      Q1.Dequeue(Elem);
    end loop;
  end T1;

  task body T2 is
  begin
    loop
      Q2.Dequeue(Elem);
      Q1.Enqueue("");
      Q2.Dequeue(Elem);
    end loop;
  end T2;

begin
  null;
end Deq_Deq;
```

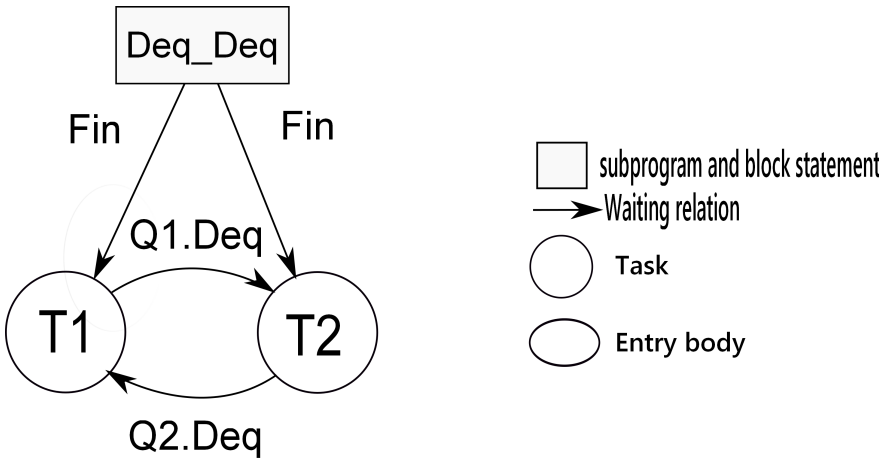

Figure 7.6: TWFG of Example Program 6

Example Program 7: Tasking Deadlocks with Dequeue and Enqueue Operations

This program has dequeue waiting relations and enqueue waiting relations of different queues. The two dequeue waiting relations form a circle that cannot be resolved by the program itself, when the enqueue of Q1 of T2 is enqueue waiting, while the Dequeue of Q2 of T1 is dequeue waiting, causing a dequeue and enqueue operaitons related tasking deadlock. The synchronization waiting circle in Figure 7.7 is expressed as follows:

$$T_2 \rightarrow_{D_{eq}} T_1 \rightarrow_{E_{nq}} T_2$$

Example Program 7

```
with Ada.Containers.Synchronized_Queue_Interfaces;
with Ada.Containers.Bounded_Synchronized_Queues;
procedure Deq_Enq is
  type Queue_Element is new String(1..0);
  package String_Queues is new Ada.Containers.Synchronized_Queue_Interfaces
          (Element_Type => Queue_Element);
  package String_Priority_Queues is new Ada.Containers.Bounded_Synchronized_Queues
          (Queue_Interfaces => String_Queues,
           Default_Capacity => 1);
  Q1, Q2 : String_Priority_Queues.Queue;
  Elem : Queue_Element;
  task T1;
  task T2;

  task body T1 is
  begin
    loop
      Q2.Dequeue(Elem);
      Q1.Dequeue(Elem);
    end loop;
  end T1;
  task body T2 is
  begin
    loop
      Q1.Enqueue("");
      Q1.Enqueue(Elem);
      Q2.Enqueue(Elem);
    end loop;
  end T2;
begin
  null;
end Deq_Enq;
```

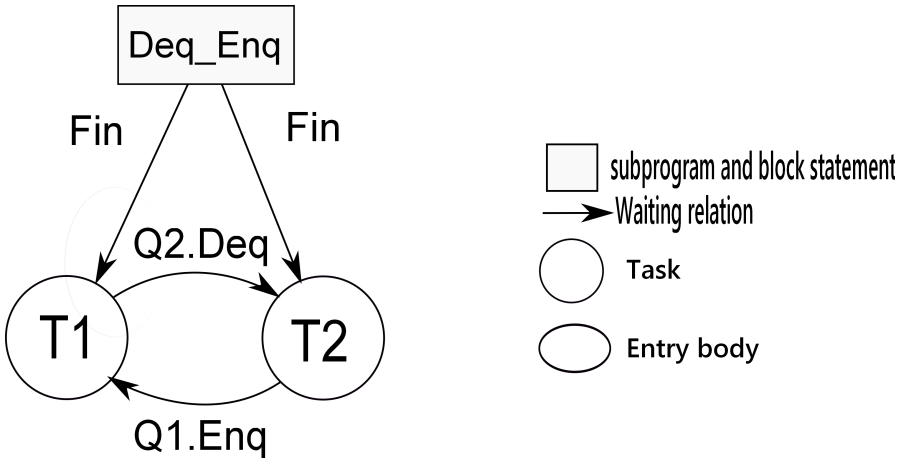

Figure 7.7: TWFG of Example Program 7

Example Program 8: Tasking Deadlocks with Queue Operations

This program has requeue operation, dequeue and enqueue waiting relations of different queues. The three queue operations form a circle that cannot be resolved by the program itself, when the Enqueue of Q1 of T2 is enqueue waiting, and the Dequeue of Q2 of T3 is dequeue waiting, meanwhile the requeue operation of entry call E2 of T2, during the activation of the task T1, is waiting to rendezvous the finalization of the body of the accept statement of T2, causing queue operaiton related tasking deadlocks together. The synchronization waiting circles in Figure 7.8 are described as follows:

$$T_1 \rightarrow_{PEC} E2E \rightarrow_{F_{in}} E_1 \rightarrow_{R_{eq}} T_2 \rightarrow_{E_{nq}} T_3 \rightarrow_{D_{eq}} T_1$$

$$T_2 \rightarrow_{E_{nq}} T_3 \rightarrow_{D_{eq}} T_1 \rightarrow_{PEC} E2E \rightarrow_{F_{in}} E_1 \rightarrow_{R_{eq}} T_2$$

$$T_3 \rightarrow_{D_{eq}} T_1 \rightarrow_{PEC} E2E \rightarrow_{F_{in}} E_1 \rightarrow_{R_{eq}} T_2 \rightarrow_{E_{nq}} T_3$$

Example Program 8

```
with Ada.Containers.Synchronized_Queue_Interfaces;
with Ada.Containers.Bounded_Synchronized_Queues;

procedure Queue_Deadlock is
  type Queue_Element is new String(1..0);
  package String_Queues is new Ada.Containers.Synchronized_Queue_Interfaces
          (Element_Type => Queue_Element);
  package String_Priority_Queues is new Ada.Containers.Bounded_Synchronized_Queues
          (Queue_Interfaces => String_Queues,
           Default_Capacity => 1);

  Q1, Q2 : String_Priority_Queues.Queue;
  Elem : Queue_Element;

  task T1 is
    entry E1;
  end T1;

  task T2 is
    entry E2;
  end T2;
```

```
  task T3;

  protected E2E is
    entry E1;
  end E2E;

  protected body E2E is
    entry E1 when True is
    begin
      requeue T2.E2;
    end E1;
  end E2E;

  procedure P1 renames E2E.E1;

  function GET return Integer is
  begin
    P1;
    return 0;
  end GET;

  task body T1 is
    I :Integer :=GET;
  begin
    accept E1;
    Q2.Enqueue(Elem);
  end T1;

  task body T2 is
  begin
    accept E2 do
      Q1.Enqueue(Elem);
      Q1.Enqueue(Elem);
    end E2;
    T1.E1;
  end T2;

  task body T3 is
  begin
    Q2.Dequeue(Elem);
    Q1.Dequeue(Elem);
    T2.E2;
  end T3;

begin
  null;
end Queue_Deadlock;
```

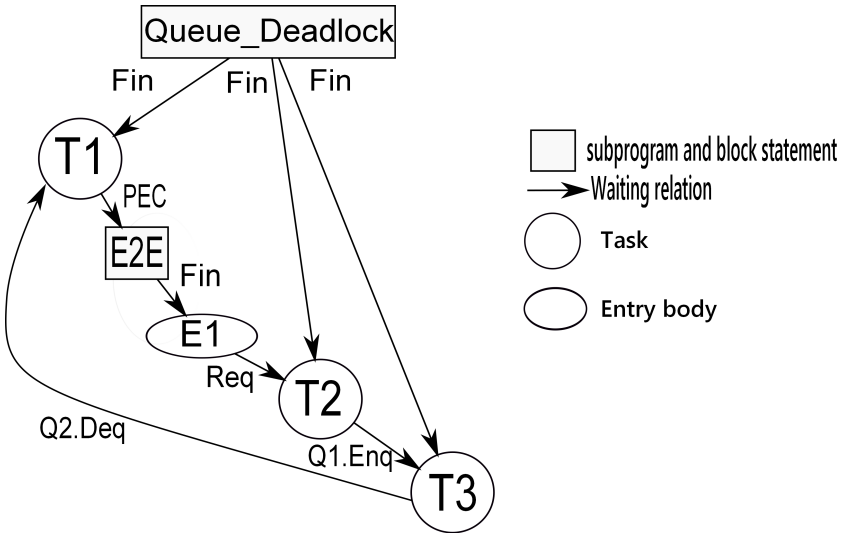

Figure 7.8: TWFG of Example Program 8

Example Program 9: Tasking Deadlocks with All Synchronization Waiting Relations

This program has 11 types of synchronization waiting relations. The synchronization waiting relations form five cycles that cannot be resolved by the program itself. Therefore, five tasking deadlocks must occur in this program. The five cycles of synchronization waiting relations in this program are below and its TWFG is Figure 7.9:

$$T_1 \rightarrow_{A_{cc}} T_4 \rightarrow_{PEC} V \rightarrow_{GET1} \rightarrow_{EC} T_2 \rightarrow_{Com} T_1$$

$$T_1 \rightarrow_{A_{cc}} T_5 \rightarrow_{F_{in}} B \rightarrow_{F_{in}} T_7 \rightarrow_{S_{us}} T_2 \rightarrow_{C_{om}} T_1$$

$$T_1 \rightarrow_{A_{cc}} T_5 \rightarrow_{F_{in}} B \rightarrow_{F_{in}} T_8 \rightarrow_{D_{eq}} T_6 \rightarrow_{F_{in}} T_9 \rightarrow_{F_{in}} GET2 \rightarrow_{BR} T_2 \rightarrow_{Com} T_1$$

$$T_1 \rightarrow_{Acc} T_5 \rightarrow_{Fin} B \rightarrow_{Fin} W \rightarrow_{Pro} V \rightarrow_{Fin} GET1 \rightarrow_{EC} T_2 \rightarrow_{Com} T_1$$

$$T_4 \rightarrow_{PEC} V \rightarrow_{Fin} GET1 \rightarrow_{EC} T_2 \rightarrow_{Com} T_3 \rightarrow_{Enq} T_4$$

Example Program 9

```
pragma Task_Dispatching_Policy
  (Fifo_Within_Priorities);
with System;
with Ada.Containers.Synchronized_Queue_Interfaces;
with Ada.Containers.Bounded_Synchronized_Queues;
with Ada.Synchronous_Task_Control;
with Ada.Synchronous_Barriers;
with Ada.Text_IO;
use Ada.Synchronous_Task_Control;
use Ada.Synchronous_Barriers;
use Ada.Containers;
use Ada.Text_IO;
procedure dl4 is
   type Queue_Element is new String(1..0);
   package String_Queues
    is new Synchronized_Queue_Interfaces
    (Element_Type => Queue_Element);
   package String_Priority_Queues
    is new Bounded_Synchronized_Queues
```

```
 (Queue_Interfaces
    => String_Queues, Default_Capacity =>1);
Q1, Q2: String_Priority_Queues.Queue;
El : Queue_Element;
Number_Of_Tasks : constant := 2;
BO : Ada.Synchronous_Barrier.
 Synchronous_Barrier(Number_Of_Tasks);
Notif           : Boolean := False;
type ITEM is new Integer;
task T1 is entry E1; end T1;
task T2 is entry E2; end T2;
task T3; task T4;
S : Suspension_Object;
function GET1 return ITEM is
begin
   T2.E2;
   return 0;
end GET1;
function GET2 return ITEM is
begin
   Wait_For_Release(BO, Notif);
   return 0;
end GET2;
protected V is
   procedure W (X : in ITEM);
   entry R (X : out ITEM);
private
   Var : ITEM := 0;
end V;
protected body V is
   procedure W (X : in ITEM) is
   begin
      Var := X;
   end W;
   entry R (X : out ITEM) when True is
   begin
      X := GET1;
   end R;
end V;
task body T1 is
   task T5;
   task T6;
   task body T5 is
   begin
      B : declare
         task T7;
         task T8;
         task body T7 is
         begin
            Suspend_Until_True(S);
         end T7;
         task body T8 is
         begin
            Q2.Dequeue(El);
         end T8;
         Y : ITEM;
      begin
         V.W (Y);
      end B;
      T1.E1;
   end T5;
   task body T6 is
      task T9;
      task body T9 is
         I : ITEM := GET2;
      begin
         null;
      end T9;
   begin
```

```
            Q2.Enqueue("");
        end T6;
    begin
        accept E1;
    end T1;
    task body T2 is
    begin
        select when False =>
                accept E2;
        or
            terminate;
        end select;
        Set_True (S);
        Wait_For_Release(BO, Notif);
    end T2;
    task body T3 is
    begin
        Q1.Enqueue("");
        Q1.Enqueue("");
    end T3;
    task body T4 is
        Z : ITEM;
    begin
        V.R (Z);
        Q1.Dequeue(E1);
        T1.E1;
    end T4;
begin
    null;
end dl4;
```

7.4 Principle of Detecting Tasking Deadlocks

As previous sections, we have prensented some synchronization waiting relations. A task that issued a simple entry call on an protected entry is blocked until the execution of the corresponding entry body has finished. Similarly, a task that issued a timed entry call, a conditional entry call, and the corresponding protected action has started, and the execution of the corresponding entry body has started is blocked until the execution of the corresponding entry body has finished.

Note that in the above synchronization waiting relations we do not consider those selective accept statements with open delay alternatives or else part and those statements of asynchronous selects which have not yet been accepted because a task reaching any such selective accept or entry call can change its own waiting state by itself. As a result, all of above synchronization waiting relations have a common property. Therefore, a circular synchronization waiting relation formed among some tasks implies that a tasking deadlock might have occurred there.

We described the TWFG in previous section. Having TWFGs as a formal representation for the waiting state of task synchronization in an execution of an Ada program, a run-time detector for tasking deadlocks can work by monitoring the tasking behavior of the program, managing a TWFG for the program, detecting cycles in the TWFG, and reporting detected tasking deadlocks.

There are alternative approaches to monitor the tasking behavior of Ada programs. One is the source program transformation approach. In the approach, a target concurrent Ada program P is transformed by a preprocessor into another

Ada program P', called subject program of P, such that P' preserves the tasking behavior of P and during its execution P' will communicate with a run-time monitor when each tasking event of P occurs in P' and pass information about the tasking event to the run-time monitor. The other is the run-time environment support approach. In the Approach, a run-time monitor is implemented as a component of the underlying run-time support environment of Ada language and information concerning tasking events in a target concurrent Ada program is provided directly by the run-time monitor.

We implemented the Ada 2012 tasking deadlock detector in the source program transformation approach to monitor tasking behavior of Ada 2012 programs. Our tool is in two separate components, i.e., a preprocessor and a run-time monitor. The preprocessor transforms a target Ada program P into its subject program P' such that some Ada codes are inserted at each point where a tasking event occurs for passing information about the tasking event to the runtime monitor. To monitor tasking behavior of P, P' is compiled, linked with the separately compiled run-time monitor, and then executed. During its execution, P' communicates with the run-time monitor when a tasking event of P occurs in P' and pass information about the tasking event to the run-time monitor. The run-time monitor records the collected information, manages the TWFG of the target program, detects cycles in the TWFG when it is updated, and reports detected tasking deadlocks, if any.

In monitoring a concurrent program, since its behavior is generally run-time dependent, there is an "uncertainty principle," i.e., any monitoring mechanism will interfere with the performance of the program and even may alter its behavior. Therefore, an important issue in implementing an execution monitor for concurrent programs is how to devise mechanisms to reduce the interference and alteration imposed on the behavior of target programs by monitoring actions. The authors have proposed a notion, named "partial order transparency," for monitoring concurrent systems. It is a minimum requirement for an execution monitor in order to reduce its interference and alternation to the behavior of a target system being monitored. An execution monitor satisfying the requirement is transparent to the partial order with respect to occurrences of concurrent events during an execution of the target system. Our monitor of Ada 2012 tasking deadlock detector is partial order transparent.

7.5 Run-Time Detection of Tasking Deadlocks in Ada 2012 Programs

We have used the ASIS to develop a run-time detector for tasking deadlocks in Ada 95 programs. The ASIS is short for the Ada Semantic Interface Specification, which is an interface between an Ada environment and any tool requiring information from it, is supposed to be effective in implementing the preprocessor. An Ada compiler is regarded as a part of the system of our tasking deadlock detector and an ASIS-based tool makes use of an Ada environment which is managed by the compiler. As a future work, we should translate the current preprocessor into an ASIS-based tool in terms of the cooperation between the compiler and the tool.

We have added some codes for transforming a target Ada 2012 program P into another Ada 2012 program P'. Transformed another Ada 2012 program P' is inserted some procedure calls and function calls to the tasking information collector at each tasking information extraction point in target programs to observe tasking events.

There is the execution result of Figure 7.9 corresponding to the program above, such that they represent that some tasking deadlocks will occur at some run-time. Namely, all 11 types of tasking deadlocks in Ada 2012 programs can be run-time detected. We just show a snippet of results.

```
Tasking deadlock will occur in this program!
* At    0.005565337 sec after execution start,
  there is such a situation as follows :
* task Main.T2 is waiting for terminating together with
 task Main.T3
* TASKS Main.T3 is calling entry R of protected object Main.V
* PROTECTED_OBJECTS Main.V is calling subprogram Main.GET
* FUNCTIONS Main.GET is calling entry E2 of task Main.T2
* Status of all tasks are follows:
* task name              => MAIN_TASK
* state of the task      => WORKING_FOR_INTERNAL_AFFAIRS
* task name              => Main.T1
* parent of the task     => Main
* state of the task      => ACCEPTING
* communication entry    => Main.T1.E1
* task name              => Main.T1.T4
* parent of the task     => Main.T1
* state of the task      => WORKING_FOR_INTERNAL_AFFAIRS
* task name              => Main.T1.T5
* parent of the task     => Main.T1
* state of the task      => ACTIVATING
* task name              => Main.T1.T5.T8
* parent of the task     => Main.T1.T5
* state of the task      => S IMPLE_ENTRY_CALLING
* communication entry    => Main.T2.E2
* task name              => Main.T2
* parent of the task     => Main
* state of the task      => TERMINATION_SELECTING
* task name              => Main.T3
* parent of the task     => Main
* state of the task      => SIMPLE_ENTRY_CALLING
```

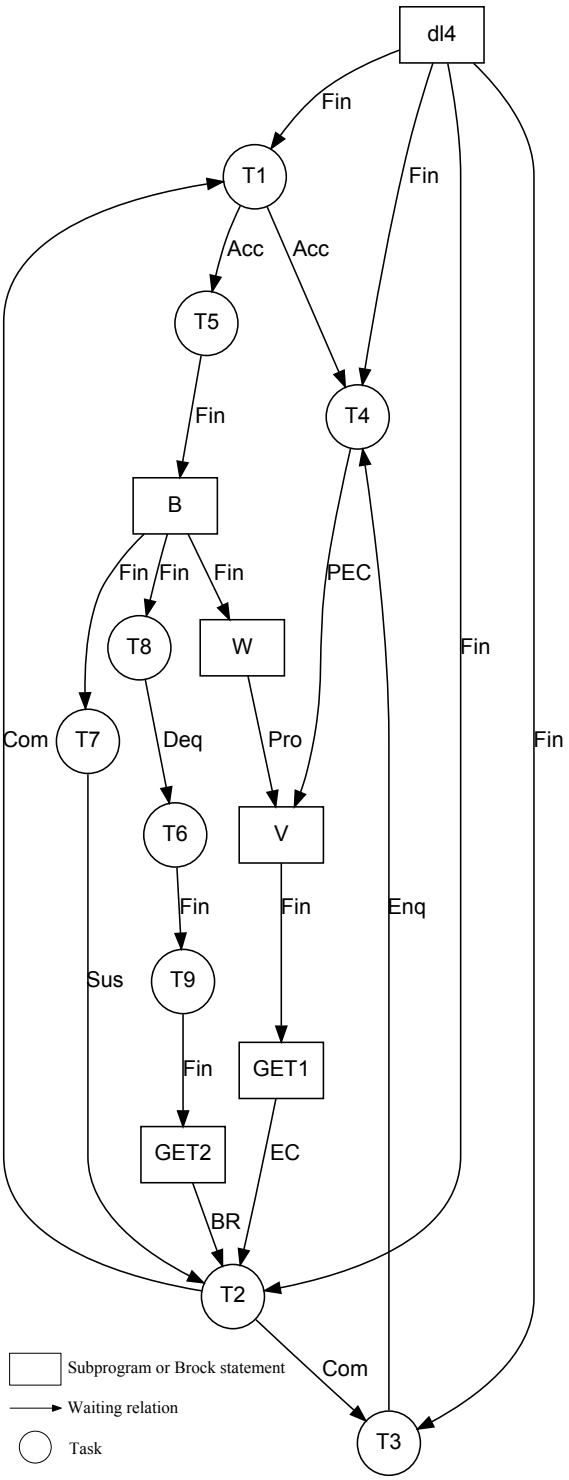

Figure 7.9: TWFG of Example Program 9

Chapter 8

Contract-Based Programming for Future Computing with Ada 2012

8.1 An Overview

For future advanced software engineering, speed is not everything, but it requires cleverer software, computing architecture, and higher reliability of software. However, since future advanced software engineering becomes more and more complex, it is facing insecure, changing world, the design, development, operation, and maintenance of any information/software systems have to consider information security issues carefully and seriously. To ensure security of information/software systems has many new issues that are intrinsically different from those issues to ensure reliability and efficiency of information/software systems, and therefore, it requires some new considerations, methodologies, and supporting tools [28].

Cheng has compared traditional and future software engineering, then proposed some new criteria, such as security, continuity, reactive-ability, predictability, anticipatable-ability, self-healing-ability, and autonomous-evolution-ability for future advanced software engineering [28]. We face some new challenges for the new criteria, such as challenges in developing persistent computing systems, anticipatory reasoning-reacting systems, genuine autonomic computing systems and information/software systems with reactive-ability, predicate-ability anticipatable-ability, and autonomous-evolution-ability in future software engineering. Therefore, the problem of solving the challenges is indispensable.

Contract-Based Programming (CBP) [40] can strictly limit and assure the correctness of programs. In CBP, we usually use preconditions, postconditions of the subprograms to ensure to satisfy some requirements of parameters passing and/or returned values between two components. In many cases, we also use some contracts of types to ensure the correctness of building objects, such as type invariant, static predicate, and dynamic predicate.

Furthermore, the latest version of programming language Ada, known as Ada 2012, has written the concept of contract-based programming into an international standard [40]. Ada 2012 is the next generation of the world's premier programming language for engineering safe, secure and reliable software. We should utilize

contract-based with Ada 2012 to solve the new criteria and challenges in future advanced software engineering [1].

Although CBP is important to future advanced software engineering, there is no report about how CBP can solve the criteria and challenges in future advanced software engineering. Therefore, we expounds methods how CBP with Ada 2012 solve the criteria and challenges in future advanced software engineering from five areas, such as security, continuity, reactive-ability, predictability, anticipatable-ability.

We present future advanced software engineering and its challenges, show Contract-Based Programming with Ada 2012, demonstrate methods of solving the challenges of future advanced software engineering with Ada 2012.

8.2 Future Advanced Software Engineering and Its Challenges

Traditional software engineering is established based on the following main quality evaluation criteria for information/software systems: reliability, safety, efficiency, maintainability, usability, cost/performance ratio. Obviously, these traditional quality evaluation criteria are not adequate for future advanced SE including ISE [28].

Cheng has proposed some new quality evaluation criteria for important advanced information/software systems (i.e., those safety-critical, security-critical, reliability critical systems) as follows [28]. (1) Security: This basically means the whole security of a target system, i.e., the ability (measure) of the target system to prevent and protect against any access to any piece of information by unauthorized recipients as well as any intentional but unauthorized destruction or alteration of any piece of information. (2) Continuity (Persistency): This is the ability (measure) of a target system to perform its required functions under stated conditions anytime (continuity) and/or all the time (persistency). Note that this requires that the target system can function well not only when all things are ordinary and normal but also when it is being maintained, upgraded, or reconfigured, it had some trouble, or it is being attacked. (3) Reactivability (Reactive-ability): This is the ability (measure) of a target system to response to any interference coming from its external environment as quickly as possible. Note that this is a quality evaluation criterion stronger than the ordinary "real-time" (even "hard-real-time"). (4) Predictability: This is the ability (measure) of a target system to detect and predict omens of attacks beforehand and then to inform some certain agent(s) (its managers, its users, and so on) with responsibility. (5) Anticipatability (Anticipatable-ability): This is the ability (measure) of a target system to take some anticipatory actions and perform some operations to defend attacks by itself, i.e., taking anticipation to prevent damage and disasters. (6) Self-healing-ability: This is the ability (measure) of a target system to recover from damage by itself. (7) Autonomous-evolution-ability: This is the ability (measure) of a target system to manage a gradual process in which something changes into a different and usually better, more mature, or more complete form by conforming to the system ' s

own laws only, and not subject to some higher ones.

Cheng ' s fundamental consideration to underlie these new quality evaluation criteria is that any-time (all the time) attack to any target system is the rule rather than the exception in this insecure, complex, changing world, and therefore, what we can do is to develop and maintain target systems stronger day after day.

The above new quality evaluation criteria for important advanced information/software systems lead to many new challenges in future advanced SE.

Traditional SE has never explicitly taken the requirement that a computing system should run continuously and persistently into account as an essential and/or general requirement. As a result, in general a traditional computing system often has to stop its running and functioning when it needs to be maintained, upgraded, or reconfigured, it has some trouble, or it is attacked. Can we make computing systems continuously live for functioning persistently? Cheng has proposed "persistent computing", a new methodology and/or paradigm that aim to design and develop continuously dependable and dynamically adaptive reactive-systems, called "Persistent Computing Systems," in order to build continuously available, reliable, and secure systems [20, 23]. There are many challenges in developing persistent computing systems and applying them to practices in the real world.

Almost all existing reactive systems are passive, i.e. the systems only can perform those operations in response to instructions explicitly issued by users or application programs, but have no ability to do something themselves actively. Therefore, a traditional passive reactive system only has some quite weak capability to defend attacks from its external environment. In order to prevent attacks beforehand, it is desired that a reactive system itself can detect and predict omens of attacks anticipatorily and then take some anticipatory actions to inform its users and perform some operations to defend attacks. The next generation of reactive systems should be more active and anticipatory than the traditional reactive systems in order to satisfy the requirements from advanced applications. The present author has proposed a new type of reactive systems, named "Anticipatory Reasoning-Reacting Systems" [19], as a certain class of anticipatory systems [29, 46, 49]. Although we have developed some prototypes of anticipatory reasoning-reacting systems to confirm their effectiveness in practical applications, there are many challenges in developing anticipatory reasoning-reacting systems and applying them to practices in the real world.

Autonomic Computing is IBM ' s vision that aims to develop intelligent, open, and self-managing systems, called "Autonomic Computing Systems," in order to build more automated IT infrastructures [35, 43]. Autonomic computing systems have the four key characteristics and/or fundamental features (so-called self-CHOP): (1) self configuring, i.e., the systems can adapt automatically to dynamically changing environments, (2) self-healing, i.e., the systems can discover, diagnose, and react to disruptions, (3) self-optimizing, i.e., the systems can monitor and tune resources automatically, and (4) self-protecting, i.e., the systems can anticipate, detect, identify, and protect themselves from attacks from anywhere. There are many challenges for us to implement genuine autonomic computing systems [20].

8.3 Contract-Based Programming with Ada 2012

Programming language Ada, the next generation of the world's premier programming language for engineering safe, secure, and reliable software, has evolved to Ada 2012, i.e., the latest version of the Ada language standard [7, 8, 40]. It is intended for large and long-lived systems, and widely used in the worlds of high-integrity, safety-critical, high-security, which are closely related to commercial, military airborne avionics, air traffic control, railroad systems, and medical equipment [30]. Ada 2012 has a lot of changes and extensions from the previous version of Ada 2005, including a seismic shift for supporting contract-based programming explicitly, which effectively improves the reliability of a program [7, 30]. Robert Dewar gave an explanation as follows: "In the context of Software, a 'contract' is an assertion of some property of a program component reflecting a requirement that the component must meet." [30] However, the notation came from the Eiffel that is a programming language, designed by Bertrand Meyer [31]. It is the first programming language introduced *Design by Contract*^{TM} [31].

In spite of this, Ada is the first programming language that integrated the notion "contract-based programming" into an ISO/IEC international standard. By contract-based programming, Ada can protect the program from incorrect composition at run time [7].

Programming language Ada 2012 introduced five types of contracts, which are precondition, postcondition, type_invariant, static_predicate, and dynamic_predicate, respectively. In Ada 2012, one of the most important points is that using new syntax (using with) to introduce aspect specifications which enable certain properties of entities to be stated where they are declared rather than later using representation clauses. It is good to use pre- and postconditions for subprograms and similar assertions for types and subtypes with predicate.

A precondition is an obligation on the caller to ensure that it is true before the subprogram is called and it is a guarantee to the implementer of the body that it can be relied upon on entry to the body [7]; A postcondition is an obligation on the implementer of the body to ensure that it is true on return from the subprogram and it is a guarantee to the caller that it can be relied upon on return [7]; Type_Invariant specify some invariant properties of private types; Static_Predicate and Dynamic_Predicate are used to supply subtype declarations and type declarations as aspect specifications as a type of contracts [8].

8.4 Contract-Based Programming for Future Advanced Software Engineering with Ada 2012

8.4.1 An Overview

We proposed the methods of utilizing CBP with Ada 2012 to develop some systems to solve the criteria and challenges in future advanced software engineering from five areas, such as security, continuity, reactive-ability, predictability, and anticipatable-ability. The systems for challenges can be developed by utilizing CBP

with Ada 2012 for persistent computing, anticipatory reasoning-reacting, genuine autonomic computing, reactive-ability, predictability, and anticipate-ability.

8.4.2 Methods of Developing Persistent Computing Systems with Contract-Based Programming

A persistent computing system is a reactive system that functions continuously anytime without stopping its reactions even when it is being maintained, upgraded, or reconfigured, it had some trouble, or it is being attacked [23]. Persistent computing systems have the two key characteristics and/or fundamental features: 1) persistently continuous functioning, i.e., the systems can function continuously and persistently without stopping its reactions and 2) dynamically adaptive functioning, i.e., the systems can be dynamically maintained, upgraded, or reconfigured during its continuous functioning [23].

Cheng has designed a persistent computing system, which is based on Soft System Bus (SSB) methodology [18, 21]. SSB-based system is a component-based system consisting a group of control components including self-measuring, self-monitoring [18, 21]. In order to implement an SSB-based system, Cheng has proposed some requirements and facilities. Here, by utilizing CBP with Ada 2012, we may design a method to develop an SSB-based system.

First, we can use the contracts to specify some conditions of any component in an SSB-based system in advance.

Second, we can use type_invariant and some assertion functions to measure and monitor system states for self-measuring, self-monitoring, self-recoding. We should specify some conditions for requirements we need. The conditions can be added to central controller/scheduler to monitor and control each data/instruction flow every moment.

Third, we can use preconditions to check data/instructions receiving from data-instruction stations and check data/instructions sending to data-instruction stations.

Fourth, we can use preconditions of data-instruction stations to check specified partners and in cooperation and communication.

Fifth, we can add a condition for every contract condition, which can send a message to central control components for reporting whether or not the component is running exception. If there is an exception, the control components can add, update, replace, and remove the functional components and starting or stopping the running of them.

Sixth, we can use preconditions of data/instruction stations to preserver data/ instructions and postconditions of stations for their retransmission.

Seventh, we can use preconditions to authentication.

Eighth, we can use postconditions to encipherment.

We give an algorithm to develop an SSB-based system as follows:

```
Procedure SSB_Based_System is
   pragma Assertion_Policy (Check);

   function  Component(Arr: in out Component_Type)
```

```
   return Component_Type
      with Pre => Specify_Pre(Arr),
           Post => Specify_Post(Arr);

   Data_Station(Arr: in  Component_Receive_Type,
                Arr: out Component_Send_Type)
   return Component_Send_Type
      with Pre => Check_Data_Receive(Arr)
                  and then Check_Partner(Arr)
                  and then Preserver_Data(Arr),
           Post => Check_Data_Send(Arr)
                  and then Retransmission(Arr);

   procedure Self_Measure(Arr: in
                          Component_Receive_Type;
                          Arr:
                          out Component_Send_Type)
      with Invariant => (
                  if not Satisfy_Requirements(Arr)
                  then
                             Stop_Component;
                          or
                             Replace_Component;
                          or
                             Update_Component;
                          );

   procedure Self_Monitor(Arr:
                        in  Component_Recive_Type;
                           Arr:
                         out Component_Send_Type )
      with Invariant => (
                if not Satisfy_Requirements(Arr)
                          then
                             Stop_Component;
                          or
                             Replace_Component;
                          or
                             Update_Component;
                          );

begin
   null;
end SSB_Based_System;
```

Once we implement an SSB-based system with the algorithm, we bulid continuously available, reliable, and secure systems.

8.4.3 Methods of Developing Anticipatory Reasoning-Reacting Systems with Contract-Based Programming

An ARRS uses logical reasoning to predict and make decisions of active response [19, 50]. The basic idea of logical reasoning method is to explicitly separate the

underlying logic system, reasoning/computing mechanism, and empirical knowledge in any prediction/decision making such that both underlying logic system and empirical knowledge can be revised/replaced/customized in various predictions/decision making processes performed by an area-independent, task-independe nt, general-purpose reasoning mechanism [25, 27, 50].

The method of prediction/decision making by logic reasoning has the following characteristics [25, 27, 50].

First, to explicitly separate the underlying logic system, reasoning/computing mechanism, and empirical knowledge directly results in an adaptive prediction/decision making method such that one can easily develop, modify, and improve the underlying logic system, reasoning/computing mechanism, and empirical knowledge separately to satisfy different requirements from various application areas. Second, as the underlying logic system (i.e., logical validity criterion) and empirical knowledge are separated explicitly, the prediction/decision making results can be evaluated from two aspects: logical aspect that is area/problem independent and empirical aspect that is area/problem dependent. In fact, for any prediction/decision making, because both the predicted thing and its truth must be unknown before the completion of that prediction/decision making process, an absolute, area/problem independent criterion of correctness/validity is intrinsically important to any satisfactory prediction/decision making method. Third, as a characteristic of development and maintenance technology, the underlying logic system, reasoning/computing mechanism, and empirical knowledge can be developed and maintained separately. Fourth, because logic systems and their formal languages are adopted as the absolute, area/problem independent criterion of correctness/validity and representation languages, the approach of prediction/decision making by logic reasoning is more suitable to qualitative methods rather than quantitative methods.

Shi et al. implemented a general purpose ARRS as a persistent computing system for malice defense [50]. Shi et al. designed architecture of the ARRS, which includes the following components: LTDB, ETDB, RDB, observers, filter, formula generator, forward reasoning engine, formula chooser, enactor, actuators, notifier, reporter, maintainer, and soft system bus [50].

LTDB is a logical theorem database, which stores fragments of logical systems [19].

ETDB is an empirical theory database storing anticipatory model, including *world model*, *predictive model*, and *behavioral model*, which express the real world, predictive laws and behavioral patterns of the target domain as empirical theories represented by logical formulas correspondingly.

RDB is a rule database storing *filter rules*, *translation rules*, *interesting formula definitions*, *interesting terms*, and *actions mapping rules*.

Observers monitor activities and status of the networks, host servers, or application services, then send these trivial sensory data to the filter. There are two kinds of observers: inline observers and passive observers. An inline observer is deployed so that the network traffic/data transfer it is monitoring must pass through it, such as a hybrid firewall, a proxy, and a security gateway. A passive observer is deployed so that it monitors a copy of the actual network traffic, NetFlow or

sFlow, host system status, and application logs and status.

Filter filters out the trivial sensory data and generates important and useful information for detection/prediction or decision making.

Formula generator encodes the sensory data into logical formulas used by the forward reasoning engine according to the translation rules.

Forward reasoning engine is a program to automatically draw new conclusions by repeatedly applying inference rules, to given premises and obtained conclusions until some previously specified conditions are satisfied [24]. The forward reasoning engine gets logical formulas translated at the formula generator, fragment of (a) logic system(s), and the anticipatory model, then it deduces predictions/next actions.

Formula chooser chooses nontrivial conclusions, i.e., predictions and next actions, from forward reasoning engine according to interesting formula definitions and interesting terms.

Enactor receives next actions from formula chooser, then gives corresponding instructions according to the actions mapping rules to actuators.

Actuators receive instructions from the enactor, and carry out actions specified by the instructions to stop/prevent malicious behaviors. An actuator could be a firewall (either network firewall or host-based firewall), a router, or a switch, which can block access from the attacker or to the target; An actuator could be a program that kills connections to or from a particular host, network, and port, such as Tcpkill; An actuator could be a security proxy, which can change an attack's content; An actuator could also be an application service, while some application could terminate a network connection or a user session, block access form certain hosts or user accounts.

Notifier notifies security administrators notifications about important detections/predictions/responses about malicious activities. The notifier gives notifications by e-mail, short message, as well as the system's user interface.

Reporter summarizes the detections/predictions/responses about malicious activities, as well as provides details of this information.

Maintainer checks whether there are contradictional theorems in ETDB against a newly added/modified empirical theorem, when we construct/update/modify the anticipatory model. If the contradictional theorems exist, the maintainer lists up theorems deduced from the contradictional theorems in ETDB.

Soft system bus [21] and *central control components* provide a communication channel for each component and an infrastructure for persistent computing [21].

However, the method is quite complex and cannot ensure execution efficiency. The method consists of three steps as follows: First, it is to detect and predict malicious activities; second, it is to choose anticipatory actions; third, it is to take the anticipatory actions to stop/prevent the malicious activities. And then there are four steps to detect/predict malice.

The biggest problem of the method is facing to guarantee the correctness of data that comes from observers monitor activities and status of environment. If data is not correctness, the following operation is unmeaning. So we must guarantee the correctness of data. We may use CBP with Ada 2012 to develop Observe. As contracts, some requirements and/or conditions to guarantee the correctness are

needed.

On the other hand, if we make these functional components of anticipation CBP, the ARRS becomes concise and effective. Here, we give an algorithm as following:

```
Procedure ARRS is
   pragma Assertion_Policy (Check);

   function  Observe(Arr: in Data)
   return Raw_Data
      with Pre => Gurantee(Arr);

   function Generate_Formulas(Arr: in Data;
                              Arr: out Data)
   return Data(Fragment_Logic_Systems)
      with Pre => Filter(Observe(Arr)),
      with Post => Check_Translation_Rule(Arr);

   funtion Formula chooser (Arr: in Predictions)
   return Nontrivial_Conclusions
      with Post => Satify_formula_definitions
                    (Forward_Reasoning(Arr));

   procedure Forward_Reasoning(Arr: in
                          Fragment_Logic_Systems;
                          Arr: out Predictions)
      with Invariant => (
                   if not Satisfy_Requirements
                   then
                      precedure Forward_Reasoning;
                          );

   procedure Enactor(Arr:
                      in  Nontrivial_Conclusions)

      with Pre => (
                  if not Satisfy_Mapping_Rule
                     (Forward_Reasoning(Arr))
                  then
                      precedure Enactor;
                    );

   procedure Notifier(Arr:
                       in Nontrivial_Conclusions)
       with Pre => (
                    if not Check_Security
                       (Forward_Reasoning(Arr))
                    then
                       precedure Notifier;
                    );

begin
   null;
end ARRS;
```

Once we implement an ARRS with the algorithm, we can bulid the systems with Reactivability, Predictability, and Anticipatability.

8.4.4 Disscussion

IBM proposed four key characteristics and/or fundamental features for autonomic computing systems, such as self-configuring, self-healing, self-optimizing, and self-protecting. For CBP, it cannot change environment dynamically, as well as it cannot react to disruptions. Therefore, Utilizing CBP, we cannot construct self-configuring and self-healing autonomic computing systems.However, as above, By CBP, we can monitor resources automatically, as well as CBP must support protect systems essentially. Thus, Utilizing CBP, we can construct self-optimizing and self-protecting autonomic computing systems.

Chapter 9

Conclusions

9.1 Contributions

This thesis presented a software engineer environment for Ada 2012 programs. At first, we defined the defintion-use net (DUN) of Ada 2012 programs to represent multiple control flows, definitions and uses of variables, and inter-process synchronization and communication in a concurrent Ada 2012 program. We designed and implemented a definition-use net generator for Ada 2012 programs, which can automatically generate DUNs of target Ada 2012 programs. By exploiting DUNs of Ada 2012 programs, we can get various programs dependences and interprocedural relations. For Ada 2012 programs, we found some new types of program dependences and one interprocedural relation. For capture SDNs of Ada 2012 programs, we formally defined new types of program depences, and then we defined the SDN. After that, we designed and implemented a system dependence net generator for Ada 2012 programs. In the software engineer environment, we developed a run-time tasking deadlock detector for Ada 2012 programs. At first, we found some types of queue operation related tasking deadlocks in Ada 2012 programs. Then, since there are some new synchronization waiting relations in Ada 2012 programs, we designed and developed a run-time tasking deadlock detector for Ada 2012 programs. Meanwhile, we showed how to utilize CBP with Ada 2012 to solve challenges of future advanced software engineering. We proposed methods and solutions from five areas, such as security, continuity, reactive-ability, predictability, anticipatable-ability. By CBP, we gave algorithms to develop an SSB-based system as a persistent computing system and an ARRS as an anticipatory system. We also disscussed that CBP cannot solve self-healing and self-configuring, but CBP can construct self-optimizing and self-protecting.

9.2 Future Works

In the future, At first, we will develop a program slicer for Ada 2012 programs, which can be designed based on SDNs. Second, a method of deadlock prevention with contract-based programming for Ada 2012 programs should be proposed and implemented. We will energetically apply this software engineering environment

for software engineering practices.

Publications

Refereed papers published in journals or books (first author)

- Bo Wang, Yuichi Goto, and Jingde Cheng: Queue Operation Related Tasking Deadlocks in Ada 2012 Programs, ACM SigAda Ada Letters, Vol. 34, No. 2, pp. 9-25, ACM Press, New York, August 2014.
- Bo Wang, Takeo Ekiba, Yuichi Goto, and Jingde Cheng: A Tasking Deadlock Detector for Ada 2012 Programs, in J. J. Park, H. Chao, H. Arabnia, and N. Y. Yen (Eds.), Advanced Multimedia and Ubiquitous Engineering - Future Information Technology, The 10th FTRA International Conference, FutureTech 2015, Hanoi, Vietnam, May 18-20, 2015, Proceedings, Lecture Notes in Electrical Engineering, Vol. 352, pp. 15-22, Springer, Heidelberg, May 2015.
- Bo Wang, Hongbiao Gao, and Jingde Cheng: A Supporting Environment for Contract-Based Programming with Ada 2012, in J. J. Park, H. Jin, Y. S. Jeong, and M. K. Khan (Eds.), Advanced Multimedia and Ubiquitous Engineering - FutureTech MUE, The 11th FTRA International Conference, FutureTech 2016, Beijing, China, Proceedings, Lecture Notes in Electrical Engineering, Vol. 393, pp. 69-77, Springer, Singapore, August 2016.
- Bo Wang, Hongbiao Gao, and Jingde Cheng: New Definition-Use Nets for Ada 2012 Programs, ACM SigAda Ada Letters, ACM Press (to appear).
- Bo Wang, Hongbiao Gao, and Jingde Cheng: Definition-Use Net and System Dependence Net Generators for Ada 2012 Programs and Their Applications, Ada User Journal, Vol. 38, No. 1, Ada-Europe, March 2017 (to appear).

Refereed papers published in international conference proceedings (first author)

- Bo Wang, Kai Shi, Yuichi Goto, and Jingde Cheng: Generation of System Dependence Nets for Ada 2005 Programs, Proceedings of the 2nd IEEE International Conference on Computer Science and Automation Engineering, CSAE 2012, Zhangjiajie, China, Vol. 3, pp. 401-406, IEEE Press, May 2012.

- Bo Wang, Yuichi Goto, and Jingde Cheng: New Types of Program Dependences and Interprocedural Relations in Ada 2012 Programs, Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science, ICSESS 2013, Beijing, China, pp. 718-723, IEEE Press, May 2013.
- Bo Wang, Hongbiao Gao, and Jingde Cheng: Contract-Based Programming for Future Computing with Ada 2012, Proceedings of 2016 International Conference on Advanced Cloud and Big Data, CBD 2016, Chengdu, China, pp. 322-327, IEEE Computer Society Press, August 2016.

Refereed papers published in journals or books (co-author)

- Zhe Wang, Yuan Zhou, Bo Wang, Yuichi Goto, and Jingde Cheng: An Extension of QSL for E-testing and Its Application in an Offline E-testing Environment, in J. J. Park, H. Chao, H. Arabnia, and N. Y. Yen (Eds.), Advanced Multimedia and Ubiquitous Engineering - Future Information Technology, The 10th FTRA International Conference, FutureTech 2015, Hanoi, Vietnam, May 18-20, 2015, Proceedings, Lecture Notes in Electrical Engineering, Vol. 352, pp. 7-14, Springer, Heidelberg, May 2015.

Refereed papers published in international conference proceedings (co-author)

- Liqing Xu, Bo Wang, Ning Zhang, Yuichi Goto, and Jingde Cheng: Providing Users With Suitable Services of Information Security Engineering Cloud Based on ISO/IEC 15408, Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science, ICSESS 2013, Beijing, China, pp. 321-325, IEEE Press, May 2013.
- Kai Shi, Bo Wang, Yuichi Goto, Zhiliang Zhu, and Jingde Cheng: An Anticipatory Reasoning-Reacting System for Defending Against Malice Anticipatorily, Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science, ICSESS 2013, Beijing, China, pp. 732-737, IEEE Press, May 2013.

Bibliography

[1] AdaCore: Ada 2012. http://www.ada2012.org/, 2012 (accessed 16.11.30).

[2] AdaCore: The Ada Programming Language. http://www.adacore.com/adaanswers/about/ada, 2012 (accessed at 16.11.30).

[3] AdaCore: ASIS-for-GNAT User's Guide. http://docs.adacore.com/asis-docs/asis_ug_3.html, 2014 (accessed at 16.11.30).

[4] AdaCore: Ada Web Server. http://www.adacore.com/aws, 2016 (accessed at 16.11.30).

[5] Butterfield, A., Ngondi, G.E.: A Dictionary of Computer Science, seventh ed. Oxford University Press, Oxford, 2016.

[6] Bove, A., Dybjer, P.: Dependent Types at Work. In: Bove, A., Barbosa, L.S., Pardo, A., Pinto, J.S. (Eds.), Language Engineering and Rigorous Software Development. In: Lecture Notes in Computer Science, 5520, pp. 57-99, 2009.

[7] Barnes, J.: Ada 2012 Rationale: The Language - The Standard Libraries (Lecture Notes in Computer Science 8338 / Programming and Software Engineering). Springer, Heidelberg, 2013.

[8] Barnes, J., 2014. Programming in Ada 2012. Cambridge University Press, Cambridge.

[9] Cheng, J.: A Classification of Tasking Deadlocks. In: ACM Ada Letters, Vol. 10, No. 5, pp. 110-127, ACM, New Year, 1990.

[10] Cheng, J.: Task-Wait-For Graphs and Their Application to Handling Tasking Deadlocks. In: Proc. 3rd ACM Annual TRIAda Conference, pp. 376-390, ACM, New Year, 1990.

[11] Cheng, J.: Task Dependence Net as a Representation for Concurrent Ada Programs. In: Katwijk, J.(Eds.), Ada: Moving towards 2000, 11th Ada-Europe International Conference. In: Lecture Notes in Computer Science, 603, pp. 156-171, 1992.

[12] Cheng, J.: Process Dependence Net of Distributed Programs and Its Applications in Development of Distributed Systems. In: Proc. 17th Annual International Computer Software & Applications Conference. IEEE Computer Society Press, Phoenix, USA, pp. 231-240, 1993.

[13] Cheng, J.: Slicing Concurrent Programs - A Graph-Theoretical Approach. In: Fritzson, P. A. (Eds.), First International Workshop, Automated and Algorithmic Debugging. In: Lecture Notes in Computer Science, 749, pp. 223-240, 1993.

[14] Cheng, J.: Complexity Metrics for Distributed Programs. In: Proc. 4th International Symposium on Software Reliability Engineering. IEEE Computer Society Press, Denver, USA, pp. 132-141, 1993.

[15] Cheng, J.: Nondeterministic Parallel Control-Flow / Definition-Use Nets and Their Applications. In: Joubert, G.R., Trystram, D., Prters, F.J., Evans, D.J. (Eds.), Parallel computing: Trends and Applications. Elsevier B.V., Grenoble, France, pp. 589-592, 1994.

[16] Cheng, J., Ushijima, K.: Tasking Deadlocks in Ada 95 Programs and Their Detection. In: A. Strohmeier (ed.) Reliable Software Technologies - Ada-Europe 1996. Ada-Europe International Conference on Reliable Software Technologies, Montreux, Switzerland, June 10-14, 1996, Proceedings, Lecture Notes in Computer Science, Vol. 1088, pp. 135-146, Springer, Heidelberg, 1996.

[17] Cheng, J.: Task Dependence Nets for Concurrent Systems with Ada 95 and Its Application. In: 1997 ACM TRI-Ada Proceeding of the Conference. ACM Press, St. Louis, USA, pp. 67-78, 1997.

[18] Cheng, J.: Soft System Bus as a Future Software Technology. In: Proc. 8th International Symposium on Future Software Technology, Systems Engineering, Vol. 7, pp. 1-6, 2004.

[19] Cheng, J.: Temporal Relevant Logic as the Logical Basis of Anticipatory Reasoning-Reacting Systems. In: Proc. Computing Anticipatory Systems: CASYS - 6th International Conference, AIP Conference Proceedings, Vol. 718, pp. 362-375, 2004.

[20] Cheng, J.: Comparing Persistent Computing with Autonomic Computing. In: Proc. 11th International Conference on Parallel and Distributed Systems, IEEE, pp. 428-432, 2005.

[21] Cheng, J.: Connecting Components with Soft System Buses: A New Methodology for Design, Development, and Maintenance of Reconfigurable, Ubiquitous, and Persistent Reactive Systems. In: Proc. 19th International Conference on Advanced Information Networking and Applications, IEEE, pp. 667-672, 2005.

[22] Cheng, J.: Run-Time Detection of Tasking Deadlocks in Real Time Systems with the Ada 95 Annex of Real-Time Systems. In: Reliable Software Technologies - Ada-Europe 2006. 11th International Conference on Reliable Software Technologies, Porto, Portugal, June 5-6, 2006, Proceedings, Lecture Notes in Computer Science, Vol. 4006, pp. 167-178, Springer, Heidelberg (2006)

[23] Cheng, J., Shang, F.: Persistent Computing Systems as an Infrastructure of Computing Anticipatory Systems. International Journal of Computing Anticipatory Systems, 18:61-74, 2006.

[24] Cheng, J., Nara, S., Goto, Y.: FreeEnCal: A Forward Reasoning Engine with General-Purpose. In: Proc. 11th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part II, Lecture Notes in Artificial Intelligence, Springer-Verlag, Vol. 4693, pp. 444-452, 2007.

[25] Cheng, J.: Adaptive Prediction by Anticipatory Reasoning Based on Temporal Relevant Logic. In: Proc. 8th International Conference on Hybrid Intelligent Systems, IEEE, pp. 410-416, 2008.

[26] Cheng, J., Goto, Y., Morimoto, S., Horie, D.: A Security Engineering Environment Based on ISO/IEC Standards: Providing Standard, Formal, and Consistent Supports for Design, Development, Operation, and Maintenance of Secure Information Systems. In: Proc. 2nd International Conference on Information Security and Assurance, IEEE Computer Society Press, pp. 350-354, 2008.

[27] Cheng, J.: Adaptive Decision Making by Reasoning Based on Relevant Logics. In: Proc. 9th International FLINS Conference on Foundations and Applications of Computational Intelligence, World Scientific, pp. 541-546, 2010.

[28] Cheng, J.: New Challenges in Future Software Engineering. In: Future Information Technology, FutureTech 2014, Lecture Notes in Electrical Engineering, Springer, Vol. 309, pp. 31-36, 2014.

[29] Dubois, D.M.: Mathematical Foundations of Discrete and Functional Systems with Strong and Weak Anticipations. In: Anticipatory Behavior in Adaptive Learning Systems, Lecture Notes in Artificial Intelligence, Springer, Vol. 2684, pp. 110-132, 2003.

[30] Dewar, R.: Ada 2012: Ada with Contracts. http://www.drdobbs.com/architecture-and-design/ada-2012-ada -with-contracts/240150569, 2013 (accessed 16.11.30).

[31] Eiffel Software: Building bug-free O-O software: An Introduction to Design by Contract. https://www.eiffel.com/values/design-by-contract/introduction/ , 2001 (accessed 16.11.30).

[32] Ekiba, T., Goto, Y., Cheng, J.: New Types of Tasking Deadlocks in Ada 2012 Programs. In: ACM Ada Letters, Vol. 33, pp. 169-179, ACM, New Year (2013)

[33] Ferrante, J., Ottenstein, K.J., Warren, J.D., 1987. The Program Dependence Graph and Its Use in Optimization. ACM Transactions on Programming Languages and Systems. 9(3), pp. 319-349.

[34] Gela Project: Gela ASIS. http://gela.ada-ru.org/gela_asis, 2010 (accessed 16.11.30).

[35] Ganek, A.G., Corbi, T.A.: The Dawning of the Autonomic Computing Era. IBM Systems Journal, 42(1):5-18, 2003.

[36] Horwitz, S., Reps, T., Binkley, D.: Interprocedural Slicing Using Dependence Graphs. ACM Transactions on Programming Languages and Systems. 12(1), 26-60, 1990.

[37] International Organization for Standardization, ISO/IEC 8652:1987 (en): Information Technology - Programming Language - Ada. ISO, Geneva, 1987.

[38] International Organization for Standardization, ISO/IEC 8652:1995 (en): Information Technology - Programming Language - Ada. ISO, Geneva, 1995.

[39] International Organization for Standardization, ISO/IEC 8652:2007 (en): Information Technology - Programming Language - Ada. ISO, Geneva, 2007.

[40] International Organization for Standardization, ISO/IEC 8652:2012 (en): Information Technology - Programming Language - Ada. ISO, Geneva, 2012.

[41] International Organization for Standardization, ISO/IEC 15291:2012 (en): Information Technology - Programming Language - Ada Semantic Interface Specification. ISO, Geneva, 2013.

[42] Kasahara, Y., Cheng, J., Ushijima,K.: Task Dependence Net of Concurrent Ada Programs and Its Automatic Generation. Transactions of IEICE. J79-D-I, pp. 925-935, 1996 (In Japanese).

[43] Kephart, J., Chess, D.: The Vision of Autonomic Computing. Computing, IEEE, 36(1):41-50, 2003.

[44] Nonaka, Y., Hatano, K., Nomura, Y., Cheng, J., Ushijima, K.: A System Dependence Net Generator for Ada Programs. In: Proc. Sixth Asia-Pacific Software Engineering Conference. IEEE Computer Society Press, Takamatsu, Japan, pp. 441-448, 1999.

[45] Nonaka, Y., Cheng, J., Ushijima, K.: A Tasking Deadlock Detector for Ada 95 Programs. In: Ada User Journal, Vol. 20, No. 1, pp. 79-92, 1999.

[46] Nadin, M.: Anticipatory Computing. ACM Ubiquity, 2000(December):4, 2000.

[47] Ottenstein, K.J., Ottenstein, L.M.: The Program Dependence Graph in a Software Development Environment. ACM Sigplan Notices. 19(5), 177-184, 1984.

[48] Podgurski, A., Clarke, L.: A Formal Model of Program Dependences and its Implications for Software Testing, Debugging, and Maintenance. IEEE Transactions on Software Engineering. 16(9), 965–979, 1990.

[49] Rosen, R.: Anticipatory Systems: Philosophical, Mathematical, and Methodological Foundations, 2nd edn. Springer, 2012.

[50] Shi, K., Wang, B., Goto, Y., Zhu, Z., Cheng, J.: An Anticipatory Reasoning-Reacting System for Defending against Malice Anticipatorily. In: Proc. 4th IEEE International Conference on Software Engineering and Service Science (ICSESS), IEEE, pp. 732-737, 2013.

[51] Weiser, M.: Program Slicing. In: Proceedings of the 5th International Conference on Software Engineering. IEEE Press, San Diego, USA, pp. 439-449, 1981.

[52] Weiser, M., 1982. Programmers Use Slices when Debugging. Communications of the ACM. 25(7), 446-452.

[53] Wang, B., Goto, Y., Cheng, J.: New Types of Program Dependences and Interprocedural Relations in Ada 2012 Programs. In: Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science. IEEE Press, Beijing, China, pp. 718-723, 2013.

[54] Yau, S.S., Collofello, J.S., MacGregor, T.M.: Ripple Effect Analysis of Software Maintenance. In: The IEEE Computer Society's Second International, Computer Software and Applications Conference. IEEE Computer Society Press, Chicago, USA, pp. 60-65, 1978.

[55] Zhao, J., Cheng, J., Ushijima, K.: System Dependence Net: An Interprocedural Program Dependence Representation for Occam 2 Programs. In: Noguchi, S., Ota, M. (Eds), Correct Models of Parallel Computing, IOS Press, pp. 87-96, 1997.

Appendix A

A Definition-Use Net Generator for Ada 2012 Programs

A.1 Definition-Use Net Generator

A.1.1 Actuals_For_Traversing Package

```
------------------------------------------------------------------------------
--                                                                          --
--                    ASIS APPLICATION TEMPLATE COMPONENTS                  --
--                                                                          --
--                 A C T U A L S _ F O R _ T R A V E R S I N G              --
--                                                                          --
--                                 S p e c                                  --
--                                                                          --
--           Copyright (c) 2000, Free Software Foundation, Inc.             --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Application Templates were developed and are now maintained by Ada  --
-- Core Technologies Inc (http://www.gnat.com).                             --
--                                                                          --
------------------------------------------------------------------------------

--  This package contains the definition of the actual types and subprograms
--  to be used to instantiate Asis.Iterator.Traverse_Element.
--  Actually, the declarations given in this package are templates in the
--  sense, that, being formally correct and making the whole code of the
--  template ASIS applications compilable, they do nothing and therefore they
--  have to be replaced in the real ASIS applications.
--
--  The bodies of the subprograms declared in this package are implemented
--  as subunits to localize changes needed to provide real Pre- and
--  Post-operations in real operations

with Asis;

package Actuals_For_Traversing is

   type Traversal_State is (Not_Used);
   --  A placeholder declaration, used as an actual for State_Information in
   --  the template instantiation of Traverse_Element.
   --  If your ASIS application needs some non-trivial state, you should
   --  either change the definition of this type (and of the constant
   --  Initial_Traversal_State) below or use some other type as the actual
   --  for State_Information when instantiating Traverse_Element.

   Initial_Traversal_State : constant Traversal_State := Not_Used;

   procedure Pre_Op
     (Element :        Asis.Element;
      Control : in out Asis.Traverse_Control;
      State   : in out Traversal_State);
   --  This is the template for the actual Pre-Operation procedure. It does
   --  nothing, and it contains the exception handler which is supposed to
   --  catch all the exception raised in this procedure.
   --  The body of this procedure is implemented as a subunit - in case when
   --  you  would like to provide your own Pre-operation when building your
   --  ASIS tool from the given set of templates (and if you do not need
   --  non-trivial traversal state), you can replace this subunit by your own
   --  code and reuse the rest of the template code.

   procedure Post_Op
```

```
     (Elem :        Asis.Element;
      Control : in out Asis.Traverse_Control;
      State   : in out Traversal_State);
   --  This is the template for the actual Post-Operation procedure. It does
   --  nothing, and it contains the exception handler which is supposed to
   --  catch all the exception raised in this procedure.
   --  The body of this procedure is implemented as a subunit - in case when
   --  you  would like to provide your own Post-operation when building your
   --  ASIS tool from the given set of templates (and if you do not need
   --  non-trivial traversal state), you can replace this subunit by your own
   --  code and reuse the rest of the template code.

end Actuals_For_Traversing;




------------------------------------------------------------------------------
--                                                                          --
--                    ASIS APPLICATION TEMPLATE COMPONENTS                  --
--                                                                          --
--                 A C T U A L S _ F O R _ T R A V E R S I N G              --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--           Copyright (c) 2000, Free Software Foundation, Inc.             --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Application Templates were developed and are now maintained by Ada  --
-- Core Technologies Inc (http://www.gnat.com).                             --
--                                                                          --
------------------------------------------------------------------------------

--Bo Wang reconstitution 2014.12


with Asis;
with Asis.Errors;
with Asis.Exceptions;
with Asis.Implementation;
with Asis.Ada_Environments;
with Asis.Compilation_Units;
with Asis.Elements;
with Asis.Iterator;
with Asis.Declarations;
with Asis.Definitions;
with Asis.Expressions;
with Asis.Statements;
with Asis.Text;
with Gela_Ids;
with ID_List;
--with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;
with Ada.Text_IO;
with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;
with Ada.Command_Line;
with Ada.Characters.Handling;
use Ada.Characters.Handling;

with Ada.Calendar;

with Dun_Handler, V_Strings, Stacks;
use Dun_Handler, V_Strings;

with Ada.Integer_Text_IO;
with Ada.Wide_Text_IO;
with Ada.Characters.Handling;
with Ada.Exceptions;

with Asis.Exceptions;
with Asis.Errors;
with Asis.Implementation;
with Asis.Elements;
with Asis.Declarations;
with Asis.Text;

with Metrics_Utilities;

package body Actuals_For_Traversing is
   package Line_IO is new Integer_IO(Asis.Text.Line_Number);
   package Node_IO is new Integer_IO(Nodenum_T);
   package Int_IO is new Integer_IO(Integer);

   Skip_Process : Boolean := False;
   My_Context : Asis.Context;
   The_Unit : Asis.Compilation_Unit;
   The_Unit_Spec : Asis.Compilation_Unit;
   The_Unit_Body : Asis.Compilation_Unit;
   Current_Statement : Nodenum_T;
--   Args_Nodenum : Nodenum_T;
   My_Args_Item : Args_Item_Link;
   Caller_Nodenum : Nodenum_T := 0; -- function call

   Disc_Range_Flag : Boolean := False; -- ASIS bug

   use Varnum_Stacks;
   use Var_Item_Stacks;
   use Vstring_Stacks;
```

```
   function Get_Varnum(Ename : Asis.Expression) return Varnum_T is
      -- Expression
      -- in  : expression
      -- out : var num
      use Asis;
      Temp_Def : Asis.Defining_Name := Nil_Element;
   begin
      if Is_Debug_Mode then
         Put_Line("get_varnum");
         Put_Line(Asis.Text.Element_Image(Ename));
--       Put_Line(Asis.Elements.Debug_Image(Asis.Elements.Enclosing_Element(Ename)));
--       Put_Line(Asis.Elements.Debug_Image(Ename));
      end if;

      declare
      begin

         if Asis.Elements.Element_Kind(Ename) = A_Defining_Name then
            Temp_Def := Ename;
         else
            select
               delay 0.1;
               Put_Line("warning: Corresponding_Name_Definition is canceled.");
            then abort
              Temp_Def := Asis.Expressions.Corresponding_Name_Declaration(Ename);--Corresponding_Name_Definition
            end select;
         end if;
--       if Temp_Def = Nil_Element then
         if Asis.Elements.Is_Nil(Temp_Def) then
            return 0;
         else
            return Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Temp_Def));
         end if;
      exception
         when Asis.Exceptions.Asis_Inappropriate_Element =>
            Put_Line("warning : raised Asis.Inappropriate_Element in get_varnum!");
         when Asis.Exceptions.Asis_Failed =>
            Put_Line("warning : raised Asis_Failed in get_varnum!");

      end;
      return 0;
   end Get_Varnum;


   function Is_Type_Equal(A, B : Asis.Expression) return Boolean is
     -- ASIS-for-gnat bug
     Result : Boolean := False;
     use Asis;
   begin
--       select
--       delay 0.2;
--       then abort
--       Result := (Asis.Expressions.Corresponding_Expression_Type(A)
--          = Asis.Expressions.Corresponding_Expression_Type(B));
         Result := Asis.Elements.Is_Equal
           (Asis.Expressions.Corresponding_Expression_Type(A),
            Asis.Expressions.Corresponding_Expression_Type(B));
--       end select;
      return Result;
   end Is_Type_Equal;

--    -- expression
--    procedure Analyze_Name(Exp : in Asis.Expression;
--                           Varnum : out Varnum_T;
--                           Is_Normalize : out Boolean;
--                           Normalized_Form : out V_String) is
--    begin
--       case Asis.Elements.Expression_Kind(Exp) is
--       when An_Identifier =>
--       when An_Explicit_Dereference =>
--       when An_Indexed_Component =>
--       when A_Selected_Component =>
--       when An_Attribute_Reference =>
--       when



   package Nodenum_stacks is new stacks(Nodenum_T);
   use Nodenum_Stacks;
   -- terminate
   taskhead_stack: nodenum_stacks.stack:= null;
   -- return
   Subprogramhead_Stack: Nodenum_Stacks.Stack := null;
   -- requeue
   Accepthead_Stack : Nodenum_Stacks.Stack := null;

   package Fnode_Stacks is new Stacks(Function_Nodes);
   use Fnode_Stacks;
   -- function
   Fnode_Stack : Fnode_Stacks.Stack := null;

--   package Args_Stacks is new Stacks(Args);
   use Args_Stacks;
--   Args_Stack : Args_Stacks.Stack := null;

   package Call_Stacks is new Stacks(Subprogram_Call);
   use Call_Stacks;

   Call_Stack : Call_Stacks.Stack := null;

      procedure Use_Arg_Stacks is
--       Self_Info : Element_Info := Get_Info(Elem);
--       Self_Node : Nodenum_T;
         Arg_Node_Item : Node_Item_Link;
      begin
--       if Self_Info /= Null_Element_Info then
--          Self_Node := Self_Info.Top;
         while not Nodenum_Stack.Isempty(Arg_Join_Stack) loop
```

```
            Arg_Node_Item := Search_Node(Nodenum_Stack.Top(Arg_Join_Stack));
            if Arg_Node_Item.Branch = null then
               Add_Branch(Nodenum_Stack.Top(Arg_Join_Stack), Num_Of_Nodes +1);
            end if;
            Nodenum_Stack.Remove(Arg_Join_Stack);
         end loop;
--       else
--          Nodenum_Stack.Clear(Arg_Join_Stack);
--       end if;
      end Use_Arg_Stacks;

   function Is_Access(Decl : in Asis.Declaration) return Boolean is


      Name_Def : Asis.Defining_Name := Asis.Expressions.Corresponding_Name_Definition
      (Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Decl)));
      Type_Def : Asis.Definition;
      use Asis;
   begin
--       if Name_Def = Nil_Element then
      if Asis.Elements.Is_Part_Of_Implicit(Name_Def) then
         return False;
      end if;
      Type_Def := Asis.Elements.Enclosing_Element(Name_Def);
--      Put_Line(Asis.Text.Element_Image(Type_Def));
--      Put_Line(Asis.Elements.Debug_Image(Asis.Declarations.Type_Declaration_View(Type_Def)));
      if Asis.Elements.Type_Kind(Asis.Declarations.Type_Declaration_View(Type_Def)) = An_Access_Type_Definition then
--       Put_Line("access!");
         return True;
      end if;
      return False;
   end Is_Access;

   procedure Set_Is_Access(Decl : in Asis.Declaration; Is_Access : out Boolean;
                           Type_Ident : in out V_String) is
     Name_Def : Asis.Defining_Name;
     Type_Def, Temp_Def, Tmp_Def : Asis.Definition;
     Temp_Exp, Tmp_Exp: Asis.Expression;
      use Asis;
   begin
      Put_Line("248");
      Put_Line(Asis.Elements.Debug_Image(Decl));
      case Asis.Elements.Declaration_Kind(Decl) is
         when A_Parameter_Specification | A_Formal_Object_Declaration =>
            Put_Line("251");
            Put_Line(Asis.Elements.Debug_Image(Decl));
            Tmp_Def := Asis.Declarations.Object_Declaration_View(Decl);
            Put_Line(Asis.Elements.Debug_Image(Tmp_Def));
--            Temp_Exp := Asis.Declarations.Declaration_Subtype_Mark(Decl);
            while asis.Elements.Attribute_Kind(Tmp_Def) /= Not_An_Attribute loop
               Tmp_Def := Asis.Expressions.Prefix(Tmp_Def);
            end loop;
            Put_Line(Asis.Text.Element_Image(Tmp_Def));
            <<Set_Is_Access_Check_Tmp_Def_Type>>
            if Asis.Elements.Expression_Kind(Tmp_Def) = An_Identifier then
               Name_Def := Asis.Expressions.Corresponding_Name_Definition(Tmp_Def);
            elsif Asis.Elements.Expression_Kind(Tmp_Def) = A_Selected_Component then
               Put_Line(Asis.Text.Element_Image(Tmp_Def));
               Name_Def := Asis.Expressions.Corresponding_Name_Definition(Asis.Expressions.Selector(Tmp_Def));
               Put_Line("259");
            elsif Asis.Elements.Access_Definition_Kind(Tmp_Def) = An_Anonymous_Access_To_Procedure then
               Put_Line(Asis.Elements.Debug_Image(Tmp_Def));
            elsif asis.Elements.Access_Definition_Kind(Tmp_Def) = An_Anonymous_Access_To_Function or
                  Asis.Elements.Access_Definition_Kind(Tmp_Def) = An_Anonymous_Access_To_Protected_Function then
                  Tmp_Def := Asis.Definitions.Access_To_Function_Result_Profile(Tmp_Def);
                  goto Set_Is_Access_Check_Tmp_Def_Type;
            elsif Asis.Elements.Access_Definition_Kind(Tmp_Def) = An_Anonymous_Access_To_Variable then
               Tmp_Exp := Asis.Definitions.Anonymous_Access_To_Object_Subtype_Mark(Tmp_Def);
               if Asis.Elements.Attribute_Kind(Tmp_Exp) = A_Class_Attribute then
                  Tmp_Exp := Asis.Expressions.Prefix(Asis.Definitions.Anonymous_Access_To_Object_Subtype_Mark(Tmp_Def));
               end if;
               Name_Def := Asis.Expressions.Corresponding_Name_Definition(Tmp_Exp);
               --Put_Line(Asis.Elements.Debug_Image(Asis.Definitions.Anonymous_Access_To_Object_Subtype_Mark(Tmp_Def)));
            end if;
         when A_Component_Declaration =>
            Put_Line("263");
            Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Component_Subtype_Indication(Asis.Declarations.Object_Declaration_View(Decl)));
            while Asis.Elements.Expression_Kind(Temp_Exp) = A_Selected_Component loop
               Temp_Exp := Asis.Expressions.Selector(Temp_Exp);
            end loop;
            Name_Def := Asis.Expressions.Corresponding_Name_Definition(Temp_Exp);
         when others =>
            Put_Line("266");
            Put_Line(Asis.Elements.Debug_Image(Decl));
            Put_Line(Asis.Text.Element_Image(Decl));
            if Asis.Elements.Declaration_Kind(Decl) = A_Variable_Declaration then
               Put_Line("a variable declaration");
            end if;
            Temp_Def := Asis.Declarations.Object_Declaration_View(Decl);
            case Asis.Elements.Type_Kind(Temp_Def) is
               when A_Constrained_Array_Definition
                  | An_Unconstrained_Array_Definition =>
                  if Asis.Elements.Expression_Kind(Asis.Definitions.Subtype_Mark(Asis.Definitions.Component_Subtype_Indication(
                  Asis.Definitions.Array_Component_Definition(Temp_Def)))) = A_Selected_Component then
                     Name_Def := Asis.Expressions.Corresponding_Name_Definition(Asis.Expressions.Selector(
                     Asis.Definitions.Subtype_Mark(Asis.Definitions.Component_Subtype_Indication(Asis.Definitions.Array_Component_Definition(Temp_Def)))));
                  else
                     Name_Def := Asis.Expressions.Corresponding_Name_Definition(Asis.Definitions.Subtype_Mark(
                     Asis.Definitions.Component_Subtype_Indication(Asis.Definitions.Array_Component_Definition(Temp_Def))));
                  end if;
               when others =>
                  if Asis.Elements.Access_Definition_Kind(Temp_Def) = An_Anonymous_Access_To_Procedure then
                     Temp_Def := Asis.Declarations.Initialization_Expression(Decl);
                     while Asis.Elements.Attribute_Kind(Temp_Def) /= Not_An_Attribute loop
                        Temp_Def := Asis.Expressions.Prefix(Temp_Def);
                     end loop;
                     Temp_Exp := Temp_Def;
                  else
                     Temp_Exp := Asis.Definitions.Subtype_Mark(Temp_Def);
```

```
                end if;
                if Asis.Elements.Expression_Kind(Temp_Exp) = An_Identifier
                then
                   Name_Def :=
                     Asis.Expressions.Corresponding_Name_Definition(Temp_Exp);
                elsif Asis.Elements.Attribute_Kind(Temp_Exp) = A_Class_Attribute then
                   Name_Def := Asis.Expressions.Corresponding_Name_Definition(Asis.Expressions.Prefix(Temp_Exp));
                else
                   Name_Def :=
                     Asis.Expressions.Corresponding_Name_Definition(Asis.Expressions.Selector(Temp_Exp));
                end if;
          end case;
      end case;
      if Is_Debug_Mode then
         Put_Line("set_is_access");
      end if;
--       if Name_Def = Nil_Element then
      if Asis.Elements.Trait_Kind(Decl) = An_Aliased_Trait then
         Type_Ident := "Aliased-" & Type_Ident;
      end if;
      if Asis.Elements.Is_Part_Of_Implicit(Name_Def) then
         Is_Access := False;
         return;
      end if;
      -- if Asis.Elements.Declaration_Kind
      Type_Def := Asis.Declarations.Type_Declaration_View(Asis.Elements.Enclosing_Element(Name_Def));
      if Is_Debug_Mode then
         Put_Line(Asis.Text.Element_Image(Type_Def));
--       Put_Line(Asis.Elements.Debug_Image(Asis.Declarations.Type_Declaration_View(Type_Def)));
      end if;
      if Asis.Elements.Type_Kind(Type_Def) = An_Access_Type_Definition then
         Is_Access := True;
         case Asis.Elements.Access_Type_Kind(Type_Def) is
            when An_Access_To_Variable | An_Access_To_Constant =>
               Put_Line("298");
               Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Access_To_Object_Definition(Type_Def));
               while Asis.Elements.Expression_Kind(Temp_Exp) = An_Attribute_Reference loop
                  Temp_Exp := Asis.Expressions.Prefix(Temp_Exp);
               end loop;
               if Asis.Elements.Expression_Kind(Temp_Exp) = A_Selected_Component then
                  Temp_Exp := Asis.Expressions.Selector(Temp_Exp);
               end if;
               Type_Ident := "Aliased-" & To_V(Asis.Expressions.Name_Image(Temp_Exp));
               Put_Line("300");
            when others =>
               null;
         end case;
      else
         Is_Access := False;
      end if;


   end Set_Is_Access;

   --
   -- aliased access
   function Get_Type_Ident(Decl : Asis.Declaration) return V_String is
      Temp_Def : Asis.Definition;
      Temp_Exp : Asis.Expression;
      Task_Decl : Asis.Declaration;
      Task_Type : Task_Type_Info_Link;
      Keep_Next : Task_Type_Item_Link;
      use Asis;
   begin
      Put_Line("check Get Type Ident");
      --Put_Line(Asis.Text.Element_Image(Decl));
      Put_Line(Asis.Elements.Debug_Image(Decl));
      --case Asis.Elements.Declaration_Kind(Decl) is
        -- when A_Parameter_Specification =>
         --   Put_Line("328");
         --   Temp_Exp := Asis.Declarations.Object_Declaration_View(Decl);
          --  Put_Line(Asis.Text.Element_Image(Temp_Exp));
         --   Put_Line("331");
            --Temp_Exp := Asis.Declarations.Declaration_Subtype_Mark(Decl);
--       when A_Loop_Parameter_Specification =>
--          Temp_Def := Asis.Declarations.Specification_Subtype_Definition(Decl);
--          Temp_Exp := Asis.Definitions.Subtype_Mark(Temp_Def);
        -- when A_Formal_Object_Declaration =>
         --   Temp_Exp := Asis.Declarations.Declaration_Subtype_Mark(Decl);
         --when others =>
            Temp_Def := Asis.Declarations.Object_Declaration_View(Decl);
          case Asis.Elements.Expression_Kind(Temp_Def) is
              when An_Identifier | A_Selected_Component | An_Attribute_Reference =>
              Temp_Exp := Temp_Def;
              when others =>
            case Asis.Elements.Definition_Kind(Temp_Def) is
               when An_Access_Definition =>
                  case Asis.Elements.Access_Definition_Kind(Temp_Def) is
                     when An_Anonymous_Access_To_Procedure =>
                        Temp_Exp := Asis.Declarations.Initialization_Expression(Decl);
                     when An_Anonymous_Access_To_Function | An_Anonymous_Access_To_Protected_Function =>
                          Temp_Exp := Asis.Definitions.Access_To_Function_Result_Profile(Temp_Def);
                            while Asis.Elements.Access_Definition_Kind(Temp_Exp) /= Not_An_Access_Definition loop
                              Temp_Exp := Asis.Definitions.Access_To_Function_Result_Profile(Temp_Exp);
                            end loop;
                     when An_Anonymous_Access_To_Constant | An_Anonymous_Access_To_Variable =>
                          Temp_Exp := Asis.Definitions.Anonymous_Access_To_Object_Subtype_Mark(Temp_Def);
                     when others =>
                        null;
                  end case;
               when A_Component_Definition =>
                  Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Component_Subtype_Indication(Temp_Def));
               when A_Type_Definition =>
                  case Asis.Elements.Type_Kind(Temp_Def) is
                     when A_Constrained_Array_Definition
                       | An_Unconstrained_Array_Definition =>
                        Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Component_Subtype_Indication(Asis.Definitions.Array_Component_Definition(Temp_Def)));
                     when others =>
                        Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Component_Subtype_Indication(Temp_Def));
```

```
                end case;
             when others =>
                Temp_Exp := Asis.Definitions.Subtype_Mark(Temp_Def);
          end case;
        end case;
     Put_Line(Asis.Elements.Debug_Image(Temp_Def));
     Put_Line(Asis.Elements.Debug_Image(Temp_Exp));
     --end case;
   if Is_Debug_Mode then
        Put_Line(Asis.Elements.Debug_Image(Temp_Exp));
    end if;
     --if Asis.Elements.Attribute_Kind(Temp_Exp) = A_Class_Attribute then
     --   Put_Line("not a expression");
     --end if;
     while asis.Elements.Attribute_Kind(Temp_Exp) /= Not_An_Attribute loop
        Put_Line(Asis.Text.Element_Image(Temp_Exp));
        Temp_Exp := Asis.Expressions.Prefix(Temp_Exp);
     end loop;
     Put_Line(Asis.Text.Element_Image(Temp_Exp));
     if Asis.Elements.Expression_Kind(Temp_Exp) = A_Selected_Component then
        Temp_Exp := Asis.Expressions.Selector(Temp_Exp);
     end if;
     -- task type
     Put_Line("413");
     Put_Line(Asis.Text.Element_Image(Temp_Exp));
     Put_Line("415");
     Task_Decl := Asis.Expressions.Corresponding_Name_Declaration(Temp_Exp);
     if Asis.Elements.Declaration_Kind(Task_Decl) = A_Task_Type_Declaration then
        Task_Type := Task_Type_Id_List.Search_Node(Task_Type_Decl_List, Gela_Ids.Create_Id(Task_Decl));
        if Task_Type /= null then
           Keep_Next := Current_Unit.Decl_By_Task_Type;
           Current_Unit.Decl_By_Task_Type := new Task_Type_Item;
           Current_Unit.Decl_By_Task_Type.Info := Task_Type;
           Current_Unit.Decl_By_Task_Type.Is_Access_Type := False;
           Current_Unit.Decl_By_Task_Type.Next := Keep_Next;
        end if;
     end if;
     return To_V(Asis.Expressions.Name_Image(Temp_Exp));
  end Get_Type_Ident;

  procedure Make_Arg_From_Proc(Decl : Asis.Declaration) is
     Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Decl);
     Position : Natural := 1;
     use Asis;
  begin

     if Plist /= Nil_Element_List then
        for I in Plist'Range loop
           declare
              DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
           begin
              for J in Dname_List'Range loop

                 Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));
                 Position := Position + 1;
              end loop;
           end;
        end loop;
     end if;
  end Make_Arg_From_Proc;


  procedure Push_Exit_Node(Loop_Element_ID : in Gela_Ids.Id;
                           Exit_Node : in Nodenum_T) is
     Tmp_Node : Nodenum_Stack.Stack := Node_Stack_List.Search_Node(Exit_Stack_List, Loop_Element_ID);
  begin
--      if Nodenum_Stack.Isempty(Tmp_Node) then
     Nodenum_Stack.Push(Tmp_Node, Exit_node);
     Node_Stack_List.Set_Node(Exit_Stack_List, Loop_Element_ID, Tmp_Node);
--      else
--       Nodenum_Stack.Push(Tmp_Node, Exit_node);
--      end if;
  end Push_Exit_Node;

  procedure Push_Exit_Node(Loop_Element : in Asis.Element;
                           Exit_Node : in Nodenum_T) is
  begin
     Push_Exit_Node(Gela_Ids.Create_Id(Loop_Element), Exit_Node);
  end Push_Exit_Node;

  -- Element
  function Pop_Exit_Node(Loop_Element_ID : Gela_Ids.Id) return Nodenum_T is
     Tmp_Node : Nodenum_Stack.Stack := Node_Stack_List.Search_Node(Exit_Stack_List, Loop_Element_ID);
     Tmp_Nodenum : Nodenum_T;
  begin
     if Nodenum_Stack.Isempty(Tmp_Node) then
        return 0;
     else
        Nodenum_Stack.Pop(Tmp_Node, Tmp_Nodenum);
        Node_Stack_List.Set_Node(Exit_Stack_List, Loop_Element_ID, Tmp_Node);
        return Tmp_Nodenum;
     end if;
  end Pop_Exit_Node;

  function Pop_Exit_Node(Loop_Element : Asis.Element) return Nodenum_T is
  begin
     return Pop_Exit_Node(Gela_Ids.Create_ID(Loop_Element));
  end Pop_Exit_Node;


  -- Traverse Control 's  Pre Operation.
  -- evaluation
  -- Traverse Control
  -------------
  -- Post_Op --
  -------------

  procedure Post_Op
    (Elem :        Asis.Element;
```

```
      Control : in out Asis.Traverse_Control;
      State   : in out Traversal_State)
   is separate;

   ------------
   -- Pre_Op --
   ------------

   procedure Pre_Op
     (Element :        Asis.Element;
      Control : in out Asis.Traverse_Control;
      State   : in out Traversal_State)
   is separate;
--   procedure Generate_DUN is new Asis.Iterator.Traverse_Element
--     (Boolean, Pre_Op, Post_Op);


end Actuals_For_Traversing;
```

A.1.2 Actuals_For_Traversing-Pre_Op Package

```
------------------------------------------------------------------------------
--                                                                          --
--                       ASIS TUTORIAL COMPONENTS                           --
--                                                                          --
--          A C T U A L S _ F O R _ T R A V E R S I N G . P R E _ O P       --
--                                                                          --
--                                B o d y                                   --
--                                                                          --
--           Copyright (c) 2000, Free Software Foundation, Inc.             --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Tutorial was developed and are now maintained by Ada Core           --
-- Technologies Inc (http://www.gnat.com).                                  --
--                                                                          --
------------------------------------------------------------------------------

--  This is the body of Pre_Op to be used as an example of the metrics tool
--  solution (Task 2) built on top of the ASIS Application Templates
--  provided in ASIS-for-GNAT. This file is supposed to replace the file with
--  the same name which is a part of the ASIS Application Templates

--Bo Wang reconstitution 2014.12

with Asis;
use Asis;
with Asis.Errors;
with Asis.Exceptions;
with Asis.Implementation;
with Asis.Ada_Environments;
with Asis.Compilation_Units;
with Asis.Elements;
with Asis.Iterator;
with Asis.Declarations;
with Asis.Definitions;
with Asis.Expressions;
with Asis.Statements;
with Asis.Text;
with Gela_Ids;
with ID_List;
with Asis.Set_Get;

with Ada.Wide_Text_IO; use Ada.Wide_Text_IO;
with Ada.Text_IO;
with Ada.Command_Line;
with Ada.Characters.Handling;
use Ada.Characters.Handling;
with Ada.Calendar;
with Dun_Handler, V_Strings, Stacks;
use Dun_Handler, V_Strings;

with Ada.Integer_Text_IO;
with Ada.Wide_Text_IO;
with Ada.Characters.Handling;
with Ada.Strings.Wide_Maps;

with Asis.Exceptions;
with Asis.Errors;
with Asis.Implementation;
with Asis.Elements;
with Asis.Declarations;
with Asis.Text;

with Metrics_Utilities;

separate (Actuals_For_Traversing)


procedure Pre_Op
  (Element :        Asis.Element;
   Control : in out Asis.Traverse_Control;
   State   : in out Traversal_State)
is
   Argument_Kind      : Asis.Element_Kinds;
```

```
   --Arg_Statement_Kind : Asis.Statement_Kinds;
   Arg_Declaration_Kind : Asis.Declaration_Kinds;
   --Arg_Pragma_Kind : Asis.Pragma_Kinds;
   --Arg_Defining_Name_Kind : Asis.Defining_Name_Kinds ;
   Arg_Definition_Kind : Asis.Definition_Kinds;
   Arg_Expression_Kind : Asis.Expression_Kinds;
   Arg_Path_Kind : Asis.Path_Kinds;
   --Arg_Clause_Kind : Asis.Clause_Kinds;
   --Arg_Representation_Clause_Kind : Asis.Representation_Clause_Kinds;
   Arg_Association_Kind : Asis.Association_Kinds;

      use Asis;
      -- assign_stack
      -- Define
      procedure Assign_Stacks is
         Self_Node : Nodenum_T := Get_Info(Element).Top;
      begin
         while not isempty(assign_stack) loop
            assign_variable(top(assign_stack), Self_Node);
            remove(assign_stack);
         end loop;
      end Assign_Stacks;

      --
      -- package declaration
      procedure Dlist_Add(Dlist : in Asis.Element_List) is
        Tmp_Nodenum : Nodenum_T;
      begin

         if Dlist /= Nil_Element_List then
            for I in Dlist'Range loop

               if Asis.Elements.Element_Kind(Dlist(I)) = A_Declaration then
                  case Asis.Elements.Declaration_Kind(Dlist(I)) is
                     when A_Variable_Declaration |
                       A_Constant_Declaration |
                       A_Formal_Object_Declaration =>

                        if Is_Debug_Mode then
                           Put_Line(Asis.Text.Element_Image(Dlist(I)));
                        end if;
                        declare
                           Init_Exp : Asis.Expression := Asis.Declarations.Initialization_Expression(Dlist(I));
                           DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Dlist(I));
                           Type_Ident : V_String := Get_Type_Ident(Dlist(I));
--                         Decl_Is_Access : Boolean := Is_Access(Dlist(I));
                           Decl_Is_Access : Boolean;
                        begin
                           if Is_Debug_Mode then
                              Put_Line("pre_is_access");
                           end if;
                           Set_Is_Access(Dlist(I), Decl_Is_Access, Type_Ident);
                           if Is_Debug_Mode then
                              Put_Line("post_is_access");
                           end if;
                           Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Dlist(I))));
                           Set_Info(Dlist(I), (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));

                           Caller_Nodenum := Tmp_Nodenum;
                           for J in Dname_List'Range loop

                              if Is_Debug_Mode then
                                 Put_Line("declared in a block: " &
                                          Asis.Declarations.Defining_Name_Image(Dname_List(J)));
                              end if;


--                            Assign_Variable(Enter_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J)))), Tmp_Nodenum);
                              Assign_Variable(Enter_Variable(Dname_List(J), Type_Ident, Decl_Is_Access), Tmp_Nodenum);

                              if Is_Debug_Mode and Asis.Elements.Declaration_Kind(Dlist(I)) /= A_Formal_Object_Declaration then
                                 Put(Asis.Declarations.Defining_Name_Image(Dname_List(J)));
                                 Put(":");
                                 Put_Line(Asis.Text.Element_Image(Asis.Declarations.Object_Declaration_View(Dlist(I))));
--                               Put_Line(Asis.Expressions.Name_Image(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Dlist(I)))));
--                               Put_Line(Asis.Elements.Debug_Image(Asis.Declarations.Object_Declaration_View(Dlist(I))));
--                               Put_Line(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names(Asis.Declarations.Object_Declaration_View(Dlist(I)))(1)));
                              end if;
                           end loop;
                        end;
                     when A_Package_Declaration =>
                        Enter_Package_Frame(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names(Dlist(I)) (1))),
                                            Asis.Elements.Declaration_Kind(Dlist(I)));
                        declare
                           Vlist : Asis.Element_List := Asis.Declarations.Visible_Part_Declarative_Items(Dlist(I));
                           Plist : Asis.Element_List := Asis.Declarations.Private_Part_Declarative_Items(Dlist(I));
                        begin

                           -- add node
                           Dlist_Add(Vlist);

                           if Asis.Declarations.Is_Private_Present(Dlist(I)) then
                              Dlist_Add(Plist);
                           end if;
                        end;
                        Exit_Package;
                     when A_Package_Body_Declaration |
                       A_Protected_Type_Declaration |
                       A_Single_Protected_Declaration =>

                        Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Dlist(I))));
--                      End_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Dlist(I))));
                        Set_Info(Dlist(I), (Tmp_Nodenum, Tmp_Nodenum +1 , Tmp_Nodenum + 1));
                     when others =>
                        null;
                  end case;
               end if;
            end loop;
```

```
         end if;
         if Is_Debug_Mode then
            Put_Line("end dlist_add");
         end if;
      end Dlist_Add;


      Tmp_NodeNum : Nodenum_T;
      Info : Element_Info;
   --  This is added for Task 2
begin
   --Put_Line("check line 25 bug");
   --Put_Line("begin of the pre op");
  --Put_Line(to_s(Current_Unit.Name));
   --  Note, that the code below may be rewritten in more compact way (with
   --  the same functionality). But we prefer to go step-by-step,
   --  demonstrating the important ASIS queries
   --Put_Line("start pre-op");
   --Put_Line("current unit: " & to_s(Current_Unit.Name));
   Argument_Kind := Asis.Elements.Element_Kind (Element);
   Put_Line(asis.Text.Element_Image(Element));
   --Ada.Integer_Text_IO.Put(Asis.Elements.Hash(Element));
   --Ada.Wide_Text_IO.New_Line;
   case Argument_Kind is
      when Asis.An_Association =>
         Put_Line("check association");
         Arg_Association_Kind := Asis.Elements.Association_Kind (Element) ;
         Case Arg_Association_Kind is
            when Asis.A_Parameter_Association =>
               if Skip_Process = True then
                  goto End_Of_Parameter_Association;
               end if;
                  declare
                     Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
                     Param_Name : V_String;
                     Temp_Exp : Asis.Expression;
                     Arg_Item : Args_Item_Link;
                  begin
                     if Current_Call.Is_Same_Compilation_Unit then
                        if Asis.Expressions.Is_Normalized(Element) then
                           Param_Name := To_V(Asis.Declarations.Defining_Name_Image(Asis.Expressions.Formal_Parameter(Element)));
                           Arg_Item := Search_Args_Item(Current_Call.Args, Param_Name);
                        else
                           Temp_Exp := Asis.Expressions.Formal_Parameter(Element);
--                         if Temp_Exp = Nil_Element then
                           if Asis.Elements.Is_Nil(Temp_Exp) then
                              Arg_Item := Current_Call.Args;
                              --  post op.
                           else
                              Arg_Item := Search_Args_Item(Current_Call.Args, To_V(Asis.Expressions.Name_Image(Temp_Exp)));
                           end if;
                        end if;
                        if Is_Param_Node_Approach then
                           --
                           if Arg_Item.Mode /= An_Out_Mode then
                              if Current_Call.Callkind /= An_Entry_Call then
                                 Set_Arg_Branch(Current_Call.First_Node);
                                 Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                                 Remove_Branch(Tmp_Nodenum-1, Tmp_Nodenum);
                                 Add_Branch(Current_Call.First_Node, Tmp_Nodenum);
                                 if Arg_Item.Mode = A_Default_In_Mode or Arg_Item.Mode = An_In_Mode then
                                    Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                                    Set_Arg_Branch(Tmp_Nodenum);
                                    if Is_Debug_Mode then
                                       Ada.Text_Io.Put_Line("=== arg_join_stack : " & Nodenum_T'Image(Tmp_Nodenum));
                                    end if;
                                    Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                                 end if;
                                 Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                                 Caller_Nodenum := Tmp_Nodenum;
                                 if Is_Debug_Mode then
                                    Ada.Text_Io.Put_Line("caller nodenum: " & Nodenum_T'Image(Caller_Nodenum));
                                 end if;

                                 if Arg_Item.Formal_In = 0 then
                                    Nodenum_Stack.Push(Arg_Item.Actual_In,
                                                       Tmp_Nodenum);
                                 else
                                    Add_Arg_Branch(Tmp_Nodenum,
                                                   Arg_Item.Formal_In);
                                 end if;
                              end if;
                           end if;
                        end if;
                     end if;
                  end;
               <<End_Of_Parameter_Association>>
            when others =>
               null;
         end Case;
      when Asis.A_Definition =>
         Put_Line("check definition");
         Arg_Definition_Kind := Asis.Elements.Definition_Kind (Element);

         case Arg_Definition_Kind is

         when Asis.A_Discrete_Range =>
                  if Asis.Elements.Discrete_Range_Kind(Element) = A_Discrete_Simple_Expression_Range then
                     Disc_Range_Flag := True; -- ASIS bug
                  end if;
         when Asis.An_Aspect_Specification =>
            declare
               Tmp_Mark: Asis.Element;
               Tmp_Nodelink : Node_Item_Link;
               begin
                  Put_Line("290");
                  Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                  Remove_Branch (Tmp_NodeNum, Tmp_NodeNum + 1);
                  Tmp_Mark := Asis.Definitions.Aspect_Mark(Element);
```

```
               while asis.Elements.Attribute_Kind(Tmp_Mark) /= Not_An_Attribute loop
                  Tmp_Mark := Asis.Expressions.Prefix(Tmp_Mark);
               end loop;
               if asis.Expressions.Name_Image(Tmp_Mark) = Asis.Set_Get.To_Program_Text("Pre") then
               Put_Line("294");
               Tmp_Nodelink := Search_Node(Tmp_NodeNum);
               Tmp_Nodelink.Is_Pre := True;
               Current_Unit.Subprograms.Precon_Nodenum := Tmp_NodeNum;
               end if;
               if asis.Expressions.Name_Image(Tmp_Mark) = asis.Set_Get.To_Program_Text("Post") then
               Put_Line("296");
               Tmp_Nodelink := Search_Node(Tmp_NodeNum);
               Tmp_Nodelink.Is_Post := True;
               Current_Unit.Subprograms.Postcon_Nodenum := Tmp_NodeNum;
               end if;
            end;
      when others =>

         null;
      end case;



   when Asis.An_Expression =>
      Put_Line("check expression");
      Arg_Expression_Kind := Asis.Elements.Expression_Kind(Element);
      Case Arg_Expression_Kind is
      when Asis.An_Identifier =>
        -- if Asis.Text.Element_Image(Element) = Asis.Set_Get.To_Program_Text("Pre") then
           -- Put_Line("311");
           -- goto End_of_Identifier_Process;
         --end if;
        -- if Asis.Text.(Element) = Asis.Set_Get.To_Program_Text("Post") then
            --goto End_of_Identifier_Process;
         --end if;
              -- Put_Line("317");
               Put_Line(Asis.Text.Element_Image(Element));
               --  assignment_statement
               --
              -- Put_Line("321");
               if Get_Info(Element) /= Assigned_Element_Info then
--                Push(Vstring_Stack, To_V(Asis.Expressions.Name_Image(Elem)));
                  declare                        Tmp_Var : Varnum_T := Get_Varnum(Element);
                     --
                     Parent_Exp : Asis.Element;
                  begin
                     if Is_Debug_Mode then
                        Ada.Text_Io.Put_Line("get_varnum: " & Varnum_T'Image(Tmp_Var));
                     end if;
                     if Tmp_Var /= 0 then
                        --
                        Parent_Exp := Element;
                        if (not Disc_Range_Flag) and then Asis.Elements.Element_Kind(Asis.Elements.Enclosing_Element(Parent_Exp)) = An_Expression then
                           declare
                              Iter : Asis.Expression := Asis.Elements.Enclosing_Element(Parent_Exp);
                              Target : Asis.Expression;
                              Norm_Stack : Vstring_Stacks.Stack := null;
                              Norm_Form : V_String := Null_Str;
                              Isnot_First : Boolean := False;
                              Is_Norm : Boolean := False;
                           begin
--                            Push(Norm_Stack, To_V(Asis.Expressions.Name_Image(Elem)));

                              Iter_Loop:
                              while Asis.Elements.Element_Kind(Iter) = An_Expression loop
                                 if Is_Debug_Mode then
                                    Put_Line("iter: " & Asis.Text.Element_Image(Iter));
                                 end if;
                                 case Asis.Elements.Expression_Kind(Iter) is
                                    when A_Selected_Component =>
                                       Target := Asis.Expressions.Selector(Iter);
                                       if Is_Debug_Mode then
                                          Put_Line(Asis.Text.Element_Image(Target));
                                       end if;
--                                     if Target = Elem then
                                       if Asis.Elements.Is_Equal
                                         (Target,Element) then
                                          exit Iter_Loop;
--                                     elsif Asis.Expressions.Corresponding_Expression_Type(Target) /= Asis.Expressions.Corresponding_Expression_Type(Elem) then
                                       elsif not Is_Type_Equal(Target, Element) then
                                          if Is_Debug_Mode then
                                             Put_Line("pushed " & Asis.Expressions.Name_Image(Target));
                                          end if;
                                          Push(Norm_Stack, To_V(Asis.Expressions.Name_Image(Target)));
                                       else

                                          Is_Norm := True;
                                       end if;

                                       Parent_Exp := Iter;

                                       Iter := Asis.Elements.Enclosing_Element(Iter);

                                    when An_Explicit_Dereference =>
                                       exit Iter_Loop;
                                    when An_Identifier =>

                                       exit Iter_Loop;
                                    when An_Indexed_Component =>
--                                     if Asis.Expressions.Prefix(Iter) = Parent_Exp then
                                       if Asis.Elements.Is_Equal
                                         (Asis.Expressions.Prefix(Iter),
                                          Parent_Exp) then
                                          Parent_Exp := Iter;
                                       end if;
                                       exit Iter_Loop;
                                    when others =>
                                       exit Iter_Loop;
                                 end case;
```

```
                                 end loop Iter_Loop;
                                 if Is_Debug_Mode then
                                    Put_Line("iter loop exited.");
                                 end if;

                                 --  prefix/selector
                                 if not Isempty(Norm_Stack) then
                                    while not Isempty(Norm_Stack) loop
                                       if Isnot_First then
                                          Norm_Form := Top(Norm_Stack) & "." & Norm_Form;
                                       else
                                          Isnot_First := True;
                                          Norm_Form := Top(Norm_Stack);
                                       end if;
                                       Remove(Norm_Stack);
                                    end loop;
                                    Norm_Form := To_V(Asis.Expressions.Name_Image(Element)) & "." & Norm_Form;
                                    --  element
                                    -- Assign
                                    if Get_Info(Parent_Exp) /= Assigned_Element_Info then
--                                     Put_Line("case norm:use");
                                       Push(Var_Item_Stack, (Tmp_Var, Is_Norm, Norm_Form, null));
                                    else
--                                     Put_Line("case norm:assign");
                                       Push(A_Var_Item_Stack, (Tmp_Var, Is_Norm, Norm_Form, null));
                                    end if;
                                 else
                                    if Get_Info(Parent_Exp) /= Assigned_Element_Info then
--                                     Put_Line("case :use");
                                       Push(Var_Stack, Tmp_Var);
                                    else
--                                     Put_Line("case :assign");
                                       Push(A_Var_Stack, Tmp_Var);
                                    end if;
                                 end if;
                              end;
                           else
                              Push(Var_Stack, Tmp_Var);
                           end if;
                        end if;
                     end;

--                   use_variable(To_V(Asis.Expressions.Name_Image(Elem)), Num_Of_Nodes);
                  end if;
                  <<End_of_Identifier_Process>>
            when Asis.A_Function_Call =>
                  if Is_Debug_Mode then
                     Put_line("function call");
                     Put_Line(Asis.Text.Element_Image(Element));
                  end if;
                  declare
                     Called_Entity : Asis.Declaration;
                     Subprog : Subprogram_Item_Link;
                     New_Call_Branch : Call_Branch_Item_Link := null;
                  begin
                     begin
                        Arg_Expression_Kind := asis.Elements.Expression_Kind(asis.Expressions.Prefix(Element));
                        case Arg_Expression_Kind is
                        when asis.An_Operator_Symbol | asis.An_Attribute_Reference =>
                        begin
                          --Put_Line("126");
                          Called_Entity := asis.Expressions.Prefix(Element);
                        end;
                        when asis.A_Selected_Component =>
                        begin
                           --Put_Line("127");
                           Called_Entity := Asis.Expressions.Corresponding_Name_Declaration(asis.Expressions.Selector(asis.Expressions.Prefix(Element)));
                        end;
                        when others =>
                        begin
                           --Put_Line("123");
                           --  Put_Line(asis.Text.Element_Image(asis.Elements.Enclosing_Element(Asis.Expressions.Corresponding_Name_Definition(asis.Expressions.Prefix(Element)))));
                           --Put_Line("233");
                        -- Asis.Expressions.Prefix(Elem);
                        -- if asis.Elements.Attribute_Kind(element) /= Not_An_Attribute then
                         -- Put_Line(Asis.Text.Element_Image(Asis.Expressions.Attribute_Designator_Identifier(Element)));
                        -- end if;
                        -- Put_Line(Asis.Text.Element_Image(Asis.Expressions.Corresponding_Name_Definition(asis.Expressions.Prefix(Element))));
                           --Put_Line("127");
                           --Called_Entity := asis.Elements.Enclosing_Element(Asis.Expressions.Corresponding_Name_Declaration(asis.Expressions.Prefix(Element)));
                           Called_Entity := Asis.Expressions.Corresponding_Name_Declaration(asis.Expressions.Prefix(Element));
                        end;
                        end case;
                        --Called_Entity := Asis.Elements.Enclosing_Element(Asis.Expressions.Corresponding_Name_Definition(Asis.Expressions.Selector(Asis.Expressions.Prefix(Element))));
                        --Put_Line("124");
                        --Put_Line(gela_ids.Debug_Image(gela_ids.Create_Id(Called_Entity)));
                        -- ASIS bug
                     exception
                        when others =>
                           --Put_Line("125");
                           Put_Line(Asis.Text.Element_Image(Asis.Expressions.Prefix(Element)));
                           --Put_Line("666");
                           Called_Entity := Asis.Expressions.Corresponding_Called_Function(Element);
                     end;
                     Subprog := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_entity));
--                   if Called_Entity /= Nil_Element and then Asis.Elements.Enclosing_Compilation_Unit(Called_Entity) = The_Unit then
                     if Subprog /= null then
--                        Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                        Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                       --Put_Line(asis.Text.Element_Image(Called_Entity));
                       -- Put_Line(to_s(Current_Routine.Name));
                       -- Put_Line(to_s(Subprog.Name));
                        --Put_Line(to_s(Current_Routine.Subprogram.Name));
                        Set_Call_Tree(Current_Routine.Subprogram, Subprog, Num_Of_Nodes);
                        if Subprog.Callee = 0 then
--                           Put_Line("501");
                           if Subprog.Precon_Nodenum /= 0 then
                              Put_Line("503");
```

```
                              New_Call_Branch := Add_Call_Branch(Caller_Nodenum, Subprog.Precon_Nodenum);
                              Put_Line("513");
                              Add_Branch(Subprog.Precon_Nodenum, 0);
                              Put_Line("514");
                           else
                              New_Call_Branch :=
                              Add_Call_Branch(Caller_Nodenum, 0);
                           end if;
                           Call_Branch_Stack.Push(Subprog.Caller,
                                                  New_Call_Branch);
                        else
                           Put_Line("513");
                           if Subprog.Precon_Nodenum /= 0 then
                              Put_Line("515");
                              New_Call_Branch := Add_Call_Branch(Caller_Nodenum, Subprog.Precon_Nodenum);
                              Add_Branch(Subprog.Precon_Nodenum, Subprog.callee);
                           else
                              New_Call_Branch :=
                              Add_Call_Branch(Caller_Nodenum,
                                              Subprog.Callee);
                           end if;
                        end if;
                        if Subprog.Postcon_Nodenum /= 0 then
                          Add_Branch(Subprog.Postcon_Nodenum, Caller_Nodenum);
                        end if;
                        if Is_Debug_Mode then
                           Ada.Text_Io.Put_Line("caller nodenum: " & Nodenum_T'Image(Caller_Nodenum));
                        end if;
                        Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Subprog.Args, A_Function_Call, True, False, Caller_Nodenum, null, New_Call_Branch));
                     else
                        Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, A_Function_Call, False, False, 0, null, null));
                     end if;

                     end;
--                   end if;
         when Asis.An_Allocation_From_Subtype =>          ----????
            Put_Line("557");
            --Put_Line(Asis.Text.Element_Image(Asis.Expressions.Selector(Asis.Definitions.Subtype_Mark(Asis.Expressions.Allocator_Subtype_Indication(Element)))));
                  declare
                     Task_Decl : Asis.Declaration;
                     Task_Type : Task_Type_Info_Link;
                  begin
                     if Asis.Elements.Expression_Kind(Asis.Definitions.Subtype_Mark(Asis.Expressions.Allocator_Subtype_Indication(Element))) = A_Selected_Component then
                        Task_Decl := Asis.Expressions.Corresponding_Name_Declaration
                        (Asis.Expressions.Selector(Asis.Definitions.Subtype_Mark(Asis.Expressions.Allocator_Subtype_Indication(Element))));
                     else
                        Task_Decl := Asis.Expressions.Corresponding_Name_Declaration(Asis.Definitions.Subtype_Mark(Asis.Expressions.Allocator_Subtype_Indication(Element)));
                     end if;
                     if Asis.Elements.Declaration_Kind(Task_Decl) = A_Task_Type_Declaration then
                        Task_Type := Task_Type_Id_List.Search_Node(Task_Type_Decl_List, Gela_Ids.Create_Id(Task_Decl));
                        if Task_Type /= null  then
                           Add_P_Branch_From_Allocate(Task_Type);
                        end if;
                     end if;
                  end;
         when others =>
            null;
         end case;



      when Asis.A_Path =>
         Put_Line("check path");
         Put_Line("544");
         Arg_Path_Kind := Asis.Elements.Path_Kind (Element);
         Case Arg_Path_Kind is
            when Asis.Not_A_Path =>
               Ada.wide_Text_IO.Put("Not_A_Path : ");
               Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.First_Line_Number(Element)));
               Ada.wide_Text_IO.Put("-");
               Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.Last_Line_Number(Element)));
               Ada.wide_Text_IO.Put(":");
               Ada.Wide_Text_IO.Put(Asis.Text.Element_Image(Element));
               Ada.Wide_Text_IO.New_Line;

            when Asis.An_If_Path =>
                  Set_Info(Asis.Statements.Condition_Expression(Element), Get_Info(Asis.Elements.Enclosing_Element(Element)));
            when Asis.An_Elsif_Path =>
                  Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                  Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                  Set_Info(Asis.Statements.Condition_Expression(Element),
                           (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
            when Asis.A_Select_Path =>
                  declare
                     Guard_Item : Asis.Element := Asis.Statements.Guard(Element);
                  begin
--                   if Guard_Item /= Nil_Element then
                     if not Asis.Elements.Is_Nil(Guard_Item) then
                        Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Guard_item)));
                        Set_Info(Guard_Item, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum +1));
                     end if;
                  end;
            when Asis.An_Or_Path =>
                  declare
                     Guard_Item : Asis.Element := Asis.Statements.Guard(Element);
                  begin
--                   if Guard_Item /= Nil_Element then
                     if not Asis.Elements.Is_Nil(Guard_Item) then
                        Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Guard_item)));
                        Set_Info(Guard_Item, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum +1));
                     end if;
                  end;
            when others =>
               null;
         end case;

      when Asis.An_Exception_Handler =>
            if Is_Exception_Node then
               if Current_Unit.Exception_Point = 0 then
```

```
                  Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                  Remove_Branch(Num_Of_Nodes, Num_Of_Nodes + 1);
                  Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                  Remove_Branch(Num_Of_Nodes, Num_Of_Nodes + 1);
                  Current_Unit.Exception_Point := Tmp_Nodenum;
               else
                  Remove_Branch(Current_Unit.Exception_Point -1, Num_Of_Nodes +1);
               end if;
               Add_Branch(Current_Unit.Exception_Point, Num_Of_Nodes + 1);
            else
               Control := Abandon_Children;
            end if;
         when Asis.A_Statement =>
            if Is_Debug_Mode then
               Put_Line(Asis.Elements.Debug_Image(Element));
               Put_Line(Asis.Text.Element_Image(Element));
            end if;

            if Asis.Elements.Statement_Kind(Element) /= A_Terminate_Alternative_Statement then

               Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
               Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
               Caller_Nodenum := Tmp_Nodenum;
               if Is_Debug_Mode then
                  Ada.Text_Io.Put_Line("caller nodenum: " & Nodenum_T'Image(Caller_Nodenum));
               end if;
               Assign_Stacks;

--             Put("(");
--             Node_Io.Put(tmp_Nodenum);
--             Put(", ");
--             Node_Io.Put(tmp_Nodenum);
--             Put(", ");
--             Node_Io.Put(Tmp_Nodenum+1);
--             Put_line(")");
            end if;

            Clear(Vstring_Stack);
            Clear(Var_Stack);
            Clear(A_Var_Stack);

            Current_Statement := Tmp_Nodenum;
            declare
               package Statement_IO is new Enumeration_IO(Asis.Statement_Kinds);
               package Declaration_IO is new Enumeration_IO(Asis.Declaration_Kinds);
               package Expression_IO is new Enumeration_IO(Asis.Expression_Kinds);
               use Statement_IO;
               use Declaration_IO;
               use Expression_IO;
            begin
               case Asis.Elements.Statement_Kind ( Element ) is
                  when A_Null_Statement =>
                     null;
                  when An_Assignment_Statement =>
                     --           Line_IO.Put(Asis.Text.First_Line_Number(Elem));
                     --           Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                     --           Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
--                   Put(Asis.Elements.Expression_Kind(Asis.Expressions.Prefix(Asis.Statements.Assignment_Variable_Name(Elem))));
--                   New_Line;
--                   if Asis.Elements.Expression_Kind(Asis.Statements.Assignment_Variable_Name(Elem)) = A_Selected_Component then
--                      Put_Line(Asis.Text.Element_Image(Asis.Expressions.Prefix(Asis.Statements.Assignment_Variable_Name(Elem))));
--                      Put(Asis.Elements.Expression_Kind(Asis.Expressions.Prefix(Asis.Statements.Assignment_Variable_Name(Elem))));
--                      New_Line;
--                   end if;
--                   Put_Line("test name_definition:");
--                   Put_Line(Asis.Text.Element_Image(Asis.Expressions.Corresponding_Name_Definition[](Asis.Statements.Assignment_Variable_Name(Elem))));
--                   Put_Line("test name_declaration:");
--                   Put_Line(Asis.Text.Element_Image(Expressions.Corresponding_Name_Declaration(Asis.Expressions.Selector(Asis.Statements.Assignment_Variable_Name(Elem)))));
                     declare
                        Assign_Var : Asis.Expression := Asis.Statements.Assignment_Variable_Name(Element);
                     begin
                        Set_Info(Assign_Var, Assigned_Element_Info);
                        if Asis.Elements.Expression_Kind(Assign_Var) = A_Selected_Component then
                           Assign_Var := Asis.Expressions.Selector(Assign_Var);
                        end if;
                        declare
                           Tmp_Var : Varnum_T := Get_Varnum(Assign_Var);
                        begin
                           if Is_Debug_Mode then
                              Ada.Text_Io.Put_Line("get_varnum: " & Varnum_T'Image(Tmp_Var));
                           end if;
                           if Tmp_Var /= 0 then
                              Assign_Variable(Get_Varnum(Assign_Var), Tmp_Nodenum);
                           end if;
                        end;
--                      Put_Line(Asis.Text.Element_Image(Asis.Expressions.Corresponding_Expression_Type(Assign_Var)));
--                      Assign_Variable(To_V(Asis.Expressions.Name_Image(Asis.Statements.Assignment_Variable_Name(Elem))), Tmp_Nodenum);

--                      Set_Info(Asis.Statements.Assignment_Variable_Name(Elem), Assigned_Element_Info);
                     end;
                  when An_If_Statement =>
                     null;
                     --           Push(Ifnum_Stack, Tmp_Nodenum);
                     declare
--                      Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
--                   begin
--                      for I in Plist'Range loop
--                         declare
--                            Pkind : Asis.Element_Kinds := Asis.Elements.Path_Kind(Plist(I));
--                            Cond : Asis.Expression;
--                         begin
--                         if Pkind = An_If_Path or Pkind = An_Elsif_Path then
--                            Cond := Condition_Expression(pkind

                  when A_Case_Statement =>
                     null;

                  when A_Loop_Statement =>
                     null;
```

```
                  when A_While_Loop_Statement =>
                     Set_Info(Asis.Statements.While_Condition(Element), Get_Info(Element));

                  when A_For_Loop_Statement =>
                     declare
                        Param : Asis.Declaration := Asis.Statements.For_Loop_Parameter_Specification(Element);
                        Type_Ident : V_String := To_V(String'("For_Loop_Param"));
                        Tmp_Varnum : Varnum_T;
                     begin
--                      Type_Ident := Get_Type_Ident(Param);
                        if Is_Debug_Mode then
                           Put_Line("for_loop");
                           Put_Line(Asis.Text.Element_Image(Param));
                        end if;
                        Tmp_Varnum := Enter_Variable(Asis.Declarations.Names(Param)(1), Type_Ident, False);
                        Assign_Variable(Tmp_Varnum, Tmp_Nodenum);
                        Use_Variable(Tmp_Varnum, Tmp_Nodenum);
                     end;

                  when A_Block_Statement =>
                     if Asis.Statements.Is_Declare_Block( Element ) then
                        declare
                           Block_Name : Asis.Defining_Name := Asis.Statements.Statement_Identifier(Element);
                           Dlist : Asis.Declarative_Item_List := Asis.Statements.Block_Declarative_Items(Element);
                        begin
--                         if Block_Name = Nil_Element then
                           if Asis.Elements.Is_Nil(Block_Name) then
                              Enter_Unit(To_V(Unique_Identifier), Not_A_Declaration);
                           else
                              Enter_Unit(To_V(Asis.Declarations.Defining_Name_Image(Block_Name)), Not_A_Declaration);
                           end if;
                           First_Stmt;
                           Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                           Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                           Dlist_Add(Dlist);
                        end;
                        Set_Fork_Point;
                     end if;

                  when An_Exit_Statement =>
--                   declare
--                      Loop_Name : Asis.Expression := Asis.Statements.Exit_Loop_Name(Elem);
--                      Iter : Asis.Element;
--                   begin
--                      if Loop_Name = Nil_Element then
                           Push_Exit_Node(Asis.Statements.Corresponding_Loop_Exited(Element), Tmp_Nodenum);
--                      else
--                         if Asis.Elements.Expression_Kind(Loop_Name) =
--                           A_Selected_Component then
--                            Loop_Name := Asis.Expressions.Selector(Loop_Name);
--                         end if;
--                         Iter := Asis.Elements.Enclosing_Element(Elem);
--                         while Iter /= Nil_Element loop
--                            if Asis.Elements.Element_Kind(Iter) = A_Statement and then
--                              (Asis.Elements.Statement_Kind(Iter) = A_Loop_Statement or Asis.Elements.Statement_Kind(Iter) = A_For_Loop_Statement
--                               or Asis.Elements.Statement_Kind(Iter) = A_While_Loop_Statement or Asis.Elements.Statement_Kind(Iter) = A_Block_Statement) and then
--                              Equal_Insensitive(To_V(Asis.Expressions.Name_Image(Loop_Name)),
To_V(Asis.Declarations.Defining_Name_Image(Asis.Statements.Statement_Identifier(Iter)))) then
--                                 Push_Exit_Node(Iter, Tmp_Nodenum);
--                               end if;
--                               Iter := Asis.Elements.Enclosing_Element(Iter);
--                            end loop;
--                      end if;
--                   end;

                     --           Put_Line("done(corresponding_loop_exited)");
--                   if Asis.Statements.Exit_Condition(Elem) = Nil_Element then
                     if Asis.Elements.Is_Nil
                       (Asis.Statements.Exit_Condition(Element)) then
                        Remove_Branch(Tmp_Nodenum, Tmp_Nodenum+1);
                        Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, 0));
                     end if;

                  when A_Goto_Statement =>
                     null;

                  when A_Procedure_Call_Statement =>
                  --          Line_IO.Put(Asis.Text.First_Line_Number(Elem));e
                  Put_Line("804");
                  if Asis.Elements.Is_Nil(Asis.Statements.Corresponding_Called_Entity(Element)) then
                     Put_Line("dispatching call or access to procedure, skip process");
                     Tmp_NodeNum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                     Set_Info(Element, (Tmp_NodeNum, Tmp_NodeNum, Tmp_NodeNum+1));
                     Skip_Process := True;
                     goto End_Of_Procedure_Call;
                  end if;
                     declare
                        Called_Entity : Asis.Declaration;
                        Subprog : Subprogram_Item_Link;
                     begin
                        Called_Entity := Asis.Statements.Corresponding_Called_Entity(Element);
                        if Asis.Statements.Is_Dispatching_Call(Element) then
                        Put_Line("call on dispatching");
                        end if;
                        Put_Line("776");
                        Put_Line(Asis.Text.Element_Image(Element));
                        Subprog := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
                        Put_Line("778");
--                      if Called_Entity /= Nil_Element
--                        and then Asis.Elements.Enclosing_Compilation_Unit(Called_Entity) = The_Unit
--                        and then Asis.Elements.Declaration_Kind(Called_Entity) /= Not_A_Declaration then
                        if Subprog /= null then
                           Put_Line("780");
                           if Is_Debug_Mode then
                              Put("procedure call : ");
                              Put_Line(Asis.Text.Element_Image(Called_Entity));
                           end if;
                           Set_Call_Tree(Current_Routine.Subprogram, Subprog, Tmp_Nodenum);
```

```
                  declare
--                   Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
                     New_Call_Branch : Call_Branch_Item_Link;
                  begin
                     if Subprog.Callee = 0 then
                        New_Call_Branch :=
                          Add_Call_Branch(Tmp_Nodenum, 0);
                        Call_Branch_Stack.Push(Subprog.Caller,
                                               New_Call_Branch);
                     else
                        New_Call_Branch := Add_Call_Branch
                          (Tmp_Nodenum, Subprog.Callee);
                     end if;

                     Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Subprog.Args, A_Procedure_Call, True, True, Tmp_Nodenum, null, New_Call_Branch));
                  end;
               else
                  Put_Line("803");
--                if Called_Entity /= Nil_Element and then Asis.Elements.Declaration_Kind(Called_Entity) /= A_Procedure_Instantiation then
                  if not (Asis.Elements.Is_Nil(Called_Entity)) and then Asis.Elements.Declaration_Kind(Called_Entity) /= A_Procedure_Instantiation then
                     Clear(Args_Stack);
                     Make_Arg_From_Proc(Called_Entity);
                     declare
                        Base_Args, New_Args : Args_Item_Link := null;
                     begin
                        while not Isempty(Args_Stack) loop
                           New_Args := new Args_Item;
                           New_Args.Name := Top(Args_Stack).Name;
                           New_Args.Mode := Top(Args_Stack).Mode;
                           New_Args.Position := Top(Args_Stack).Position;
                           New_Args.Next := Base_Args;
                           Base_Args := New_Args;
                           Remove(Args_Stack);
                        end loop;
                        Subprog := new Subprogram_Item;
                        Subprog.Args := Base_Args;
                        Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Base_Args, A_Procedure_Call, False, False, 0, null, null));
                        Clear(Args_Stack);
                     end;
                  else
                     Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, A_Procedure_Call, False, False, 0, null, null));
                  end if;
               end if;
            exception
               when others =>
                  Put_Line("exception raised.");
                  Put_Line(Asis.Elements.Debug_Image(Element));
                  Put(Asis.Elements.Declaration_Kind ( Called_Entity ));
                  New_Line;
            end;
         <<End_Of_Procedure_Call>>
         when An_Entry_Call_Statement =>
            if Is_Debug_Mode then
               Put_Line(Asis.Text.Element_Image(Asis.Statements.Corresponding_Called_Entity(Element)));
            end if;
            declare
               Entry_Decl : Entry_Item_Link := Entry_Decl_Id_List.Search_Node(Entry_Decl_List, Gela_Ids.Create_Id(Asis.Statements.Corresponding_Called_Entity(Element)));
            begin
               if Entry_Decl /= null then -- ASIS bug
                  if Entry_Decl.Task_Link.Decl_Kind /= A_Protected_Body_Declaration then
                     Enter_Entrycall(Entry_Decl, Tmp_Nodenum);
                     Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, Entry_Decl.Args, An_Entry_Call, False, False, 0, null, null));
                  else
                     declare
                        Called_Entity : Asis.Declaration := Asis.Statements.Corresponding_Called_Entity(Element);
                        Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
                        New_Call_Branch : Call_Branch_Item_Link
                          := null;
                     begin
                        if Subprog /= null then
                           if Is_Debug_Mode then
                              Put("entry body call : ");
                              Put_Line(Asis.Text.Element_Image(Called_Entity));
                           end if;
                           Set_Call_Tree(Current_Routine.Subprogram, Subprog, Tmp_Nodenum);

                           if Subprog.Callee = 0 then
                              New_Call_Branch :=
                                Add_Call_Branch(Tmp_Nodenum, 0);
                              Call_Branch_Stack.Push(Subprog.Caller,
                                                     New_Call_Branch);
                           else
                              New_Call_Branch := Add_Call_Branch
                                (Tmp_Nodenum, Subprog.Callee);
                           end if;

                           Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Subprog.Args, A_Protected_Entry_Call, True, True, Tmp_Nodenum, null, New_Call_Branch));
                        else
--                         if Called_Entity /= Nil_Element then
                           if not (Asis.Elements.Is_Nil(Called_Entity)) then
                              Clear(Args_Stack);
                              Make_Arg_From_Proc(Called_Entity);
                              declare
                                 Base_Args, New_Args : Args_Item_Link := null;
                              begin
                                 while not Isempty(Args_Stack) loop
                                    New_Args := new Args_Item;
                                    New_Args.Name := Top(Args_Stack).Name;
                                    New_Args.Mode := Top(Args_Stack).Mode;
                                    New_Args.Position := Top(Args_Stack).Position;
                                    New_Args.Next := Base_Args;
                                    Base_Args := New_Args;
                                    Remove(Args_Stack);
                                 end loop;
                                 Subprog := new Subprogram_Item;
                                 Subprog.Args := Base_Args;
                                 Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Base_Args, A_Protected_Entry_Call, False, False, 0, null, null));
                                 Clear(Args_Stack);
                              end;
```

```
                        else
                           Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, An_Entry_Call, False, False, 0, null, null));
                        end if;
                     end if;
                  exception
                     when others =>
                        Put_Line("exception raised.");
                        Put_Line(Asis.Elements.Debug_Image(Element));
                        Put(Asis.Elements.Declaration_Kind ( Called_Entity ));
                        New_Line;
                  end;

               end if;
            else
               declare
                  Called_Entity : Asis.Declaration := Asis.Statements.Corresponding_Called_Entity(Element);
                  Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
               begin
                  if Subprog /= null then
                     if Is_Debug_Mode then
                        Put("procedure call : ");
                        Put_Line(Asis.Text.Element_Image(Called_Entity));
                     end if;

                     Set_Call_Tree(Current_Routine.Subprogram, Subprog, Tmp_Nodenum);
                     declare
                        New_Call_Branch : Call_Branch_Item_Link;
                     begin
                        if Subprog.Callee = 0 then
                           New_Call_Branch :=
                             Add_Call_Branch(Tmp_Nodenum, 0);
                           Call_Branch_Stack.Push(Subprog.Caller,
                                                  New_Call_Branch);
                        else
                           New_Call_Branch :=
                             Add_Call_Branch(Tmp_Nodenum, Subprog.Callee);
                        end if;
                        Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Subprog.Args, A_Procedure_Call, True, True, Tmp_Nodenum, null, New_Call_Branch));
                     end;

                  else
                       Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, A_Procedure_Call, False, False, 0, null, null));
                  end if;

               exception
                  when others =>
                     Put_Line("exception raised.");
                     Put_Line(Asis.Elements.Debug_Image(Element));
                     Put(Asis.Elements.Declaration_Kind ( Called_Entity ));
                     New_Line;
               end;
            end if;
         end;

      when A_Return_Statement =>
         Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
         Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
         if Current_Unit.Decl_Kind = A_Function_Body_Declaration then
            --Put_Line(to_s(Current_Unit.Name));
            Nodenum_Stack.Push(Current_Unit.Subprogram.Return_Callee, Tmp_nodenum);--add parent
            if not Nodenum_Stack.Isempty(Current_Unit.Subprogram.Return_Caller) then--add parent
               declare
                  Iterator : Nodenum_Stack.Stack := Current_Unit.Subprogram.Return_Caller;--add parent
                  use Nodenum_Stack;
               begin
                  while Iterator /= null loop

                     Add_Arg_Branch(Tmp_Nodenum, Nodenum_Stack.Top(Iterator));

                     Iterator := Iterator.Next;
                  end loop;
               end;
            end if;
            if not Is_Param_Node_Approach then
               Add_Arg_Branch(Tmp_Nodenum, Tmp_Nodenum);
            end if;
         end if;
--         if Current_Unit.Subprogram.Postcon_Nodenum /= 0 then
--            Remove_Branch(Tmp_NodeNum-1, Tmp_NodeNum);
--            Add_Branch(Tmp_NodeNum-1, Current_Unit.Subprogram.Postcon_Nodenum);
--            Add_Branch(Current_Unit.Subprogram.Postcon_Nodenum, Tmp_NodeNum);
--         end if;

      when An_Accept_Statement =>
         Enter_Accept(Gela_Ids.Create_Id(Asis.Statements.Corresponding_Entry(Element)), Tmp_Nodenum);
         declare
            Plist : Asis.Parameter_Specification_List := Asis.Statements.Accept_Parameters(Element);
         begin
            if Plist /= Nil_Element_List then
               for I in Plist'Range loop
                  declare
                     DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                  begin
                     for J in Dname_List'Range loop

                        case Asis.Elements.Mode_Kind(Plist(I)) is
                           when A_Default_In_Mode | An_In_Mode
                             | An_In_Out_Mode =>
                                Assign_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Tmp_Nodenum);
                           when others => null;
                        end case;
                     end loop;
                  end;
               end loop;
            end if;
         end;
         Push(Accepthead_Stack,Tmp_Nodenum);
      when A_Requeue_Statement |
        A_Requeue_Statement_With_Abort =>
```

```
                  declare
                     Entry_Decl : Entry_Item_Link := Entry_Decl_Id_List.Search_Node(
                     Entry_Decl_List, Gela_Ids.Create_Id(Asis.Expressions.Corresponding_Name_Declaration(
                     Asis.Statements.Requeue_Entry_Name(Element))));
                     Encl_Entry : Asis.Element := Asis.Elements.Enclosing_Element(Element);
                     Parent_Decl : Asis.Declaration;
                     Keep_Unit : Unit_Item_Link := Current_Unit;
                     Keep_Nodenum : Nodenum_T := Tmp_Nodenum;
                  begin
                     if Is_Debug_Mode then
                        Put_Line("requeue statement");
                     end if;

                     if Asis.Elements.Statement_Kind(Element) =
                       A_Requeue_Statement_With_Abort then
                        Set_Nondet_Branch(Tmp_Nodenum);
                        Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                        Set_Info(Element, (Keep_Nodenum, Tmp_Nodenum, Tmp_Nodenum +1));
                     end if;
                     Iter_Encl:
                     loop
                        case Asis.Elements.Element_Kind(Encl_Entry) is
                           when A_Declaration =>
                              if Asis.Elements.Declaration_Kind(Encl_Entry)
                                = An_Entry_Body_Declaration then
                                 Parent_Decl := Encl_Entry;
--                               Plist := Asis.Declarations.Parameter_Profile(Encl_Entry);
                                 exit Iter_Encl;
                              end if;

                           when A_Statement =>
                              if Asis.Elements.Statement_Kind(Encl_Entry)
                                = An_Accept_Statement then
                                 Parent_Decl := Asis.Statements.Corresponding_Entry(Encl_Entry);
--                               Plist := Asis.Statements.Accept_Parameters(Encl_Entry);
                                 exit Iter_Encl;
                              end if;
                           when others =>
                              null;
                        end case;
                        Encl_Entry := Asis.Elements.Enclosing_Element(Encl_Entry);
                     end loop Iter_Encl;
                     if Is_Debug_Mode then
                        Put_Line(Asis.Text.Element_Image(Parent_Decl));
                     end if;
                     declare
                        Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Parent_Decl);
                        Parent_Decl_Spec : Asis.Declaration;
                     begin

                        if Entry_Decl = null or else Entry_Decl.Task_Link.Decl_Kind /= A_Protected_Body_Declaration then
                           if Plist /= Nil_Element_List then
                              for I in Plist'Range loop
                                 declare
                                    DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                                    Type_Ident : V_String := Get_Type_Ident(Plist(I));
                                    Decl_Is_Access : Boolean;
                                 begin
                                    Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                                    for J in Dname_List'Range loop
                                       case Asis.Elements.Mode_Kind(Plist(I)) is
                                          when A_Default_In_Mode | An_In_Mode =>
                                             Use_Variable(Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Dname_List(J))), Keep_Nodenum);
                                          when An_In_Out_Mode =>
                                             Use_Variable(Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Dname_List(J))), Keep_Nodenum);
                                             Assign_Variable(Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Dname_List(J))), Tmp_Nodenum);
                                          when others =>
                                             Assign_Variable(Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Dname_List(J))), Tmp_Nodenum);
                                       end case;
                                    end loop;
                                 end;
                              end loop;
                           end if;
                           if Entry_Decl /= null then
                              Enter_RequeueEntry(Entry_Decl, Keep_Nodenum,Tmp_Nodenum);
--                            Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, An_Entry_Call, False, False, 0, null, null));

                           end if;
                        else
                           declare
--                            Called_Entity : Asis.Declaration := Asis.Statements.Corresponding_Called_Entity(Elem);
--                            Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
                              Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Entry_Decl.Decl_Id);
                              Arg_Item : Args_Item_Link;
                              Mode_Kind : Asis.Mode_Kinds;
                              New_Call_Branch : Call_Branch_Item_Link := null;
                           begin
                              Current_Unit := Search_Unit_Item(Search_Node(Get_Info(Parent_Decl).Top).Belong_Unit);
                              Parent_Decl_Spec := Prot_Entry_Decl_List.Search_Node(Protected_Entry_List, Search_Entry_Decl(To_V(
                              Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Parent_Decl) (1)))));
                              Current_Unit := Keep_Unit;
                              Plist := Asis.Declarations.Parameter_Profile(Parent_Decl_Spec);
                              if Subprog /= null then
                                 Set_Call_Tree(Current_Routine.Subprogram, Subprog, Tmp_Nodenum);
                                 if Subprog.Callee = 0 then
                                    New_Call_Branch :=
                                      Add_Call_Branch(Tmp_Nodenum, 0);
                                    Call_Branch_Stack.Push(Subprog.Caller,
                                                           New_Call_Branch);
                                 else
                                    New_Call_Branch :=
                                      Add_Call_Branch(Tmp_Nodenum,
                                                      Subprog.Callee);
                                 end if;

                                 if Plist /= Nil_Element_List then
                                    if Is_Param_Node_Approach then
                                       Set_Arg_Branch(Tmp_Nodenum);
```

```
                                    end if;
                                    Arg_Item := Subprog.Args;
                                    for I in Plist'Range loop
                                       declare
                                          DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                                          Type_Ident : V_String := Get_Type_Ident(Plist(I));
                                          Decl_Is_Access : Boolean;
                                       begin
                                          Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                                          for J in Dname_List'Range loop
                                             Mode_Kind := Asis.Elements.Mode_Kind(Plist(I));
                                             if not Is_Param_Node_Approach then
                                                Push(Var_Stack, Get_Varnum(Dname_List(J)));
                                                case Mode_Kind is
                                                   when A_Default_In_Mode | An_In_Mode =>
                                                      Add_Actual_In(New_Call_Branch.Actual_In, Arg_Item.Position, False);
                                                   when An_In_Out_Mode =>
                                                      Add_Actual_In(New_Call_Branch.Actual_In, Arg_Item.Position, True);
                                                      Add_Actual_Out(New_Call_Branch.Actual_Out, Arg_Item.Position, False);
                                                   when An_Out_Mode =>
                                                      Add_Actual_Out(New_Call_Branch.Actual_Out, Arg_Item.Position, True);
                                                   when others =>
                                                      null;
                                                end case;
                                             else

                                                case Mode_Kind is
                                                   when A_Default_In_Mode
                                                     | An_In_Mode
                                                     | An_In_Out_Mode =>
                                                      Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                                                      Remove_Branch(Tmp_Nodenum-1, Tmp_Nodenum);
                                                      Add_Branch(Keep_Nodenum, Tmp_Nodenum);
                                                   if Mode_Kind = A_Default_In_Mode or Mode_Kind = An_In_Mode then
                                                      Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                                                      Set_Arg_Branch(Tmp_Nodenum);
                                                      if Is_Debug_Mode then
                                                         Ada.Text_Io.Put_Line("=== arg_join_stack : " & Nodenum_T'Image(Tmp_Nodenum));
                                                      end if;
                                                      Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                                                   end if;
                                                   Set_Info(Element, (Keep_Nodenum, Tmp_Nodenum, Tmp_Nodenum +1));
                                                   -- set_info
                                                   if Arg_Item.Formal_In = 0 then
                                                      Nodenum_Stack.Push(Arg_Item.Actual_In,
                                                                         Tmp_Nodenum);
                                                   else
                                                      Add_Arg_Branch(Tmp_Nodenum,
                                                                     Arg_Item.Formal_In);
                                                   end if;
                                                   when others =>
                                                      null;
                                                end case;
                                             end if;
                                             Arg_Item := Arg_Item.Next;
                                          end loop;
                                       end;
                                    end loop;
                                    if Is_Param_Node_Approach then
                                       Arg_Item := Subprog.Args;

                                       for I in Plist'Range loop
                                          declare
                                             DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                                             Type_Ident : V_String := Get_Type_Ident(Plist(I));
                                             Decl_Is_Access : Boolean;
                                          begin
                                             Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                                             for J in Dname_List'Range loop
                                                case Asis.Elements.Mode_Kind(Plist(I)) is
                                                   when An_In_Out_Mode
                                                     | An_Out_Mode =>
                                                      Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                                                      Set_Arg_Branch(Tmp_Nodenum);
                                                      Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                                                      Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                                                      if Arg_Item.Mode = An_Out_Mode then
                                                      Add_Branch(Keep_Nodenum, Tmp_Nodenum);
                                                      end if;
                                                      -- set_info
                                                      if Arg_Item.Formal_Out = 0 then
                                                         Nodenum_Stack.Push(Arg_Item.Actual_Out,
                                                                            Tmp_Nodenum);
                                                      else
                                                         Add_Arg_Branch(Tmp_Nodenum,
                                                                        Arg_Item.Formal_Out);
                                                      end if;
                                                   when others =>
                                                      null;
                                                end case;
                                                Arg_Item := Arg_Item.Next;
                                             end loop;
                                          end;
                                       end loop;
                                    end if;
                                 end if;

--                               else

                                 end if;
                              end;
                           end if;
                        end;

                        Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
                        if Asis.Elements.Element_Kind(Encl_Entry) = A_Statement then
                           Make_Labeled_Branch(Tmp_Nodenum, Label_T(Top(Accepthead_Stack)));
                        else
                           Make_Labeled_Branch(Tmp_Nodenum, Label_T(Top(Subprogramhead_Stack)));
```

```
                        end if;
                        if Asis.Elements.Statement_Kind(Element) =
                          A_Requeue_Statement_With_Abort then
                           if Asis.Elements.Element_Kind(Encl_Entry) = A_Statement then
                              Make_Labeled_Branch(Keep_Nodenum, Label_T(Top(Accepthead_Stack)));
                           else
                              Make_Labeled_Branch(Keep_Nodenum, Label_T(Top(Subprogramhead_Stack)));
                           end if;
                        end if;
                     end;


                  when A_Delay_Until_Statement =>
                     null;

                  when A_Delay_Relative_Statement =>
                     null;

                  when A_Terminate_Alternative_Statement =>
                     null;

                  when A_Selective_Accept_Statement =>
                     Set_Nondet_Branch(Tmp_Nodenum);

                  when A_Conditional_Entry_Call_Statement =>
                     Set_Nondet_Branch(Tmp_Nodenum);

                  when A_Timed_Entry_Call_Statement =>
                     Set_Nondet_Branch(Tmp_Nodenum);

                  when An_Asynchronous_Select_Statement =>
--                   Set_Nondet_Branch(Tmp_Nodenum);
                     null;
                  when An_Abort_Statement =>
                     null;

                  when A_Raise_Statement =>
                     null;

                  when A_Code_Statement =>
                     null;

                  when others =>
                     null;
               end case;
            exception
               when Asis.Exceptions.Asis_Inappropriate_Element =>
                  Put_Line("raised ASIS.EXCEPTIONS.ASIS_INAPPROPRIATE_ELEMENT");
                  Put_Line(Asis.Text.Element_Image(Element));
                  Put(Asis.Elements.Statement_Kind ( Element ));
                  New_Line;
            end;
         when Asis.A_Declaration =>
            Put_Line("check delclaration");
            Put_Line(Asis.Elements.Debug_Image(Element));
            --  We have to compute the total number of all the declarations, so:
            Metrics_Utilities.Total_Declarations :=
              Metrics_Utilities.Total_Declarations + 1;
            Arg_Declaration_Kind := Asis.Elements.Declaration_Kind (Element);
            --Ada.wide_Text_IO.Put("A_Declaration : ");
            --Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.First_Line_Number(Element)));
            --Ada.wide_Text_IO.Put("-");
            --Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.Last_Line_Number(Element)));
            --Ada.wide_Text_IO.Put(":");
            --Ada.Wide_Text_IO.New_Line;
            --Ada.Wide_Text_IO.Put(Asis.Text.Element_Image(Element));
            --Ada.Wide_Text_IO.New_Line;
            --Ada.Wide_Text_IO.Put(Asis.Elements.Debug_Image(Element));
            --Ada.Wide_Text_IO.New_Line;
            Case Arg_Declaration_Kind is
               when Asis.An_Expression_Function_Declaration =>
                  null;
               when Asis.An_Ordinary_Type_Declaration =>
                     declare
                        Type_Def : Asis.Definition := Asis.Declarations.Type_Declaration_View(Element);
                        Temp_Exp : Asis.Expression;
                        Task_Decl : Asis.Declaration;
                        Task_Type : Task_Type_Info_Link;
                        Keep_Next : Task_Type_Item_Link;
                     begin
                        Put_Line("1297");
                        if Asis.Elements.Type_Kind(Type_Def) = An_Access_Type_Definition then
                           Put_Line(Asis.Elements.Debug_Image(Type_Def));
                           if Asis.Elements.Access_Type_Kind(Type_Def) = An_Access_To_Function
                              or Asis.Elements.Access_Type_Kind(Type_Def) = An_Access_To_Protected_Function then
                              Put_Line("1364");
                              Temp_Exp := Asis.Definitions.Access_To_Function_Result_Profile(Type_Def);
                              Put_Line(Asis.Text.Element_Image(Temp_Exp));
                              while Asis.Elements.Access_Type_Kind(Temp_Exp) /= Not_An_Access_Type_Definition loop
                                Temp_Exp := Asis.Definitions.Access_To_Function_Result_Profile(Temp_Exp);
                              end loop;
                           else
                             if Asis.Elements.Access_Type_Kind(Type_Def) = An_Access_To_Procedure
                                 or Asis.Elements.Access_Type_Kind(Type_Def) = An_Access_To_Protected_Procedure then
                                 goto End_of_The_Odinary_Type_Declaration;
                             else
                                Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Access_To_Object_Definition(Type_Def));
                             end if;
                           end if;
                           Put_Line(Expression_Kinds'Wide_Image
                                    (Asis.Elements.Expression_Kind(Temp_Exp)));
                           while Asis.Elements.Expression_Kind(Temp_Exp) = An_Attribute_Reference loop
                             Temp_Exp := Asis.Expressions.Prefix(Temp_Exp);
                           end loop;
                           if (Asis.Elements.Expression_Kind(Temp_Exp)
                               /= An_Identifier) then
                              Temp_Exp
                                := Asis.Expressions.Selector(Temp_Exp);
                           end if;
```

```
                        Task_Decl := Asis.Expressions.Corresponding_Name_Declaration(Temp_Exp);
                        if Asis.Elements.Declaration_Kind(Task_Decl)
                          = A_Task_Type_Declaration then
                           Task_Type := Task_Type_Id_List.Search_Node(Task_Type_Decl_List, Gela_Ids.Create_Id(Task_Decl));
                           if Task_Type /= null then

                              Keep_Next
                                := Current_Unit.Decl_By_Task_Type;
                              Current_Unit.Decl_By_Task_Type := new Task_Type_Item;
                              Current_Unit.Decl_By_Task_Type.Info := Task_Type;
                              Current_Unit.Decl_By_Task_Type.Is_Access_Type := True;
                              Current_Unit.Decl_By_Task_Type.Next := Keep_Next;
                           end if;
                        end if;
                     end if;
                     <<End_of_The_Odinary_Type_Declaration>>
                  end;

            when Asis.A_Task_Type_Declaration =>
                  Enter_Unit(To_V(Asis.Declarations.Defining_Name_Image
                                  (Asis.Declarations.Names (Element) (1))),
                             A_Task_Type_Declaration);

                  Task_Type_Id_List.Set_Node(Task_Type_Decl_List,
                                             Gela_Ids.Create_Id(Element),
                                             new Task_Type_Info'(Current_Unit, null, null, null));


            when Asis.A_Protected_Type_Declaration=>
                  Enter_Protected_Frame(To_V(Asis.Declarations.Defining_Name_Image
                                             (Asis.Declarations.Names (Element) (1))));
--                First_Stmt;
                  Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                  Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                  declare
                     The_Top_Node : Nodenum_T := Get_Info(Element).Top;
                  begin
                     Put_Line("1338");
                     if Element_Id_List.IsNULLItem(Element_Id_List.SEARCH_NODE(Info_List,gela_ids.Create_Id(Element))) then
                     Put_Line(" The Top Node is Null");
                     end if;
                     Current_Unit.First_Stmt := The_Top_Node;
                     Remove_Branch(The_Top_Node, The_Top_Node+1);
                     Add_Branch(The_Top_Node, Num_Of_Nodes +1);
                  end;

            when Asis.A_Variable_Declaration=>
                  Put_Line("1435");
                  if Is_Debug_Mode then
                     Put_Line("var/con decl.");
                  end if;
                  declare
                     Self_Info : Element_Info := Get_Info(Element);
                  begin

                     if Self_Info /= Null_Element_Info then

                        Caller_Nodenum := Get_Info(Element).Top;
                     end if;

                  end;
--                Put_Line(Asis.Text.Element_Image(Asis.Expressions.Corresponding_Expression_Type(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)))));
--                declare
--                   Exp : Asis.Expression := Asis.Expressions.Corresponding_Expression_Type(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)));
--                   Exp : Asis.Expression := Asis.Expressions.Corresponding_Expression_Type(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)));
--                begin

--                   if Exp = Nil_Element then
--                declare
--                   Hoge : Asis.Element := Asis.Expressions.Corresponding_Name_Definition(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)));
--                begin

--                   null;
--                end;
--                     Put_Line(Asis.Text.Element_Image(Asis.Elements.Enclosing_Element(Asis.Expressions.Corresponding_Name_Definition(
Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem))))));
--                   else
--                      Put_Line(Asis.Text.Element_Image(Exp));
--                   end if;
--                end;

                  Clear(Vstring_Stack);
                  Clear(Var_Stack);

            when Asis.A_Constant_Declaration=>

                  if Is_Debug_Mode then
                     Put_Line("var/con decl.");
                  end if;
                  declare
                     Self_Info : Element_Info := Get_Info(Element);
                  begin

                     if Self_Info /= Null_Element_Info then

                        Caller_Nodenum := Get_Info(Element).Top;
                     end if;

                  end;
--                Put_Line(Asis.Text.Element_Image(Asis.Expressions.Corresponding_Expression_Type(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)))));
--                declare
--                   Exp : Asis.Expression := Asis.Expressions.Corresponding_Expression_Type(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)));
--                   Exp : Asis.Expression := Asis.Expressions.Corresponding_Expression_Type(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)));
--                begin

--                   if Exp = Nil_Element then
--                declare
--                   Hoge : Asis.Element := Asis.Expressions.Corresponding_Name_Definition(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)));
--                begin
```

```
--                   null;
--                end;
--                     Put_Line(Asis.Text.Element_Image(Asis.Elements.Enclosing_Element(Asis.Expressions.Corresponding_Name_Definition(
Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem))))));
--                   else
--                      Put_Line(Asis.Text.Element_Image(Exp));
--                   end if;
--                end;

                  Clear(Vstring_Stack);
                  Clear(Var_Stack);
            when Asis.A_Single_Protected_Declaration=>
                  Enter_Protected_Frame(To_V(Asis.Declarations.Defining_Name_Image
                                        (Asis.Declarations.Names (Element) (1))));
--                First_Stmt;
--                Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                  declare
                     Top_Node : Nodenum_T := Get_Info(Element).Top;
                  begin
                     Current_Unit.First_Stmt := Top_Node;
                     Remove_Branch(Top_Node, Top_Node+1);
                     Add_Branch(Top_Node, Num_Of_Nodes +1);
                  end;
            when Asis.A_Single_Task_Declaration=>
                  Enter_Task_Frame(To_V(Asis.Declarations.Defining_Name_Image
                          (Asis.Declarations.Names (Element) (1))));

            when Asis.A_Component_Declaration=>
                  Clear(Vstring_Stack);

                  Clear(Var_Stack);

                  if Current_Unit.Decl_Kind = A_Protected_Body_Declaration then



                     declare
                        Init_Exp : Asis.Expression := Asis.Declarations.Initialization_Expression(Element);
                        DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Element);
                        Type_Ident : V_String := Get_Type_Ident(Element);
--                      Decl_Is_Access : Boolean := Is_Access(Elem);
                        Decl_Is_Access : Boolean;
                     begin

                        Set_Is_Access(Element, Decl_Is_Access, Type_Ident);

                        Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));


                        Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));


                        for J in Dname_List'Range loop

--                         declare
--                            Dname : V_String := To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J)));
--                         begin

                              Assign_Variable(Enter_Variable(Dname_List(J),
                                                             Type_Ident,
                                                             Decl_Is_Access),
                                              Tmp_Nodenum);

--                            Use_Variable(Dname, Tmp_Nodenum);

--                         end;

                        end loop;
                     end;
                  end if;

            when Asis.A_Procedure_Declaration =>
--                Put_Line("procedure.");
--                Put (Asis.Declarations.Defining_Name_Image
--                        (Asis.Declarations.Names (Elem) (1)));
                  Clear(Args_Stack);
                  Make_Arg_From_Proc(Element);
                  Enter_Proceduredec(To_V(Asis.Declarations.Defining_Name_Image
                          (Asis.Declarations.Names (Element) (1))),
                                Gela_Ids.Create_Id(Element), Args_Stack);
                  Clear(Args_Stack);
            when Asis.A_Function_Declaration =>
                  Clear(Args_Stack);
                  declare
                     Temp_Exp : Asis.Expression;
                     Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Element);
                     Function_Name : V_string := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1)));
                     Tmp_Varnum : Varnum_T;
                     Position : Natural := 1;
                  begin
                     --Clear(Args_Stack);
                     --Enter_Unit(To_V(Asis.Declarations.Defining_Name_Image
                        --(Asis.Declarations.Names (Element) (1))),
                        --Asis.Elements.Declaration_Kind(Element));
                     --Enter_Functiondec(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1))),
                        --Gela_Ids.Create_Id(Element),
                        --Args_Stack, Tmp_Varnum);
                     --Clear(Args_stack);
                     --Put_Line(to_s(Current_Unit.Parent.Name));
                     if Plist /= Nil_Element_List then
                        for I in Plist'Range loop
                           declare
                              DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                           begin
                              for J in Dname_List'Range loop
                                 Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));
                                 Position := Position + 1;
```

```
                        end loop;
                     end;
                  end loop;
               end if;
               Temp_Exp := Asis.Declarations.Result_Profile(Element);
               <<Check_Temp_Exp_Kind>>
               case Asis.Elements.Expression_Kind(Temp_Exp) is
                 when A_Selected_Component =>
                   Temp_Exp := Asis.Expressions.Selector(Temp_Exp);
                   goto Check_Temp_Exp_Kind;
                 when An_Attribute_Reference =>
                   Temp_Exp := Asis.Expressions.Prefix(Temp_Exp);
                   goto Check_Temp_Exp_Kind;

                 when others =>
                   case Asis.Elements.Definition_Kind(Temp_Exp) is
                     when An_Access_Definition =>
                        Temp_Exp := Asis.Definitions.Anonymous_Access_To_Object_Subtype_Mark(Temp_Exp);
                        goto Check_Temp_Exp_Kind;
                     when others =>
                        Tmp_Varnum := Enter_Variable(Function_Name, To_V(Asis.Expressions.Name_Image(Temp_Exp)), False);
                   end case;
               end case;
               Enter_Functiondec(Function_name,
                                 Gela_Ids.Create_Id(Element), Args_Stack,
                                 Tmp_Varnum);
            end;
--                           To_V(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(Elem))), False);
         Clear(Args_Stack);
         when A_Procedure_Body_Declaration
           | A_Function_Body_Declaration
           | A_Task_Body_Declaration
           | An_Entry_Body_Declaration =>

            case Asis.Elements.Declaration_Kind(Element) is
               when A_Procedure_Body_Declaration
                 | A_Function_Body_Declaration
                 | An_Entry_Body_Declaration =>

                  Put(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1)));
                  new_line;
                  --Put(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (2)));
                  --new_line;
                  --Put(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (3)));
                  Enter_Unit(To_V(Asis.Declarations.Defining_Name_Image
                                    (Asis.Declarations.Names (Element) (1))),
                             Asis.Elements.Declaration_Kind(Element));

                  First_Stmt;

                  Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                  Node_IO.Put(Tmp_Nodenum);
                  new_line;
                  Node_IO.Put(Tmp_Nodenum+1);
                  new_line;

                  Info:= Element_Info'(Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1);

                  Set_Info(Element, Info);

                  Push(Subprogramhead_Stack, Tmp_Nodenum);

                  if Asis.Elements.Declaration_Kind(Element)
                    = An_Entry_Body_Declaration then

                     -- *** unprotected entry ???
                     Set_Info(Asis.Declarations.Entry_Barrier(Element),
                          (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));

                  end if;
                  --Put_Line("1234");
--               Put_Line("procedure body.");
--               New_Line;
--               Put (Asis.Declarations.Defining_Name_Image
--                       (Asis.Declarations.Names (Elem) (1)));

--                  Args_Nodenum := 0;

                  case Asis.Elements.Declaration_Kind(Element) is
                     when A_Procedure_Body_Declaration |
                       An_Entry_Body_Declaration =>
                        Put_Line("1594");
                        if Is_Debug_Mode then
                           Put_Line("procedure_body");
                        end if;
                        declare
                           --check : Boolean;
                           Proc_Decl : Asis.Declaration := Nil_Element;
                           Proc_Item : Subprogram_Item_Link := null;
                        begin
                           if Asis.Elements.Declaration_Kind(Element) =
                             A_Procedure_Body_Declaration then
                              Proc_Decl := Asis.Declarations.Corresponding_Declaration(Element);

--                            if Proc_Decl /= Nil_Element then
                            if not (Asis.Elements.Is_Nil(Proc_Decl)) then
                               Proc_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Proc_Decl));
                              -- Put_Line("233");
                            end if;
                            if Proc_Item = null then
                               Clear(Args_Stack);
                               declare
                                  Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Element);
                                  Position : Natural := 1;
                               begin

                                  if Plist /= Nil_Element_List then
                                     for I in Plist'Range loop
                                        declare
```

```
                        DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                     begin
                        for J in Dname_List'Range loop
                           Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));

                           Position := Position + 1;

                        end loop;
                     end;
                  end loop;
               end if;
            end;
            Enter_Proceduredec(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1))),
                               Gela_Ids.Create_Id(Element), Args_Stack);

            Clear(Args_Stack);


            Proc_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Element));
         end if;
      else

         Proc_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Search_Entry_Decl(To_V(
         Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1)))));

      end if;
      if Is_Debug_Mode then
         Ada.Text_IO.Put_Line(Nodenum_T'Image(Current_Unit.First_Stmt));
      end if;
      Put_Line("1752");
      Put_Line(to_s(Proc_Item.Name));
      Proc_Item.Callee := Current_Unit.First_Stmt;

      Add_Stacked_Call_Branch(Proc_Item.all);

      My_Args_Item := Proc_Item.Args;
      Current_Unit.Subprogram := Proc_Item;
      Put_Line("1658");
   end;
when A_Function_Body_Declaration =>
   if Is_Debug_Mode then
      Put_Line("function body.");
   end if;
   declare
      Func_Decl : Asis.Declaration := Asis.Declarations.Corresponding_Declaration(Element);
      Func_Item : Subprogram_Item_Link;
   begin
     -- Put_Line("123");
      --Put_Line("the func decl is");
      --Put_Line(asis.Text.Element_Image(Func_Decl));

      --if Func_Decl = Nil_Element then
      --Put_Line(asis.Text.Element_Image(Func_Decl));
      if asis.Elements.Is_Nil(Func_Decl) then --or Subprogram_Decl_ID_List.IsNULLItem(Subprogram_Decl_Id_List.SEARCH_NODE
      (Subprogram_Decl_List, Gela_Ids.Create_Id(Func_Decl)))  then
      --    Put_Line("666");
      --end if;
      Put_Line("1666");
      --if Subprogram_Decl_ID_List.IsNULLItem(Subprogram_Decl_Id_List.SEARCH_NODE(Subprogram_Decl_List, Gela_Ids.Create_Id(Func_Decl))) then
        if Is_Debug_Mode then
            Put_Line("no function spec.");
        end if;
        --Put_Line("123");
         Clear(Args_Stack);
         declare
            Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Element);
            Function_Name : V_string := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1)));
            Tmp_Varnum : Varnum_T;
            Position : Natural := 1;
         begin
            if Plist /= Nil_Element_List then
            Put_Line("1680");
               for I in Plist'Range loop
                  declare
                     DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                  begin
                     for J in Dname_List'Range loop
                        Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));
                        Position := Position + 1;
                     end loop;
                  end;
               end loop;
            end if;
            declare
               Keep_Current_Unit : Unit_Item_Link := Current_Unit;
            begin
              --Current_Unit := Current_Unit.Parent;
               Put_Line("1696");
               --Put_Line(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(Element)));
               Tmp_Varnum := Enter_Variable(Function_Name, To_V(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(Element))), False);
               Current_Unit := Keep_Current_Unit;
            end;
            Put_Line("1701");
            Enter_Functiondec(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1))),
                        Gela_Ids.Create_Id(Element),
                              Args_Stack, Tmp_Varnum);
         Clear(Args_Stack);

         Func_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Element));

         end;
        elsif Subprogram_Decl_ID_List.IsNULLItem(Subprogram_Decl_Id_List.SEARCH_NODE(Subprogram_Decl_List, Gela_Ids.Create_Id(Func_Decl)))  then
          if Is_Debug_Mode then
            Put_Line("no function spec.");
          end if;
        --Put_Line("123");
         Clear(Args_Stack);
         declare
            Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Element);
```

```
                              Function_Name : V_string := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1)));
                              Tmp_Varnum : Varnum_T;
                              Position : Natural := 1;
                           begin
                           Put_Line("1722");
                              if Plist /= Nil_Element_List then
                                 for I in Plist'Range loop
                                    declare
                                       DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                                    begin
                                       for J in Dname_List'Range loop
                                          Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));
                                          Position := Position + 1;
                                       end loop;
                                    end;
                                 end loop;
                              end if;
                              declare
                                 Keep_Current_Unit : Unit_Item_Link := Current_Unit;
                              begin
                                --Current_Unit := Current_Unit.Parent;

                                 --Put_Line(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(Element)));
                                 Tmp_Varnum := Enter_Variable(Function_Name, To_V(Asis.Text.Element_Image(Asis.Declarations.Result_Profile(Element))), False);
                                 Current_Unit := Keep_Current_Unit;
                              end;
                              Put_Line("1761");
                              Enter_Functiondec(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1))),
                                          Gela_Ids.Create_Id(Element),
                                                Args_Stack, Tmp_Varnum);
                           Clear(Args_Stack);

                           Func_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Element));
                           end;


--                                       To_V(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(Element))), False );
                           --Clear(Args_Stack);

                           --Func_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Element));

                        else
                           if Is_Debug_Mode then
                              Put_Line("function spec detected.");
                           end if;
                           Func_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Func_Decl));
                        end if;
                        ada.Text_IO.Put_Line(Nodenum_T'Image(Current_Unit.First_Stmt));
                        Func_Item.Callee := Current_Unit.First_Stmt;
                        Add_Stacked_Call_Branch(Func_Item.all);
                        My_Args_Item := Func_Item.Args;
                        Current_Unit.Subprogram := Func_Item;
                        end;
                  when others => null;
               end case;
               -- add formal-in, out node
               declare
                  Decl : Asis.Declaration := Nil_Element;
                  Tmp_Subprogram : Subprogram_Item_Link;
               begin
                  if Asis.Elements.Declaration_Kind(Element) /= An_Entry_Body_Declaration then
                     Decl := Asis.Declarations.Corresponding_Declaration(Element);
                     --Ada.Wide_Text_IO.Put(Decl);
                     --new_line;
                  else
                     Decl := Prot_Entry_Decl_List.Search_Node(Protected_Entry_List, Search_Entry_Decl(
                     To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1)))));
                  end if;
--                  if Decl = Nil_Element then
                  if Asis.Elements.Is_Nil(Decl) then
                     Decl := Element;
                  end if;
                  declare
                     Plist : Asis.Parameter_Specification_List :=
                       Asis.Declarations.Parameter_Profile(Decl);
                  begin
                     Tmp_Subprogram := Subprogram_Decl_ID_List.SEARCH_NODE(Subprogram_Decl_List, gela_ids.Create_Id(Decl));
                     if Plist /= Nil_Element_List then
                        if Is_Param_Node_Approach then
                           Set_Arg_Branch(Tmp_Nodenum);
                        end if;
                        for I in Plist'Range loop
--                         Put_Line(Asis.Elements.Debug_Image(Asis.Declarations.Declaration_Subtype_Mark(Plist(I))));
                           declare
                              DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                              Tmp_Varnum : Varnum_T;
                              Type_Ident : V_String;
                              Temp_Def : Asis.Definition;--                                    Decl_Is_Access : Boolean := Is_Access(Plist(I));
                              Decl_Is_Access : Boolean;
                           begin
                              Temp_Def := Asis.Declarations.Object_Declaration_View(Plist(I));
                              if Asis.Elements.Access_Definition_Kind(Temp_Def) = An_Anonymous_Access_To_Procedure
                              or Asis.Elements.Access_Definition_Kind(Temp_Def) = An_Anonymous_Access_To_Protected_Procedure then
                                 goto End_of_The_Loop;
                              else
                                 Type_Ident := Get_Type_Ident(Plist(I));
                                 Put_Line("1925");
                                 Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                                 Put_Line("1926");
                                for J in Dname_List'Range loop
                                 --Tmp_Varnum := Enter_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))));
                                    Tmp_Varnum := Enter_Variable(Dname_List(J), Type_Ident, Decl_Is_Access);
                                    case Asis.Elements.Mode_Kind(Plist(I)) is
                                       when A_Default_In_Mode | An_In_Mode
                                         | An_In_Out_Mode =>
                                        if Is_Param_Node_Approach then
                                           Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Plist(I))));
                                           Set_Arg_Branch(Tmp_Nodenum);
                                           Add_Top_Arg_Branch(Tmp_Nodenum);
```

```
                                                Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
                                                Remove_Branch(Tmp_Nodenum-1, Tmp_Nodenum);
                                                --if Tmp_Subprogram.Precon_Nodenum /= 0 then
                                                 --Add_Branch(Tmp_Subprogram.Precon_Nodenum, Tmp_NodeNum);
                                                 --Add_Branch(Current_Unit.First_Stmt, Tmp_Subprogram.Precon_Nodenum);
                                                --else
                                                 Add_Branch(Current_Unit.First_Stmt, Tmp_Nodenum);
                                                --end if;
                                                Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                                                Set_Info(Plist(I), (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                                                Assign_Variable(Tmp_Varnum, Tmp_Nodenum);
                                                declare
                                                   Arg_Item : Args_Item_Link := Search_Args_Item(My_Args_Item, To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))));
                                                begin
                                                   if Arg_Item /= null then
                                                      Arg_Item.Formal_In := Tmp_Nodenum;
                                                      Add_Stacked_Arg_Branch_In(Arg_Item.all);
                                                   end if;
                                                end;
                                             else
                                                --if Tmp_Subprogram.Precon_Nodenum /= 0 then
                                                 --Add_Formal_In(Tmp_Varnum, Tmp_Subprogram.Precon_Nodenum);
                                                --else
                                                 Add_Formal_In(Tmp_Varnum, Tmp_Nodenum);
                                                --end if;
                                             end if;
                                          when others =>
                                             null;
                                       end case;
                                       if not Is_Param_Node_Approach then
                                          case Asis.Elements.Mode_Kind(Plist(I)) is
                                             when An_In_Out_Mode | An_Out_Mode =>
                                                --if Tmp_Subprogram.Postcon_Nodenum /=0 then
                                                 --Add_Formal_Out(Tmp_Varnum, Tmp_Subprogram.Postcon_Nodenum);
                                                --else
                                                 Add_Formal_Out(Tmp_Varnum, Tmp_Nodenum);
                                                --end if;
                                             when others =>
                                                null;
                                          end case;
                                       end if;
                                    end loop;
                                 end if;
                              end;
                           <<End_of_The_Loop>>
                           end loop;
                           Use_Arg_Stacks;
                        end if;
                     end;
                  end;


               when A_Task_Body_Declaration =>
                  Enter_Task(To_V(Asis.Declarations.Defining_Name_Image
                                  (Asis.Declarations.Names (Element) (1))));
                  First_Stmt;
                  Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
                  Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                  Push(Taskhead_Stack, Tmp_Nodenum);
               when others =>
                  null;

            end case;
            declare
               Dlist : Asis.Element_List := Asis.Declarations.Body_Declarative_Items(Element);
            begin
               Dlist_Add(Dlist);
            end;

--          Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Asis.Declarations.Body_Block_Statement(Element)))-1); -- for begin statement
            Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Asis.Declarations.Body_Statements(Element)(1)))-1); -- for begin statement
            Set_Fork_Point;
         -- when Asis.A_Procedure_Body_Declaration =>
          --   Ada.wide_Text_IO.Put("A_Procedure_Body_Declaration: ");
            -- Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.First_Line_Number(Element)));
            -- Ada.wide_Text_IO.Put("-");
           --   Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.Last_Line_Number(Element)));
            --  Ada.wide_Text_IO.Put(":");
            --  Ada.Wide_Text_IO.New_Line;
            --  Ada.Wide_Text_IO.Put(Asis.Text.Element_Image(Element));
            --  Ada.Wide_Text_IO.New_Line;

            --  Metrics_Utilities.Defining_Names :=
             --   Metrics_Utilities.Defining_Names +
              --    Asis.Declarations.Names (Element)'Length;


         -- when Asis.A_Function_Body_Declaration =>
            -- Ada.wide_Text_IO.Put("A_Function_Body_Declaration: ");
         --    Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.First_Line_Number(Element)));
         --    Ada.wide_Text_IO.Put("-");
         --    Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.Last_Line_Number(Element)));
           --   Ada.wide_Text_IO.Put(":");
           --   Ada.Wide_Text_IO.New_Line;
            --  Ada.Wide_Text_IO.Put(Asis.Text.Element_Image(Element));
            --  Ada.Wide_Text_IO.New_Line;

         --    Metrics_Utilities.Defining_Names :=
            --    Metrics_Utilities.Defining_Names +
           --      Asis.Declarations.Names (Element)'Length;
          when Asis.An_Entry_Declaration =>
                Clear(Args_Stack);
                declare
                   Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Element);
                begin
                   if Plist /= Nil_Element_List then
                      for I in Plist'Range loop
                         declare
                            DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
```

```
                        Type_Ident : V_String := Get_Type_Ident(Plist(I));
                        Tmp_Varnum : Varnum_T;
--                      Decl_Is_Access : Boolean := Is_Access(Plist(I));
                        Decl_Is_Access : Boolean;
                        Position : Natural := 1;
                     begin
                        Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                        for J in Dname_List'Range loop
                           -- Put_Line(Asis.Declarations.Defining_Name_Image(Dname_List(J)));
                           if Current_Unit.Decl_Kind /= A_Protected_Body_Declaration then
                              Tmp_Varnum := Enter_Variable(Dname_List(J), Type_Ident, Decl_Is_Access);
                           else

                              Prot_Entry_Decl_List.Set_Node(Protected_Entry_List, Gela_Ids.Create_Id(Element), Element);
                           end if;
                        Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));
                        Position := Position + 1;
                        end loop;
                     end;
                  end loop;
               end if;
            end;

            Enter_Entrydec(To_V(Asis.Declarations.Defining_Name_Image
                    (Asis.Declarations.Names (Element) (1))),
                           Gela_Ids.Create_Id(Element), Args_Stack);
            Clear(Args_Stack);

         when Asis.A_Package_Body_Declaration =>
            if Current_Unit /= Main_Task then
               Enter_Package(To_V(Asis.Declarations.Defining_Name_Image
                                  (Asis.Declarations.Names (Element) (1))));
               First_Stmt;
               declare
                  Top_Node : Nodenum_T := Get_Info(Element).Top;
                  Bottom_Node : Nodenum_T := Get_Info(Element).Bottom;
                  Dlist : Asis.Element_List := Asis.Declarations.Body_Declarative_Items(Element);
                  B_Statements : Asis.Statement_List := Asis.Declarations.Body_Statements(Element);

               begin
                  Remove_Branch(Top_Node, Bottom_Node);
                  Add_Branch(Top_Node, Num_Of_Nodes +1);
                  Dlist_Add(Dlist);
--                Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Asis.Declarations.Body_Block_Statement(Elem)))-1); -- for begin statement
                  if B_Statements /= Nil_Element_List then
                     Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(B_Statements(1)))-1); -- for begin statement
                     Set_Fork_Point;
                  end if;
               end;
            else
               Enter_Package(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1))));
               Current_Unit.Subprogram := new Subprogram_Item;
               Current_Unit.Subprogram.Name := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1)));
               First_Stmt;
               Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Element)));
               Set_Info(Element, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
               declare
                  Dlist : Asis.Element_List := Asis.Declarations.Body_Declarative_Items(Element);
               begin
                  Dlist_Add(Dlist);
               end;
               Set_Fork_Point;
            end if;

         -- when Asis.A_Task_Body_Declaration =>
           --  Ada.wide_Text_IO.Put("A_Task_Body_Declaration : ");
           --  Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.First_Line_Number(Element)));
           --  Ada.wide_Text_IO.Put("-");
           --  Ada.Wide_Text_IO.Put(Asis.Text.Line_Number'Wide_Image(Asis.Text.Last_Line_Number(Element)));
           --  Ada.wide_Text_IO.Put(":");
           --  Ada.Wide_Text_IO.New_Line;
           --  Ada.Wide_Text_IO.Put(Asis.Text.Element_Image(Element));
           --  Ada.Wide_Text_IO.New_Line;

           --  Metrics_Utilities.Defining_Names :=
           --    Metrics_Utilities.Defining_Names +
            --     Asis.Declarations.Names (Element)'Length;


         when Asis.A_Package_Declaration =>
            --Put_Line("12345");
               Put_Line(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1)));
               Enter_Package_Frame(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1))),
                                   Asis.Elements.Declaration_Kind(Element));
           -- Put_Line("54321");
               if Current_Unit.Parent = Main_Task then
                  declare
                     Vlist : Asis.Element_List := Asis.Declarations.Visible_Part_Declarative_Items(Element);
                     Plist : Asis.Element_List := Asis.Declarations.Private_Part_Declarative_Items(Element);
                  begin
                     Dlist_Add(Vlist);
                     if Asis.Declarations.Is_Private_Present(Element) then
                        Put_Line("2152");

                        Dlist_Add(Plist);
                        Put_Line("2154");
                     end if;
                  end;
                  if Current_Unit.Nodes /= null then
                     Current_Unit.Fork_Point := Num_Of_Nodes;
                  end if;
               end if;
         when Asis.A_Generic_Package_Declaration =>
               Enter_Package_Frame(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Element) (1))),
                                   Asis.Elements.Declaration_Kind(Element));
               if Current_Unit.Parent = Main_Task then
                  declare
                     Flist : Asis.Element_List := Asis.Declarations.Generic_Formal_Part(Element);
                     Vlist : Asis.Element_List := Asis.Declarations.Visible_Part_Declarative_Items(Element);
```

```
                  Plist : Asis.Element_List := Asis.Declarations.Private_Part_Declarative_Items(Element);
               begin
                  if Is_Debug_Mode then
                     Put_Line("*** generic package formal part");
                  end if;
                  Dlist_Add(Flist);
                  if Is_Debug_Mode then
                     Put_Line("*** generic package visible part");
                  end if;
                  Dlist_Add(Vlist);
                  if Asis.Declarations.Is_Private_Present(Element) then
                     if Is_Debug_Mode then
                        Put_Line("*** generic package private part");
                     end if;
                     Dlist_Add(Plist);
                  end if;
               end;
               if Current_Unit.Nodes /= null then
                  Current_Unit.Fork_Point := Num_Of_Nodes;
               end if;
            end if;


         when Asis.A_Protected_Body_Declaration =>
               Enter_Protected(To_V(Asis.Declarations.Defining_Name_Image
                                 (Asis.Declarations.Names (Element) (1))));

         when others =>
            null;
      end case;



      --  To compute the total number of all the explicitly declared
      --  names, we have to take into account that some declarations
      --  may define more then one name. Therefore for each declaration
      --  we have to get the list of the declared names and to increase
      --  the corresponding metric counter by the number of the names
      --  in this list.

      -- Metrics_Utilities.Defining_Names :=
      --    Metrics_Utilities.Defining_Names +
      --    Asis.Declarations.Names (Element)'Length;

   when others =>
      null;
end case;
exception

   when Ex : Asis.Exceptions.ASIS_Inappropriate_Context          |
        Asis.Exceptions.ASIS_Inappropriate_Container        |
        Asis.Exceptions.ASIS_Inappropriate_Compilation_Unit |
        Asis.Exceptions.ASIS_Inappropriate_Element          |
        Asis.Exceptions.ASIS_Inappropriate_Line             |
        Asis.Exceptions.ASIS_Inappropriate_Line_Number      |
        Asis.Exceptions.ASIS_Failed                         =>

      Ada.Wide_Text_IO.Put ("Pre_Op : ASIS exception (");

      Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
                            Ada.Exceptions.Exception_Name (Ex)));

      Ada.Wide_Text_IO.Put (") is raised");
      Ada.Wide_Text_IO.New_Line;

      Ada.Wide_Text_IO.Put ("ASIS Error Status is ");

      Ada.Wide_Text_IO.Put
        (Asis.Errors.Error_Kinds'Wide_Image (Asis.Implementation.Status));

      Ada.Wide_Text_IO.New_Line;

      Ada.Wide_Text_IO.Put ("ASIS Diagnosis is ");
      Ada.Wide_Text_IO.New_Line;
      Ada.Wide_Text_IO.Put (Asis.Implementation.Diagnosis);
      Ada.Wide_Text_IO.New_Line;

      Asis.Implementation.Set_Status;

   when Ex : others =>

      Ada.Wide_Text_IO.Put ("Pre_Op : ");

      Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
                            Ada.Exceptions.Exception_Name (Ex)));

      Ada.Wide_Text_IO.Put (" is raised (");

      Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
                            Ada.Exceptions.Exception_Information (Ex)));

      Ada.Wide_Text_IO.Put (")");
      Ada.Wide_Text_IO.New_Line;

end Pre_Op;
```

A.1.3 Actuals_For_Traversing-Post_Op Package

```
------------------------------------------------------------------------------
--                                                                          --
--                  ASIS APPLICATION TEMPLATE COMPONENTS                    --
--                                                                          --
--         A C T U A L S _ F O R _ T R A V E R S I N G . P O S T _ O P      --
--                                                                          --
--                               B o d y                                    --
--                                                                          --
--           Copyright (c) 2000, Free Software Foundation, Inc.             --
```

```
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Application Templates were developed and are now maintained by Ada  --
-- Core Technologies Inc (http://www.gnat.com).                             --
--                                                                          --
------------------------------------------------------------------------------
--Bo Wang reconstitution 2014.12

with Ada.Wide_Text_IO;
with Ada.Characters.Handling;
with Ada.Exceptions;
with Asis.Definitions;
with Asis.Set_Get;
with Asis.Exceptions;
with Asis.Errors;
with Asis.Implementation;

separate (Actuals_For_Traversing)
procedure Post_Op
  (Elem :        Asis.Element;
   Control : in out Asis.Traverse_Control;
   State   : in out Traversal_State)
is
      use Asis;
      Tmp_Nodenum : Nodenum_T;

--      procedure Use_Stacks(Nodenum : in Nodenum_T :=0) is
      procedure Use_Stacks is
         Self_Info : Element_Info := Get_Info(Elem);
         Self_Node : Nodenum_T;

--       Inserted_Node : Nodenum_T;
      begin
         if Self_Info /= Null_Element_Info then
            Self_Node := Self_Info.Top;
--       if Nodenum /= 0 then
--          Self_Node := Nodenum;
--       end if;
            while not isempty(vstring_stack) loop
               use_variable(top(vstring_stack), Self_Node);
               remove(vstring_stack);
            end loop;

            while not isempty(var_stack) loop
               use_variable(top(var_stack), Self_Node);
               remove(var_stack);
            end loop;

            while not isempty(Var_item_stack) loop
               use_variable(top(Var_Item_stack).number, Self_Node, Top(Var_Item_Stack).Is_Normalized, Top(Var_Item_Stack).Normalized_Form);
               remove(Var_Item_stack);
            end loop;


         else
            Clear(Vstring_Stack);
            Clear(Var_Stack);
            Clear(Var_Item_Stack);
--       Inserted_Node := Self_Node;
--       while not Isempty(Fnode_stack) loop
--          Insert_Function(Top(Fnode_stack), Inserted_Node, Elem);
--          Remove(Fnode_stack);
--       end loop;

         end if;

      end Use_Stacks;


      procedure Assign_Stacks is
         Self_Node : Nodenum_T := Get_Info(Elem).Top;
      begin

         while not isempty(A_var_stack) loop
            assign_variable(top(A_var_stack), Self_Node);
            remove(A_var_stack);
         end loop;

         while not isempty(A_Var_item_stack) loop
            assign_variable(top(A_Var_Item_stack).number, Self_Node, Top(A_Var_Item_Stack).Is_Normalized, Top(A_Var_Item_Stack).Normalized_Form);
            remove(A_Var_Item_stack);
         end loop;

      end Assign_Stacks;
begin
      case Asis.Elements.Element_Kind(Elem) is
         when Not_An_Element =>
            null;
      when A_Declaration =>
            case Asis.Elements.Declaration_Kind(Elem) is
               when A_Task_Type_Declaration | A_Single_Task_Declaration =>
                  Exit_Task_Frame;

               when A_Protected_Body_Declaration =>
                  Exit_Protected;

               when A_Package_Declaration | A_Generic_Package_Declaration =>
                  Exit_Package;
```

```
                when A_Package_Body_Declaration =>
                   Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
                   if Current_Unit.Fork_Point = 0 then
                      Set_Fork_Point;
                   end if;
                   Add_P_Branch_To_Child;
                   Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                   if Current_Unit.Parent /= Main_Task then
                      declare
                         Bottom_Node : Nodenum_T := Get_Info(Elem).Bottom;
                         Node_Item : Node_Item_Link;
                      begin
                         Add_Branch(Tmp_Nodenum, Bottom_Node);
                         if Current_Unit.Fork_Point = Tmp_Nodenum then
                            Node_Item := Search_Node(Bottom_Node);
                            Node_Item.Receive := new Dug_Channel_Item'(Current_Unit, null, False, Node_Item.Receive);
                         end if;
                      end;
                   end if;
                   Exit_Package;

                when A_Procedure_Body_Declaration
                  | A_Function_Body_Declaration
                  | An_Entry_Body_Declaration =>
                   declare
                      Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Elem);
                      Subprog_Decl : Asis.Declaration;
                      Subprog_Item : Subprogram_Item_Link;
                      Keep_Nodenum : Nodenum_T;
                   begin
                      Connect_Labels(Num_Of_Nodes +1, Label_T(Top(subprogramhead_Stack)));
                      Remove(subprogramhead_Stack);

                      Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
                      Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                      Keep_Nodenum := Tmp_Nodenum;
                      if Is_Param_Node_Approach then
                         if Plist /= Nil_Element_List then
                            Set_Arg_Branch(Keep_Nodenum);
                            if Asis.Elements.Declaration_Kind(Elem) = An_Entry_Body_Declaration then
                               Subprog_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Search_Entry_Decl(To_V(
                               Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1)))));
                            else
                               Subprog_Decl := Asis.Declarations.Corresponding_Declaration(Elem);
--                             if Subprog_Decl = Nil_Element then
                               if Asis.Elements.Is_Nil(Subprog_Decl) then
                                  Subprog_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Elem));
                               else
                                  Subprog_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Subprog_Decl));
                               end if;
                            end if;

                            for I in Plist'Range loop
                               case Asis.Elements.Mode_Kind(Plist(I)) is
                                  when An_Out_Mode | An_In_Out_Mode =>
                                     declare
                                        Dname_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                                        First_Nodenum : Nodenum_T := 0;
                                     begin
                                        for J in Dname_List'Range loop
                                           Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Plist(I))));
                                           Add_Top_Arg_Branch(Tmp_Nodenum);
                                           Remove_Branch(Tmp_Nodenum -1, Tmp_Nodenum);
                                           Add_Branch(Keep_Nodenum, Tmp_Nodenum);
                                           Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
                                           if First_Nodenum = 0 then
                                              First_Nodenum := Tmp_Nodenum;
                                           end if;
                                           Set_Info(Plist(I), (First_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                                           Use_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Tmp_Nodenum);
                                           declare
                                              Arg_Item : Args_Item_Link := Search_Args_Item(Subprog_Item.Args, To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))));
                                           begin
                                              if Arg_Item /= null then
                                                 Arg_Item.Formal_Out := Tmp_Nodenum;
                                                 Add_Stacked_Arg_Branch_Out(Arg_Item.all);
                                              end if;
                                           end;

                                        end loop;
                                     end;
                                  when others =>
                                     null;
                               end case;
                            end loop;
                         end if;
                      end if;

                      if Current_Unit.Parent /= null and then
                        Current_Unit.Parent.Decl_Kind
                        = Asis.A_Protected_Body_Declaration
                      then
                         declare
                            First_Node : Node_Item_Link := Search_Node(Current_Unit.First_Stmt);
                            Last_Node : Node_Item_Link := Search_Node(Keep_Nodenum);
                         begin
                         -- protected variable
                         -- protected procedure/function/entry

--                          First_Node.Assign := Current_Unit.Protected_Use;
--                          Last_Node.Refer := Current_Unit.Protected_Def;

                            First_Node.Receive := new Dug_Channel_Item'(Current_Unit.Parent, null, False, First_Node.Receive);
                            Last_Node.Send := new Dug_Channel_Item'(Current_Unit.Parent, null, False, Last_Node.Send);
                         end;
                      end if;
                   end;
```

```
                  Add_P_Branch_To_Child;
                  Add_P_Branch_From_Child;
                  Exit_Unit;
               when A_Task_Body_Declaration =>
                  Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
                  Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                  Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, 0));

                  Connect_Labels(Tmp_Nodenum, Label_T(Top(Taskhead_Stack)));
                  Remove(Taskhead_Stack);

                  Add_P_Branch_To_Child;
                  Add_P_Branch_From_Child;
                  if Current_Unit.Decl_Kind = A_Task_Type_Declaration then
                     Add_Stacked_P_Branch(Task_Type_Id_List.Search_Node(Task_Type_Decl_List, Gela_Ids.Create_Id(Asis.Declarations.Corresponding_Declaration(Elem))));
                  end if;
                  Exit_Task;

               when A_Single_Protected_Declaration
                 | A_Protected_Type_Declaration =>
--                Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
--                Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
--                Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, 0));

--                Protected_Last_Node(Tmp_Nodenum);
                  declare
                     Top_Node : Nodenum_T := Get_Info(Elem).Top;
                     Bottom_Node : Nodenum_T := Get_Info(Elem).Bottom;
                  begin
                     if Search_Branch(Top_Node, Num_Of_Nodes +1) then
                        Remove_Branch(Top_Node, Num_Of_Nodes +1);
                        Add_Branch(Top_Node, Bottom_Node);
                     else
                        Remove_Branch(Num_Of_Nodes, Num_Of_Nodes + 1);
                        Add_Branch(Num_Of_Nodes, Bottom_Node);
                     end if;
                  end;

                  Exit_Protected_Frame;

               when A_Variable_Declaration |
                 A_Constant_Declaration  =>
                  Use_Stacks;
               --when A_Function_Declaration =>
                 --Exit_Unit;
               when others =>
                  null;
            end case;

         when A_Definition =>
            case Asis.Elements.Definition_Kind ( Elem ) is
               when A_Discrete_Range =>
                  if Asis.Elements.Discrete_Range_Kind(Elem) = A_Discrete_Simple_Expression_Range then
                     Disc_Range_Flag := False; -- ASIS bug
                  end if;
                  null;
               when others =>
                  null;
            end case;

         when An_Association =>
            case Asis.Elements.Association_Kind ( Elem ) is
               when A_Parameter_Association =>
               if Skip_Process = True then
                  goto End_Of_Parameter_Association_Post;
               end if;
                  declare
                     Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
                     Param_Name : V_String;
                     Temp_Exp : Asis.Expression;
                     Arg_Item : Args_Item_Link;
                  begin
--                   if Current_Call.Is_Same_Compilation_Unit then
                     if Current_Call.Args /= null then
                        if Asis.Expressions.Is_Normalized(Elem) then
                           Param_Name := To_V(Asis.Declarations.Defining_Name_Image(Asis.Expressions.Formal_Parameter(Elem)));
                           Arg_Item := Search_Args_Item(Current_Call.Args, Param_Name);
                        else
                           Temp_Exp := Asis.Expressions.Formal_Parameter(Elem);
--                         if Temp_Exp = Nil_Element then
                           if Asis.Elements.Is_Nil(Temp_Exp) then
                              Arg_Item := Current_Call.Args;
                              Current_Call.Args := Current_Call.Args.Next;
                              Call_Stacks.Remove(Call_Stack);
                              Call_Stacks.Push(Call_Stack, Current_Call);
                              -- post op.
                           else
                              Arg_Item := Search_Args_Item(Current_Call.Subprogram_Called.Args, To_V(Asis.Expressions.Name_Image(Temp_Exp)));
                           end if;

                        end if;

                        if not Is_Param_Node_Approach and Current_Call.Is_Same_Compilation_Unit then
                           case Arg_Item.Mode is
                              when A_Default_In_Mode | An_In_Mode =>
                                 Add_Actual_In(Current_Call.Call_Branch.Actual_In, Arg_Item.Position, False);
                              when An_In_Out_Mode =>
                                 Add_Actual_In(Current_Call.Call_Branch.Actual_In, Arg_Item.Position, True);
                                 Add_Actual_Out(Current_Call.Call_Branch.Actual_Out, Arg_Item.Position, False);
                              when An_Out_Mode =>
                                 Add_Actual_Out(Current_Call.Call_Branch.Actual_Out, Arg_Item.Position, True);
                              when others =>
                                 null;
                           end case;

                        end if;

                        if Arg_Item.Mode = An_In_Out_Mode
                          or Arg_Item.Mode = An_Out_Mode then
```

```
                        if Current_Call.Is_Same_Compilation_Unit then
                           if Is_Param_Node_Approach then
                              Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                              Set_Arg_Branch(Tmp_Nodenum);
                              Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                              Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                              if Arg_Item.Mode = An_Out_Mode then
                                 Add_Branch(Current_Call.First_Node, Tmp_Nodenum);
                              end if;
                              declare
                                 Self_Node : Nodenum_T := Get_Info(Elem).Top;
                              begin
                                 Set_Info(Elem, (Self_Node, Tmp_Nodenum, Tmp_Nodenum + 1));
                              end;
                              while not Isempty(Var_Stack) loop
                                 Assign_Variable(Top(Var_Stack),
                                                 Tmp_Nodenum);
                                 Remove(Var_Stack);
                              end loop;

                              while not Isempty(Var_Item_Stack) loop
                                 Assign_Variable(Top(Var_Item_Stack).Number,
                                                 Tmp_Nodenum,
                                                 Top(Var_Item_Stack).Is_Normalized,
                                                 Top(Var_Item_Stack).Normalized_Form
                                                 );
                                 Remove(Var_Item_Stack);
                              end loop;
                              if Arg_Item.Formal_Out = 0 then
                                 Nodenum_Stack.Push(Arg_Item.Actual_Out,
                                                    Tmp_Nodenum);
                              else
                                 Add_Arg_Branch(Arg_Item.Formal_Out,
                                                Tmp_Nodenum);
                              end if;
                           end if;

                        elsif Current_Call.CallKind = An_Entry_Call then
                           Out_Arg_Stacks.Push(Current_Call.Out_Arg_Stack, Out_Arg_Item'(Var_Stack, Var_Item_Stack, Arg_Item));
                           Call_Stacks.Remove(Call_Stack);
                           Call_Stacks.Push(Call_Stack, Current_Call);
                           if Arg_Item.Mode = An_Out_Mode then
                              Clear(Vstring_Stack);
                              Clear(Var_Stack);
                           end if;
                        else
                           Tmp_Nodenum := Caller_Nodenum;
                           Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                           while not Isempty(Var_Stack) loop
                              Assign_Variable(Top(Var_Stack),
                                              Tmp_Nodenum);
                              if Arg_Item.Mode = An_In_Out_Mode then
                                 Use_Variable(Top(Var_Stack),
                                              Tmp_Nodenum);
                              end if;
                              Remove(Var_Stack);
                           end loop;

                           while not Isempty(Var_Item_Stack) loop
                              Assign_Variable(Top(Var_Item_Stack).Number,
                                              Tmp_Nodenum,
                                              Top(Var_Item_Stack).Is_Normalized,
                                              Top(Var_Item_Stack).Normalized_Form
                                              );
                              if Arg_Item.Mode = An_In_Out_Mode then
                                 Use_Variable(Top(Var_Item_Stack).Number,
                                              Tmp_Nodenum,
                                              Top(Var_Item_Stack).Is_Normalized,
                                              Top(Var_Item_Stack).Normalized_Form
                                              );
                              end if;
                              Remove(Var_Item_Stack);
                           end loop;


                        end if;

--                      if not Out_Arg_Stacks.Isempty(Current_Call.Out_Arg_Stack) then
--                         declare
--                            Out_Var_Stack : Varnum_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Stack;
--                            Out_Var_Item_Stack : Var_Item_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Item_Stack;
--                         begin

--                         declare
--                            Arg_Item : Args_Item_Link := Out_Arg_Stacks.Top(Current_Call.Out_Arg_stack).Arg_Item;
--                         begin
--                         end;
                           -- out_arg
--                         Out_Arg_Stacks.Push(Current_Call.Out_Arg_Stack, Out_Arg_Item'(Var_Stack, Var_Item_Stack, Arg_Item));
--                         Call_Stacks.Remove(Call_Stack);
--                         Call_Stacks.Push(Call_Stack, Current_Call);
                        end if;
                        Set_Info(Elem, (Caller_Nodenum, Caller_Nodenum, Caller_Nodenum + 1));
                        Use_Stacks;

                     end if;
                  end;
                  <<End_Of_Parameter_Association_Post>>
--                Put("parameter_association : ");
--                Put_Line(Asis.Text.Element_Image(Elem));
--                declare
--                   Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
--                   Param_Name : V_String;
--                   Arg_Item : Args_Item_Link;
--                begin
--                   if Current_Call.Callkind = A_Procedure_Call or
--                     Current_Call.Callkind = A_Function_Call or
--                     Current_Call.Callkind = An_Entry_Call then
--                      if Current_Call.Is_Same_Compilation_Unit then
```

```
--                        if not Current_Call.Has_Node then
--                           Tmp_Nodenum := Add_Out_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                           Current_Call.First_Node := Tmp_Nodenum;
--                           if Current_Call.Subprogram_Called.Callee = 0 then
--                              Nodenum_Stack.Push(Current_Call.Subprogram_Called.Caller, Tmp_Nodenum);
--                           else
--                              Add_Call_Branch(Tmp_Nodenum, Current_Call.Subprogram_Called.Callee);
--                           end if;
--                           Current_Call.Has_Node := True;
--                        end if;
--                        if Asis.Expressions.Is_Normalized(Elem) then
--
--                           Param_Name := To_V(Asis.Declarations.Defining_Name_Image(Asis.Expressions.Formal_Parameter(Elem)));
--                           Arg_Item := Search_Args_Item(Current_Call.Args, Param_Name);
--                        else
--
--                           Arg_Item := Current_Call.Args;
--                           Current_Call.Args := Current_Call.Args.Next;
--                           Call_Stacks.Remove(Call_Stack);
--                           Call_Stacks.Push(Call_Stack, Current_Call);
-- --                        Param_Name := To_V(Asis.Expressions.Name_Image(Asis.Expressions.Formal_Parameter(Elem)));
--                        end if;
--
--                        if Arg_Item = null then
--                           Put_Line("can't find arg_item...");
--                        end if;
--                        if Arg_Item.Mode /= An_Out_Mode then
--                           if Current_Call.Callkind = A_Procedure_Call then
--                              Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                           else
--                              Tmp_Nodenum := Add_Out_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                           end if;
--                           Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
--                           Use_Stacks;
--                           if Arg_Item.Formal_In = 0 then
--                              Nodenum_Stack.Push(Arg_Item.Actual_In,
--                                                 Tmp_Nodenum);
--                           else
--                              Add_Arg_Branch(Tmp_Nodenum,
--                                             Arg_Item.Formal_In);
-- --                           Put("arg_item.name : ");
-- --                           Put_Line(To_S(Arg_Item.Name));
-- --                           Put("formal_in : ");
-- --                           Node_IO.Put(Arg_Item.Formal_In);
-- --                           New_Line;
-- --                           Put("next formal_in : ");
-- --                           Node_IO.Put(Arg_Item.Next.Formal_In);
-- --                           New_Line;

--                           end if;
--                        end if;
--
--                        if Arg_Item.Mode = An_In_Out_Mode
--                          or Arg_Item.Mode = An_Out_Mode then
--                           Push(Out_Arg_Stack, Out_Arg_Item'(To_V(Asis.Expressions.Name_Image(Asis.Expressions.Actual_Parameter(Elem))), Arg_Item));
--                        end if;
--                     else
--                        null;
--                     end if;
--                  end if;
--               end;

               when others =>
                  null;
            end case;
         when An_Expression =>
            case Asis.Elements.Expression_Kind ( Elem ) is
               when A_Function_Call =>
                  if Is_Debug_Mode then
                     Put("function call : ");
                     Put_Line(Asis.Text.Element_Image(Elem));
                  end if;
                  declare
                     Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
                     Ident : V_String := To_V(Unique_Identifier);
--                   Fnode : Function_Nodes;
                  begin
                     Put_Line("527 in post");
                     if Current_Call.Is_Same_Compilation_Unit then
                        if not Is_Param_Node_Approach then
                           Add_Left_Actual_Label(Current_Call.Subprogram_Called.Args, Current_Call.Call_Branch);
                        end if;
                        Push(Var_Stack, Current_Call.Subprogram_Called.Function_Varnum);
--                      if not Current_Call.Has_Node then
--                         Tmp_Nodenum := Add_Out_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                         Current_Call.First_Node := Tmp_Nodenum;
--                         if Current_Call.Subprogram_Called.Callee = 0 then
--                            Nodenum_Stack.Push(Current_Call.Subprogram_Called.Caller, Tmp_Nodenum);
--                         else
--                            Add_Call_Branch(Tmp_Nodenum, Current_Call.Subprogram_Called.Callee);
--                         end if;
--                         Current_Call.Has_Node := True;
--                      end if;

--                      Tmp_Nodenum := Add_Out_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                      Remove_Branch(Tmp_Nodenum, Tmp_Nodenum+1);
--                      Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum));
--                      Assign_Variable(Enter_Variable(Ident, To_V(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(
Asis.Expressions.Corresponding_Called_Function(Elem)))), False), Tmp_Nodenum);
--                      Fnode.Start_Node := Current_Call.First_Node;
--                      Fnode.End_Node := Tmp_Nodenum;
--                      Push(Fnode_Stack, Fnode);
--                      Push(Vstring_Stack, Ident);
                        Nodenum_Stack.Push(Current_Call.Subprogram_Called.Return_Caller, Current_Call.First_Node);
                        if not Nodenum_Stack.Isempty(Current_Call.Subprogram_Called.Return_Callee) then
                           declare
                              Iterator : Nodenum_Stack.Stack := Current_Call.Subprogram_Called.Return_Callee;
                              use Nodenum_Stack;
                           begin
```

```
                           while Iterator /= null loop
                              Add_Arg_Branch(Nodenum_Stack.Top(Iterator), Current_Call.First_Node);
                              Iterator := Iterator.Next;
                           end loop;
                        end;

                     end if;
                     Caller_Nodenum := Current_Call.First_Node;
                     if Is_Debug_Mode then
                        Ada.Text_Io.Put_Line("caller nodenum: " & Nodenum_T'Image(Caller_Nodenum));
                     end if;
                  end if;
               end;
               Call_Stacks.Remove(Call_Stack);
            when others =>
               null;
         end case;

--          declare
--             Exp_Info : Element_Info := Get_Info(Elem);
--          begin
--             if Exp_Info /= Nil_Element_Info then
--                Use_Stacks(Exp_Info.Top);
--             end if;
--          end;

            if Get_Info(Elem) /= Null_Element_Info and then
              Get_Info(Elem) /= Assigned_Element_Info then
               if Is_Debug_Mode then
                  Put_Line("get_info /= null !!");
                  Put_Line(Asis.Text.Element_Image(Elem));
               end if;
               Use_Stacks;
            end if;
         when An_Exception_Handler =>
            Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
            Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
            Add_Branch(Current_Unit.Exception_Point -1, Tmp_Nodenum + 1);

         when A_Path =>
         case Asis.Elements.Path_Kind(elem) is
            when A_Select_Path | An_Or_Path | An_elsif_Path =>
            declare
               SList : Asis.Statement_List := Asis.Statements.Sequence_Of_Statements(Elem);
               Top_Node : Nodenum_T := Get_Info(Slist(Slist'First)).Top;
               Bottom_Node : Nodenum_T := Get_Info(Slist(Slist'Last)).Bottom;
               Next_Node : Nodenum_T := Get_Info(Slist(Slist'Last)).Next;
            begin
               Put_Line("654321");
               case Asis.Elements.Path_Kind ( Elem ) is
                  when A_Select_Path | An_Or_Path =>
                     declare
                        Guard_Item : Asis.Element := Asis.Statements.Guard(Elem);
                     begin
--                      if Guard_Item /= Nil_Element then
                        if not (Asis.Elements.Is_Nil(Guard_Item)) then
                           Top_Node := Get_Info(Guard_Item).Top;
                        end if;
                     end;
                  when An_Elsif_Path =>
                     Top_Node := Get_Info(Elem).Top;
                  when others =>
                     null;
               end case;

               Set_Info(Elem, (Top_Node, Bottom_Node, Next_node));
--             Put("(");
--             Node_Io.Put(Top_Node);
--             Put(", ");
--             Node_Io.Put(Bottom_Node);
--             Put(", ");
--             Node_Io.Put(Next_Node);
--             Put_line(")");
--             Put_Line(Gela_Ids.Debug_Image(Gela_Ids.Create_Id(Slist(Slist'first))));
--             Put_Line(Gela_Ids.Debug_Image(Gela_Ids.Create_Id(Slist(Slist'last))));
--             if Gela_Ids.Is_Equal(Gela_Ids.Create_Id(Slist(Slist'first)), Gela_Ids.Create_Id(Slist(Slist'last))) then
--                Put_Line("same!!!");
--             end if;
--             if Slist(Slist'first) = Slist(Slist'last) then
--                Put_Line("same element!!!");
--             end if;
               end;
            when others =>
               null;
         end case;
            case Asis.Elements.Path_Kind ( Elem ) is
               when An_If_Path =>
                  null;
               when An_Elsif_Path =>
                  null;
               when An_Else_Path =>
                  null;
               when A_Case_Path =>
                  null;
               when A_Select_Path =>
                  null;
               when An_Or_Path =>
                  null;
               when A_Then_Abort_Path =>
                  null;
               when others =>
                  null;
            end case;
         when A_Statement =>
            Use_Stacks;
            Assign_Stacks;
            Use_Arg_Stacks;
            case Asis.Elements.Statement_Kind ( Elem ) is
               when A_Null_Statement =>
```

```
            null;
         when An_If_Statement =>
            declare
               Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
               Self_Node : Nodenum_T := Get_Info(Elem).Top;
            begin

               Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
               Remove_Branch(Self_Node, Self_Node + 1);
               Add_Branch(Self_Node, Tmp_Nodenum);
               for I in Plist'Range loop
                  declare
                     Pkind : Asis.Path_Kinds := Asis.Elements.Path_Kind ( Plist(I) );
                     Pinfo : Element_Info := Get_Info(Plist(I));
                  begin
                     Add_Branch(Self_Node, Pinfo.Top);
                     if Pkind = An_Else_Path then
                        Remove_Branch(Self_Node, Tmp_Nodenum);
                     elsif Pkind = An_Elsif_Path then
                        Remove_Branch(Self_Node, Tmp_Nodenum);
                        Self_Node := Pinfo.Top;
                        Add_Branch(Self_Node, Tmp_Nodenum);
                     end if;
                     if Pinfo.Next /= 0 then
                        Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                        Add_Branch(Pinfo.Bottom, Tmp_Nodenum);
                     end if;
                  end;
               end loop;
               declare
                  Top_Node : Nodenum_T := Get_Info(Elem).Top;
                  Bottom_Node : Nodenum_T := Tmp_Nodenum;
                  Next_Node : Nodenum_T := Num_Of_Nodes + 1;
               begin
                  Set_Info(Elem, (Top_Node, Bottom_Node, Next_Node));
               end;
            end;
         when A_Case_Statement =>
            declare
               Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
               Self_Node : Nodenum_T := Get_Info(Elem).Top;
            begin
               Remove_Branch(Self_Node, Self_Node + 1);
               Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
               for I in Plist'Range loop
                  Add_Branch(Self_Node, Get_Info(Plist(I)).Top);
                  declare
                     Pinfo : Element_Info := Get_Info(Plist(I));
                  begin
                     if Pinfo.Next /= 0 then
                        Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                        Add_Branch(Pinfo.Bottom, Tmp_Nodenum);
                     end if;
                  end;
               end loop;

               declare
                  Top_Node : Nodenum_T := Get_Info(Elem).Top;
                  Bottom_Node : Nodenum_T := Tmp_Nodenum;
                  Next_Node : Nodenum_T := Num_Of_Nodes + 1;
               begin
                  Set_Info(Elem, (Top_Node, Bottom_Node, Next_Node));
               end;

            end;

         when A_Loop_Statement | A_While_Loop_Statement
           | A_For_Loop_Statement =>
            declare
               Self_Node : Nodenum_T := Get_Info(Elem).Top;
            begin
               Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
               Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
               Add_Branch(Tmp_Nodenum, Self_Node);
               if Asis.Elements.Statement_Kind ( Elem )
                 /= A_Loop_Statement then
                  Add_Branch(Self_Node, Tmp_Nodenum + 1);
               end if;
               Set_Info(Elem, (Self_Node, Tmp_Nodenum, Tmp_Nodenum + 1));
               loop
                  Tmp_Nodenum := Pop_Exit_Node(Elem);
                  exit when Tmp_Nodenum = 0;
                  Add_Branch(Tmp_Nodenum, Num_Of_Nodes +1);
--                Node_Io.Put(Tmp_Nodenum);
               end loop;
            end;


         when A_Selective_Accept_Statement =>
            declare
               Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
               Self_Node : Nodenum_T := Get_Info(Elem).Top;
            begin
               Remove_Branch(Self_Node, Self_Node + 1);
               for I in Plist'Range loop
                  declare
                     Slist : Asis.Statement_List := Asis.Statements.Sequence_Of_Statements(Plist(I));
                  begin
                     if Slist'Length = 1 and then
                       Asis.Elements.Statement_Kind(Slist(Slist'First))
                       = A_Terminate_Alternative_Statement then
                        Make_Labeled_Branch(Self_Node, Label_T(Top(Taskhead_Stack)));
                        -- Add_Branch(Self_Node, Num_Of_Nodes +1);
                     else
                        declare
                           Pinfo : Element_Info := Get_Info(Plist(I));
                        begin
                           Add_Branch(Self_Node, Pinfo.Top);
                           if Pinfo.Next /= 0 then
```

```
                  Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                  Add_Branch(Pinfo.Bottom, Num_Of_Nodes +1);
               end if;
            end;
         end if;
      end;
   end loop;
end;

when A_Timed_Entry_Call_Statement | A_Conditional_Entry_Call_Statement =>
   declare
      Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
      Self_Node : Nodenum_T := Get_Info(Elem).Top;
   begin
      Remove_Branch(Self_Node, Self_Node + 1);
      for I in Plist'Range loop
         declare
            Pinfo : Element_Info := Get_Info(Plist(I));
         begin
            Add_Branch(Self_Node, Pinfo.Top);
            if Pinfo.Next /= 0 then
               Remove_Branch(Pinfo.Bottom, Pinfo.Next);
               Add_Branch(Pinfo.Bottom, Num_Of_Nodes +1);
            end if;
         end;
      end loop;
   end;
when An_Asynchronous_Select_Statement =>
   declare
      Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
      Self_Node : Nodenum_T := Get_Info(Elem).Top;
   begin
      Remove_Branch(Self_Node, Self_Node + 1);
      for I in Plist'Range loop
         declare
            Pinfo : Element_Info := Get_Info(Plist(I));
         begin
            if Asis.Elements.Path_Kind(Plist(I)) = A_Select_Path then
               Add_Fork_Branch(Self_Node, Pinfo.Top);
               if Pinfo.Next /= 0 then
                  Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                  Add_Join_Branch(Pinfo.Bottom, Num_Of_Nodes +1);
               end if;

            else
               Add_Branch(Self_Node, Pinfo.Top);
               if Pinfo.Next /= 0 then
                  Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                  Add_Branch(Pinfo.Bottom, Num_Of_Nodes +1);
               end if;
            end if;
         end;
      end loop;
   end;

when An_Assignment_Statement =>
   declare
      Self_Node : Nodenum_T := Get_Info(Elem).Top;
   begin
      Set_Info(Elem, (Self_Node, Num_Of_Nodes, Num_Of_Nodes + 1));
   end;

when A_Block_Statement =>
   if Asis.Statements.Is_Declare_Block( Elem ) then
      Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
      Set_Info(Elem, (Get_Info(Elem).Top, Tmp_Nodenum, Tmp_Nodenum + 1));
      Add_P_Branch_To_Child;
      Add_P_Branch_From_Child;
      Exit_Unit;
   end if;
when An_Accept_Statement =>
   declare
      Plist : Asis.Parameter_Specification_List := Asis.Statements.Accept_Parameters(Elem);
   begin
      if Asis.Statements.Accept_Body_Statements(Elem)
        /= Nil_Element_List or else
        Plist /= Nil_Element_List then
         Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
         Set_Info(Elem, (Get_Info(Elem).top, Tmp_Nodenum, Tmp_Nodenum +1));
         Connect_Labels(Tmp_Nodenum, Label_T(Top(Accepthead_Stack)));
      else
         Tmp_Nodenum := 0;
      end if;
      if Plist /= Nil_Element_List then

         for I in Plist'Range loop
            declare
               DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
            begin
               for J in Dname_List'Range loop

                  case Asis.Elements.Mode_Kind(Plist(I)) is
                     when An_Out_Mode
                       | An_In_Out_Mode =>
                        Use_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Tmp_Nodenum);
                     when others => null;
                  end case;
               end loop;
            end;
         end loop;
      end if;
   end;
   Remove(Accepthead_Stack);
   Exit_Accept(Gela_Ids.Create_Id(Asis.Statements.Corresponding_Entry(Elem)), Tmp_Nodenum);
when A_Procedure_Call_Statement =>
if Asis.Elements.Is_Nil(Asis.Statements.Corresponding_Called_Entity(Elem)) then
   Skip_Process := False;
   goto End_Of_Procedure_Call_Post;
end if;
```

```
                  declare
                     Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
                  begin
                     if Current_Call.Is_Same_Compilation_Unit then
                        if Is_Debug_Mode then
                           Put_Line(To_S(Current_Call.Subprogram_Called.Name));
                        end if;

                        if not Is_Param_Node_Approach then
                           Add_Left_Actual_Label(Current_Call.Subprogram_Called.Args, Current_Call.Call_Branch);
                        end if;

--                      while not Out_Arg_Stacks.Isempty(Current_Call.Out_Arg_Stack) loop
--                         Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                         declare
--                            Self_Node : Nodenum_T := Get_Info(Elem).Top;
--                         begin
--                            Set_Info(Elem, (Self_Node, Tmp_Nodenum, Tmp_Nodenum + 1));
--                         end;
--                         declare
--                            Out_Var_Stack : Varnum_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Stack;
--                            Out_Var_Item_Stack : Var_Item_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Item_Stack;
--                         begin
--                            while not Isempty(Out_Var_Stack) loop
--                               Assign_Variable(Top(Out_Var_Stack),
--                                               Tmp_Nodenum);
--                               Remove(Out_Var_Stack);
--                            end loop;
--                            while not Isempty(Out_Var_Item_Stack) loop
--                               Assign_Variable(Top(Out_Var_Item_Stack).Number,
--                                               Tmp_Nodenum,
--                                               Top(Out_Var_Item_Stack).Is_Normalized,
--                                               Top(Out_Var_Item_Stack).Normalized_Form
--                                               );
--                               Remove(Out_Var_Item_Stack);
--                            end loop;
--                         end;

--                         declare
--                            Arg_Item : Args_Item_Link := Out_Arg_Stacks.Top(Current_Call.Out_Arg_stack).Arg_Item;
--                         begin
--                            if Arg_Item.Formal_Out = 0 then
--                               Nodenum_Stack.Push(Arg_Item.Actual_Out,
--                                                  Tmp_Nodenum);
--                            else
--                               Add_Arg_Branch(Arg_Item.Formal_Out,
--                                              Tmp_Nodenum);
--                            end if;
--                         end;

--                         Out_Arg_Stacks.Remove(Current_Call.Out_Arg_Stack);
--                      end loop;
                     end if;
                  end;
                  Call_Stacks.Remove(Call_Stack);
               <<End_Of_Procedure_Call_Post>>
               when An_Entry_Call_Statement =>
                  declare
                     Entry_Decl : Entry_Item_Link := Entry_Decl_Id_List.Search_Node(Entry_Decl_List, Gela_Ids.Create_Id(Asis.Statements.Corresponding_Called_Entity(Elem)));
                     Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
                  begin
                     if Entry_Decl = null or else
                       Entry_Decl.Task_Link.Decl_Kind = A_Protected_Body_Declaration then
                        if Current_Call.Is_Same_Compilation_Unit then
                           if not Is_Param_Node_Approach then
                              Add_Left_Actual_Label(Current_Call.Subprogram_Called.Args, Current_Call.Call_Branch);
                           end if;

--                         while not Out_Arg_Stacks.Isempty(Current_Call.Out_Arg_Stack) loop
--                            Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                            declare
--                               Self_Node : Nodenum_T := Get_Info(Elem).Top;
--                            begin
--                               Set_Info(Elem, (Self_Node, Tmp_Nodenum, Tmp_Nodenum + 1));
--                            end;
--                            declare
--                               Out_Var_Stack : Varnum_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Stack;
--                               Out_Var_Item_Stack : Var_Item_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Item_Stack;
--                            begin
--                               while not Isempty(Out_Var_Stack) loop
--                                  Assign_Variable(Top(Out_Var_Stack),
--                                                  Tmp_Nodenum);
--                                  Remove(Out_Var_Stack);
--                               end loop;
--                               while not Isempty(Out_Var_Item_Stack) loop
--                                  Assign_Variable(Top(Out_Var_Item_Stack).Number,
--                                                  Tmp_Nodenum,
--                                                  Top(Out_Var_Item_Stack).Is_Normalized,
--                                                  Top(Out_Var_Item_Stack).Normalized_Form);
--                                  Remove(Out_Var_Item_Stack);
--                               end loop;
--                            end;

-- --                         Assign_Variable(Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Name,
-- --                                         Tmp_Nodenum);
--                            declare
--                               Arg_Item : Args_Item_Link := Out_Arg_Stacks.Top(Current_Call.Out_Arg_stack).Arg_Item;
--                            begin
--                               if Arg_Item.Formal_Out = 0 then
--                                  Nodenum_Stack.Push(Arg_Item.Actual_Out,
--                                                     Tmp_Nodenum);
--                               else
--                                  Add_Arg_Branch(Arg_Item.Formal_Out,
--                                                 Tmp_Nodenum);
--                               end if;
--                            end;

--                            Out_Arg_Stacks.Remove(Current_Call.Out_Arg_Stack);
--                         end loop;
```

```
                     end if;
                  else
                     while not Out_Arg_Stacks.Isempty(Current_Call.Out_Arg_Stack) loop
                        declare
                           Tmp_Nextcall : Nextcall := Nextcall_Stacks.Top(Nextcall_Stack);
                        begin
                           Tmp_Nextcall.Arg_Item := Current_Call.Out_Arg_Stack;
                           Nextcall_Stacks.Remove(Nextcall_Stack);
                           Nextcall_Stacks.Push(Nextcall_Stack, Tmp_Nextcall);
                        end;
                        Out_Arg_Stacks.Remove(Current_Call.Out_Arg_Stack);
                     end loop;
                  end if;
               end;

               Call_Stacks.Remove(Call_Stack);

            when A_Requeue_Statement =>
--             Call_Stacks.Remove(Call_Stack);
               null;
            when A_Return_Statement =>
               declare
                  Self_Node : Nodenum_T := Get_Info(Elem).Top;
               begin
                  Set_Info(Elem, (Self_Node, Num_Of_Nodes, Num_Of_Nodes + 1));
               end;
               Remove_Branch(Num_Of_Nodes, Num_Of_Nodes+1);
               Make_Labeled_Branch(Num_Of_Nodes, Label_T(Top(Subprogramhead_Stack)));

            when others =>
               null;
         end case;
--       case Asis.Elements.Statement_Kind ( Elem ) is

       when others =>
         null;
     end case;
exception

--   when Ex : Asis.Exceptions.ASIS_Inappropriate_Context          |
--             Asis.Exceptions.ASIS_Inappropriate_Container        |
--             Asis.Exceptions.ASIS_Inappropriate_Compilation_Unit |
--             Asis.Exceptions.ASIS_Inappropriate_Element          |
 --             Asis.Exceptions.ASIS_Inappropriate_Line             |
 --             Asis.Exceptions.ASIS_Inappropriate_Line_Number      |
 --             Asis.Exceptions.ASIS_Failed                         =>

--      Ada.Wide_Text_IO.Put ("Post_Op : ASIS exception (");

--      Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
--              Ada.Exceptions.Exception_Name (Ex)));
--
--      Ada.Wide_Text_IO.Put (") is raised");
--      Ada.Wide_Text_IO.New_Line;

--      Ada.Wide_Text_IO.Put ("ASIS Error Status is ");

--      Ada.Wide_Text_IO.Put
 --       (Asis.Errors.Error_Kinds'Wide_Image (Asis.Implementation.Status));

 --     Ada.Wide_Text_IO.New_Line;

 --     Ada.Wide_Text_IO.Put ("ASIS Diagnosis is ");
 --     Ada.Wide_Text_IO.New_Line;
 --     Ada.Wide_Text_IO.Put (Asis.Implementation.Diagnosis);
 --     Ada.Wide_Text_IO.New_Line;

 --     Asis.Implementation.Set_Status;

 --  when Ex : others =>
  --    Ada.Wide_Text_IO.Put ("Post_Op : ");

   --    Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
   --            Ada.Exceptions.Exception_Name (Ex)));

 --     Ada.Wide_Text_IO.Put (" is raised (");

   --    Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
    --           Ada.Exceptions.Exception_Information (Ex)));

 --     Ada.Wide_Text_IO.Put (")");
    --    Ada.Wide_Text_IO.New_Line;
    when others =>
       Put_Line("Exception raised in Post_Op");
       Ada.Text_Io.Put_Line("Line : " & Asis.Text.Line_Number'Image(Asis.Text.First_Line_Number(Elem)));
       Put_Line(Asis.Text.Element_Image(Elem));
       Put_Line(Asis.Elements.Debug_Image(Elem));
       raise;
end Post_Op;
```

A.1.4 Ada2DUN Package

```
-- Define-Use Net Generator for Ada 2012 Programs.
----Bo Wang  2014.12

with Asis;
with Asis.Errors;
with Asis.Exceptions;
with Asis.Implementation;
with Asis.Ada_Environments;
with Asis.Compilation_Units;
with Asis.Elements;
with Asis.Iterator;
with Asis.Declarations;
with Asis.Definitions;
```

```
with Asis.Expressions;
with Asis.Statements;
with Asis.Text;
with Gela_Ids;

--with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;
with Ada.Text_IO;
with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;
with Ada.Command_Line;
with Ada.Characters.Handling;
use Ada.Characters.Handling;

with Ada.Calendar;

with Dun_Handler, V_Strings, Stacks;
use Dun_Handler, V_Strings;


procedure Ada2dun is

   package Line_IO is new Integer_IO(Asis.Text.Line_Number);
   package Node_IO is new Integer_IO(Nodenum_T);
   package Int_IO is new Integer_IO(Integer);

   My_Context : Asis.Context;
--   The_Unit : Asis.Compilation_Unit;
   The_Unit_Spec : Asis.Compilation_Unit;
   The_Unit_Body : Asis.Compilation_Unit;
   Current_Statement : Nodenum_T;
--   Args_Nodenum : Nodenum_T;
   My_Args_Item : Args_Item_Link;
   Caller_Nodenum : Nodenum_T := 0; -- call function

   Disc_Range_Flag : Boolean := False; -- ASIS bug

   use Varnum_Stacks;
   use Var_Item_Stacks;
   use Vstring_Stacks;

   function Get_Varnum(Ename : Asis.Expression) return Varnum_T is
      -- Expression
      -- in  : var の Expression
      -- out : var num
      use Asis;
      Temp_Def : Asis.Defining_Name := Nil_Element;
   begin
      if Is_Debug_Mode then
         Put_Line("get_varnum");
         Put_Line(Asis.Text.Element_Image(Ename));
--      Put_Line(Asis.Elements.Debug_Image(Asis.Elements.Enclosing_Element(Ename)));
--      Put_Line(Asis.Elements.Debug_Image(Ename));
      end if;

      declare
      begin

         if Asis.Elements.Element_Kind(Ename) = A_Defining_Name then
            Temp_Def := Ename;
         else
            select
               delay 0.1;
               Put_Line("warning: Corresponding_Name_Definition is canceled.");
            then abort
              Temp_Def := Asis.Expressions.Corresponding_Name_Definition(Ename);
            end select;
         end if;
--       if Temp_Def = Nil_Element then
         if Asis.Elements.Is_Nil(Temp_Def) then
            return 0;
         else
            return Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Temp_Def));
         end if;
      exception
         when Asis.Exceptions.Asis_Inappropriate_Element =>
            Put_Line("warning : raised Asis.Inappropriate_Element in get_varnum!");
         when Asis.Exceptions.Asis_Failed =>
            Put_Line("warning : raised Asis_Failed in get_varnum!");

      end;
      return 0;
   end Get_Varnum;


   function Is_Type_Equal(A, B : Asis.Expression) return Boolean is
     -- ASIS-for-gnat bug
     Result : Boolean := False;
     use Asis;
   begin
--       select
--       delay 0.2;
--       then abort
--       Result := (Asis.Expressions.Corresponding_Expression_Type(A)
--          = Asis.Expressions.Corresponding_Expression_Type(B));
         Result := Asis.Elements.Is_Equal
           (Asis.Expressions.Corresponding_Expression_Type(A),
            Asis.Expressions.Corresponding_Expression_Type(B));
--      end select;
      return Result;
   end Is_Type_Equal;

--    -- expression
--    procedure Analyze_Name(Exp : in Asis.Expression;
--                           Varnum : out Varnum_T;
--                           Is_Normalize : out Boolean;
--                           Normalized_Form : out V_String) is
--    begin
--       case Asis.Elements.Expression_Kind(Exp) is
--       when An_Identifier =>
--       when An_Explicit_Dereference =>
```

```
--       when An_Indexed_Component =>
--       when A_Selected_Component =>
--       when An_Attribute_Reference =>
--       when



   package Nodenum_stacks is new stacks(Nodenum_T);
   use Nodenum_Stacks;
   -- terminate
   taskhead_stack: nodenum_stacks.stack:= null;
   -- return
   Subprogramhead_Stack: Nodenum_Stacks.Stack := null;
   -- requeue
   Accepthead_Stack : Nodenum_Stacks.Stack := null;

   package Fnode_Stacks is new Stacks(Function_Nodes);
   use Fnode_Stacks;
   Fnode_Stack : Fnode_Stacks.Stack := null;

--   package Args_Stacks is new Stacks(Args);
   use Args_Stacks;
--   Args_Stack : Args_Stacks.Stack := null;

   package Call_Stacks is new Stacks(Subprogram_Call);
   use Call_Stacks;

   Call_Stack : Call_Stacks.Stack := null;

      procedure Use_Arg_Stacks is
--       Self_Info : Element_Info := Get_Info(Elem);
--       Self_Node : Nodenum_T;
         Arg_Node_Item : Node_Item_Link;
      begin
--       if Self_Info /= Null_Element_Info then
--          Self_Node := Self_Info.Top;
         while not Nodenum_Stack.Isempty(Arg_Join_Stack) loop
            Arg_Node_Item := Search_Node(Nodenum_Stack.Top(Arg_Join_Stack));
            if Arg_Node_Item.Branch = null then
               Add_Branch(Nodenum_Stack.Top(Arg_Join_Stack), Num_Of_Nodes +1);
            end if;
            Nodenum_Stack.Remove(Arg_Join_Stack);
         end loop;
--       else
--          Nodenum_Stack.Clear(Arg_Join_Stack);
--       end if;
      end Use_Arg_Stacks;

   function Is_Access(Decl : in Asis.Declaration) return Boolean is
      Name_Def : Asis.Defining_Name := Asis.Expressions.Corresponding_Name_Definition(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Decl)));
      Type_Def : Asis.Definition;
      use Asis;
   begin
--       if Name_Def = Nil_Element then
      if Asis.Elements.Is_Part_Of_Implicit(Name_Def) then
         return False;
      end if;
      Type_Def := Asis.Elements.Enclosing_Element(Name_Def);
--       Put_Line(Asis.Text.Element_Image(Type_Def));
--       Put_Line(Asis.Elements.Debug_Image(Asis.Declarations.Type_Declaration_View(Type_Def)));
      if Asis.Elements.Type_Kind(Asis.Declarations.Type_Declaration_View(Type_Def)) = An_Access_Type_Definition then
--       Put_Line("access!");
         return True;
      end if;
      return False;
   end Is_Access;

   procedure Set_Is_Access(Decl : in Asis.Declaration; Is_Access : out Boolean;
                           Type_Ident : in out V_String) is
     Name_Def : Asis.Defining_Name;
     Type_Def, Temp_Def : Asis.Definition;
     Temp_Exp : Asis.Expression;
      use Asis;
   begin
      case Asis.Elements.Declaration_Kind(Decl) is
         when A_Parameter_Specification | A_Formal_Object_Declaration =>
            Temp_Exp := Asis.Declarations.Declaration_Subtype_Mark(Decl);
            if Asis.Elements.Expression_Kind(Temp_Exp) /= An_Identifier then
               Name_Def := Asis.Expressions.Corresponding_Name_Definition(Asis.Expressions.Selector(Temp_Exp));
            else
               Name_Def := Asis.Expressions.Corresponding_Name_Definition(Temp_Exp);
            end if;
         when A_Component_Declaration =>
            Name_Def := Asis.Expressions.Corresponding_Name_Definition(Asis.Definitions.Subtype_Mark(
            Asis.Definitions.Component_Subtype_Indication(Asis.Declarations.Object_Declaration_View(Decl))));
         when others =>
            Temp_Def := Asis.Declarations.Object_Declaration_View(Decl);
            case Asis.Elements.Type_Kind(Temp_Def) is
               when A_Constrained_Array_Definition
                 | An_Unconstrained_Array_Definition =>
                  Name_Def := Asis.Expressions.Corresponding_Name_Definition(Asis.Definitions.Subtype_Mark(
                  Asis.Definitions.Component_Subtype_Indication(Asis.Definitions.Array_Component_Definition(Temp_Def))));
               when others =>
                  Temp_Exp := Asis.Definitions.Subtype_Mark(Temp_Def);
                  if Asis.Elements.Expression_Kind(Temp_Exp) = An_Identifier
                  then
                     Name_Def :=
                       Asis.Expressions.Corresponding_Name_Definition(Temp_Exp);
                  else
                     Name_Def :=
                       Asis.Expressions.Corresponding_Name_Definition(Asis.Expressions.Selector(Temp_Exp));
                  end if;
            end case;
      end case;
      if Is_Debug_Mode then
         Put_Line("set_is_access");
      end if;
--       if Name_Def = Nil_Element then
      if Asis.Elements.Trait_Kind(Decl) = An_Aliased_Trait then
```

```
         Type_Ident := "Aliased-" & Type_Ident;
      end if;
      if Asis.Elements.Is_Part_Of_Implicit(Name_Def) then
         Is_Access := False;
         return;
      end if;
      Type_Def := Asis.Declarations.Type_Declaration_View(Asis.Elements.Enclosing_Element(Name_Def));
      if Is_Debug_Mode then
         Put_Line(Asis.Text.Element_Image(Type_Def));
--       Put_Line(Asis.Elements.Debug_Image(Asis.Declarations.Type_Declaration_View(Type_Def)));
      end if;
      if Asis.Elements.Type_Kind(Type_Def) = An_Access_Type_Definition then

         Is_Access := True;
         case Asis.Elements.Access_Type_Kind(Type_Def) is
            when An_Access_To_Variable | An_Access_To_Constant =>
               Type_Ident := "Aliased-" & To_V(Asis.Expressions.Name_Image(Asis.Definitions.Subtype_Mark(Asis.Definitions.Access_To_Object_Definition(Type_Def))));
            when others =>
               null;
         end case;
      else
         Is_Access := False;
      end if;



   end Set_Is_Access;

   function Get_Type_Ident(Decl : Asis.Declaration) return V_String is
      Temp_Def : Asis.Definition;
      Temp_Exp : Asis.Expression;
      Task_Decl : Asis.Declaration;
      Task_Type : Task_Type_Info_Link;
      Keep_Next : Task_Type_Item_Link;
      use Asis;
   begin
      case Asis.Elements.Declaration_Kind(Decl) is
         when A_Parameter_Specification | A_Formal_Object_Declaration =>
            Temp_Exp := Asis.Declarations.Declaration_Subtype_Mark(Decl);
--       when A_Loop_Parameter_Specification =>
--          Temp_Def := Asis.Declarations.Specification_Subtype_Definition(Decl);
--          Temp_Exp := Asis.Definitions.Subtype_Mark(Temp_Def);
         when others =>
            Temp_Def := Asis.Declarations.Object_Declaration_View(Decl);
            case Asis.Elements.Definition_Kind(Temp_Def) is
               when A_Component_Definition =>
                  Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Component_Subtype_Indication(Temp_Def));
               when A_Type_Definition =>
                  case Asis.Elements.Type_Kind(Temp_Def) is
                     when A_Constrained_Array_Definition
                       | An_Unconstrained_Array_Definition =>
                        Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Component_Subtype_Indication(Asis.Definitions.Array_Component_Definition(Temp_Def)));
                     when others =>

                        Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Component_Subtype_Indication(Temp_Def));
                  end case;
               when others =>

                  Temp_Exp := Asis.Definitions.Subtype_Mark(Temp_Def);
            end case;
      end case;
      if Is_Debug_Mode then
         Put_Line(Asis.Elements.Debug_Image(Temp_Exp));
      end if;
      if Asis.Elements.Expression_Kind(Temp_Exp) /= An_Identifier then
         Temp_Exp := Asis.Expressions.Selector(Temp_Exp);
      end if;
      -- task type
      Task_Decl := Asis.Expressions.Corresponding_Name_Declaration(Temp_Exp);
      if Asis.Elements.Declaration_Kind(Task_Decl) = A_Task_Type_Declaration then
         Task_Type := Task_Type_Id_List.Search_Node(Task_Type_Decl_List, Gela_Ids.Create_Id(Task_Decl));
         if Task_Type /= null then
            Keep_Next := Current_Unit.Decl_By_Task_Type;
            Current_Unit.Decl_By_Task_Type := new Task_Type_Item;
            Current_Unit.Decl_By_Task_Type.Info := Task_Type;
            Current_Unit.Decl_By_Task_Type.Is_Access_Type := False;
            Current_Unit.Decl_By_Task_Type.Next := Keep_Next;
         end if;
      end if;

      return To_V(Asis.Expressions.Name_Image(Temp_Exp));
   end Get_Type_Ident;

   procedure Make_Arg_From_Proc(Decl : Asis.Declaration) is
      Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Decl);
      Position : Natural := 1;
      use Asis;
   begin
      if Plist /= Nil_Element_List then
         for I in Plist'Range loop
            declare
               DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
            begin
               for J in Dname_List'Range loop

                  Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));
                  Position := Position + 1;
               end loop;
            end;
         end loop;
      end if;
   end Make_Arg_From_Proc;


   procedure Push_Exit_Node(Loop_Element_ID : in Gela_Ids.Id;
                            Exit_Node : in Nodenum_T) is
      Tmp_Node : Nodenum_Stack.Stack := Node_Stack_List.Search_Node(Exit_Stack_List, Loop_Element_ID);
   begin
--       if Nodenum_Stack.Isempty(Tmp_Node) then
```

```
      Nodenum_Stack.Push(Tmp_Node, Exit_node);
      Node_Stack_List.Set_Node(Exit_Stack_List, Loop_Element_ID, Tmp_Node);
--      else
--       Nodenum_Stack.Push(Tmp_Node, Exit_node);
--      end if;
   end Push_Exit_Node;

   procedure Push_Exit_Node(Loop_Element : in Asis.Element;
                            Exit_Node : in Nodenum_T) is
   begin
      Push_Exit_Node(Gela_Ids.Create_Id(Loop_Element), Exit_Node);
   end Push_Exit_Node;

   function Pop_Exit_Node(Loop_Element_ID : Gela_Ids.Id) return Nodenum_T is
      Tmp_Node : Nodenum_Stack.Stack := Node_Stack_List.Search_Node(Exit_Stack_List, Loop_Element_ID);
      Tmp_Nodenum : Nodenum_T;
   begin
      if Nodenum_Stack.Isempty(Tmp_Node) then
         return 0;
      else
         Nodenum_Stack.Pop(Tmp_Node, Tmp_Nodenum);
         Node_Stack_List.Set_Node(Exit_Stack_List, Loop_Element_ID, Tmp_Node);
         return Tmp_Nodenum;
      end if;
   end Pop_Exit_Node;

   function Pop_Exit_Node(Loop_Element : Asis.Element) return Nodenum_T is
   begin
      return Pop_Exit_Node(Gela_Ids.Create_ID(Loop_Element));
   end Pop_Exit_Node;


   -- Traverse Control Pre Operation.
   procedure Pre_Op(Elem : in Asis.Element;
                    Control : in out Asis.Traverse_Control;
                    State : in out boolean);

   -- Traverse Control  Post Operation.
   procedure Post_Op(Elem : in Asis.Element;
                     Control : in out Asis.Traverse_Control;
                     State : in out boolean);

   -- Traverse Control の具現化
   procedure Generate_DUN is new Asis.Iterator.Traverse_Element
     (Boolean, Pre_Op, Post_Op);

   procedure Pre_Op(Elem : in Asis.Element;
                    Control : in out Asis.Traverse_Control;
                    State : in out boolean) is
      use Asis;
      procedure Assign_Stacks is
         Self_Node : Nodenum_T := Get_Info(Elem).Top;
      begin
         while not isempty(assign_stack) loop
            assign_variable(top(assign_stack), Self_Node);
            remove(assign_stack);
         end loop;
      end Assign_Stacks;


      procedure Dlist_Add(Dlist : in Asis.Element_List) is
        Tmp_Nodenum  : Nodenum_T;
      begin

         if Dlist /= Nil_Element_List then
            for I in Dlist'Range loop

               if Asis.Elements.Element_Kind(Dlist(I)) = A_Declaration then

                  case Asis.Elements.Declaration_Kind(Dlist(I)) is
                     when A_Variable_Declaration |
                       A_Constant_Declaration |
                       A_Formal_Object_Declaration =>

                        if Is_Debug_Mode then
                           Put_Line(Asis.Text.Element_Image(Dlist(I)));
                        end if;
                        declare
                           Init_Exp : Asis.Expression := Asis.Declarations.Initialization_Expression(Dlist(I));
                           DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Dlist(I));
                           Type_Ident : V_String := Get_Type_Ident(Dlist(I));
--                         Decl_Is_Access : Boolean := Is_Access(Dlist(I));
                           Decl_Is_Access : Boolean;
                        begin

                           if Is_Debug_Mode then
                              Put_Line("pre_is_access");
                           end if;

                           Set_Is_Access(Dlist(I), Decl_Is_Access, Type_Ident);

                           if Is_Debug_Mode then
                              Put_Line("post_is_access");
                           end if;

                           Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Dlist(I))));

                           Set_Info(Dlist(I), (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));


                           Caller_Nodenum := Tmp_Nodenum;

                           for J in Dname_List'Range loop

                              if Is_Debug_Mode then
                                 Put_Line("declared in a block: " &
                                          Asis.Declarations.Defining_Name_Image(Dname_List(J)));
                              end if;
```

```
--                          Assign_Variable(Enter_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J)))), Tmp_Nodenum);

                            Assign_Variable(Enter_Variable(Dname_List(J), Type_Ident, Decl_Is_Access), Tmp_Nodenum);

                            if Is_Debug_Mode and Asis.Elements.Declaration_Kind(Dlist(I)) /= A_Formal_Object_Declaration then
                               Put(Asis.Declarations.Defining_Name_Image(Dname_List(J)));
                               Put(":");
                               Put_Line(Asis.Text.Element_Image(Asis.Declarations.Object_Declaration_View(Dlist(I))));
--                          Put_Line(Asis.Expressions.Name_Image(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Dlist(I)))));
--                          Put_Line(Asis.Elements.Debug_Image(Asis.Declarations.Object_Declaration_View(Dlist(I))));
--                          Put_Line(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names(Asis.Declarations.Object_Declaration_View(Dlist(I)))(1)));
                            end if;
                         end loop;

                      end;
                   when A_Package_Declaration =>

                      Enter_Package_Frame(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names(Dlist(I)) (1))),
                                          Asis.Elements.Declaration_Kind(Dlist(I)));
                      declare
                         Vlist : Asis.Element_List := Asis.Declarations.Visible_Part_Declarative_Items(Dlist(I));
                         Plist : Asis.Element_List := Asis.Declarations.Private_Part_Declarative_Items(Dlist(I));
                      begin

                         Dlist_Add(Vlist);

                         if Asis.Declarations.Is_Private_Present(Dlist(I)) then
                            Dlist_Add(Plist);
                         end if;
                      end;
                      Exit_Package;
                   when A_Package_Body_Declaration |
                     A_Protected_Type_Declaration |
                     A_Single_Protected_Declaration =>

                      Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Dlist(I))));
--                    End_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Dlist(I))));
                      Set_Info(Dlist(I), (Tmp_Nodenum, Tmp_Nodenum +1 , Tmp_Nodenum + 1));
                   when others =>
                      null;
                end case;
             end if;
          end loop;
       end if;
       if Is_Debug_Mode then
          Put_Line("end dlist_add");
       end if;
    end Dlist_Add;


    Tmp_NodeNum : Nodenum_T;
    Info : Element_Info;
    begin -- pre_op
      --Put_Line(Asis.Elements.Debug_Image(Elem));

    case Asis.Elements.Element_Kind(Elem) is
       when Not_An_Element =>
          null;

       when A_Declaration =>
          if Is_Debug_Mode then
             Put_Line("declaration");
             Put_Line(Asis.Elements.Debug_Image(Elem));
          end if;

          case Asis.Elements.Declaration_Kind(Elem) is
             when A_Task_Type_Declaration =>

                Enter_Unit(To_V(Asis.Declarations.Defining_Name_Image
                                (Asis.Declarations.Names (Elem) (1))),
                           A_Task_Type_Declaration);

                Task_Type_Id_List.Set_Node(Task_Type_Decl_List,
                                           Gela_Ids.Create_Id(Elem),
                                           new Task_Type_Info'(Current_Unit, null, null, null));

             when A_Single_Task_Declaration =>

                Enter_Task_Frame(To_V(Asis.Declarations.Defining_Name_Image
                        (Asis.Declarations.Names (Elem) (1))));

             when A_Component_Declaration =>

                Clear(Vstring_Stack);

                Clear(Var_Stack);

                -- protected body
                if Current_Unit.Decl_Kind = A_Protected_Body_Declaration then



                   declare
                      Init_Exp : Asis.Expression := Asis.Declarations.Initialization_Expression(Elem);
                      DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Elem);
                      Type_Ident : V_String := Get_Type_Ident(Elem);
--                    Decl_Is_Access : Boolean := Is_Access(Elem);
                      Decl_Is_Access : Boolean;
                   begin

                      Set_Is_Access(Elem, Decl_Is_Access, Type_Ident);

                      Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));


                      Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
```

```
                        for J in Dname_List'Range loop

--                         declare
--                            Dname : V_String := To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J)));
--                         begin

                              Assign_Variable(Enter_Variable(Dname_List(J),
                                                             Type_Ident,
                                                             Decl_Is_Access),
                                              Tmp_Nodenum);

--                            Use_Variable(Dname, Tmp_Nodenum);

--                         end;

                        end loop;
                     end;
                  end if;

               when A_Variable_Declaration | A_Constant_Declaration =>

                  if Is_Debug_Mode then
                     Put_Line("var/con decl.");
                  end if;
                  declare
                     Self_Info : Element_Info := Get_Info(Elem);
                  begin

                     if Self_Info /= Null_Element_Info then

                        Caller_Nodenum := Get_Info(Elem).Top;
                     end if;

                  end;
--                Put_Line(Asis.Text.Element_Image(Asis.Expressions.Corresponding_Expression_Type(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)))));
--                declare
--                   Exp : Asis.Expression := Asis.Expressions.Corresponding_Expression_Type(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)));
--                   Exp : Asis.Expression := Asis.Expressions.Corresponding_Expression_Type(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)));
--                begin

--                   if Exp = Nil_Element then
--                declare
--                   Hoge : Asis.Element := Asis.Expressions.Corresponding_Name_Definition(Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem)));
--                begin

--                   null;
--                end;
--                      Put_Line(Asis.Text.Element_Image(Asis.Elements.Enclosing_Element(Asis.Expressions.Corresponding_Name_Definition(
Asis.Definitions.Subtype_Mark(Asis.Declarations.Object_Declaration_View(Elem))))));
--                   else
--                      Put_Line(Asis.Text.Element_Image(Exp));
--                   end if;
--                end;

                  Clear(Vstring_Stack);
                  Clear(Var_Stack);
               when A_Deferred_Constant_Declaration =>
                  null;
--             when A_Single_Task_Declaration | A_Protected_Type_Declaration =>
--                null;
               when A_Protected_Body_Declaration =>

                  Enter_Protected(To_V(Asis.Declarations.Defining_Name_Image
                                       (Asis.Declarations.Names (Elem) (1))));

               when A_Procedure_Declaration =>
--                Put_Line("procedure.");
--                Put (Asis.Declarations.Defining_Name_Image
--                       (Asis.Declarations.Names (Elem) (1)));
                  Clear(Args_Stack);
                  Make_Arg_From_Proc(Elem);
                  Enter_Proceduredec(To_V(Asis.Declarations.Defining_Name_Image
                           (Asis.Declarations.Names (Elem) (1))),
                                 Gela_Ids.Create_Id(Elem), Args_Stack);
                  Clear(Args_Stack);

               when A_Function_Declaration =>
                  Clear(Args_Stack);
                  declare
                     Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Elem);
                     Function_Name : V_string := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1)));
                     Tmp_Varnum : Varnum_T;
                     Position : Natural := 1;
                  begin

                     if Plist /= Nil_Element_List then
                        for I in Plist'Range loop
                           declare
                              DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                           begin
                              for J in Dname_List'Range loop

                                 Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));
                                 Position := Position + 1;
                              end loop;
                           end;
                        end loop;
                     end if;
                     Tmp_Varnum := Enter_Variable(Function_Name, To_V(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(Elem))), False);
                     Enter_Functiondec(Function_name,
                                       Gela_Ids.Create_Id(Elem), Args_Stack,
                                       Tmp_Varnum);
                  end;
--                                  To_V(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(Elem))), False);
                  Clear(Args_Stack);

               when A_Parameter_Specification =>
                  null;
               when A_Procedure_Body_Declaration
```

```
                    | A_Function_Body_Declaration
                    | A_Task_Body_Declaration
                    | An_Entry_Body_Declaration =>


                     case Asis.Elements.Declaration_Kind(Elem) is
                        when A_Procedure_Body_Declaration
                          | A_Function_Body_Declaration
                          | An_Entry_Body_Declaration =>


                           Put(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1)));
                           new_line;
                           --Put(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (2)));
                           --new_line;
                           --Put(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (3)));

                           Enter_Unit(To_V(Asis.Declarations.Defining_Name_Image
                                        (Asis.Declarations.Names (Elem) (1))),
                                      Asis.Elements.Declaration_Kind(Elem));

                           First_Stmt;

                           Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                           Node_IO.Put(Tmp_Nodenum);
                           new_line;
                           Node_IO.Put(Tmp_Nodenum+1);
                           new_line;

                           Info:= Element_Info'(Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1);

                           Set_Info(Elem, Info);

                           Push(Subprogramhead_Stack, Tmp_Nodenum);

                           if Asis.Elements.Declaration_Kind(Elem)
                             = An_Entry_Body_Declaration then
                              Set_Info(Asis.Declarations.Entry_Barrier(Elem),
                                   (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));

                           end if;

--                   Put_Line("procedure body.");
--                   New_Line;
--                   Put (Asis.Declarations.Defining_Name_Image
--                           (Asis.Declarations.Names (Elem) (1)));

--                         Args_Nodenum := 0;

                           case Asis.Elements.Declaration_Kind(Elem) is
                              when A_Procedure_Body_Declaration |
                                An_Entry_Body_Declaration =>

                                 if Is_Debug_Mode then
                                    Put_Line("procedure_body");
                                 end if;
                                 declare
                                    Proc_Decl : Asis.Declaration := Nil_Element;
                                    Proc_Item : Subprogram_Item_Link := null;
                                 begin

                                    if Asis.Elements.Declaration_Kind(Elem) =
                                      A_Procedure_Body_Declaration then

                                       Proc_Decl := Asis.Declarations.Corresponding_Declaration(Elem);
--                                     if Proc_Decl /= Nil_Element then
                                       if not (Asis.Elements.Is_Nil(Proc_Decl)) then

                                          Proc_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Proc_Decl));
                                       end if;
                                       if Proc_Item = null then

                                          Clear(Args_Stack);
                                          declare
                                             Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Elem);
                                             Position : Natural := 1;
                                          begin

                                             if Plist /= Nil_Element_List then
                                                for I in Plist'Range loop
                                                   declare
                                                      DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                                                   begin

                                                      for J in Dname_List'Range loop
                                                         Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));

                                                         Position := Position + 1;

                                                      end loop;
                                                   end;
                                                end loop;
                                             end if;

                                          end;

                                          Enter_Proceduredec(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1))),
                                                             Gela_Ids.Create_Id(Elem), Args_Stack);

                                          Clear(Args_Stack);


                                          Proc_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Elem));

                                       end if;
                                    else

                                       Proc_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Search_Entry_Decl(To_V(
                                       Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1)))));
```

```
                        end if;
                        if Is_Debug_Mode then
                           Ada.Text_IO.Put_Line(Nodenum_T'Image(Current_Unit.First_Stmt));
                        end if;

                        Proc_Item.Callee := Current_Unit.First_Stmt;

                        Add_Stacked_Call_Branch(Proc_Item.all);

                        My_Args_Item := Proc_Item.Args;

                        Current_Unit.Subprogram := Proc_Item;

                     end;

                  when A_Function_Body_Declaration =>
                     if Is_Debug_Mode then
                        Put_Line("function body.");
                     end if;
                     declare
                        Func_Decl : Asis.Declaration := Asis.Declarations.Corresponding_Declaration(Elem);
                        Func_Item : Subprogram_Item_Link;
                     begin
--                      if Func_Decl = Nil_Element then
                        if Asis.Elements.Is_Nil(Func_Decl) then
                           if Is_Debug_Mode then
                              Put_Line("no function spec.");
                           end if;
                           Clear(Args_Stack);
                           declare
                              Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Elem);
                              Function_Name : V_string := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1)));
                              Tmp_Varnum : Varnum_T;
                              Position : Natural := 1;
                           begin
                              if Plist /= Nil_Element_List then
                                 for I in Plist'Range loop
                                    declare
                                       DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                                    begin
                                       for J in Dname_List'Range loop
                                          Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));
                                          Position := Position + 1;
                                       end loop;
                                    end;
                                 end loop;
                              end if;
                              declare
                                 Keep_Current_Unit : Unit_Item_Link := Current_Unit;
                              begin
                                 Current_Unit := Current_Unit.Parent;
                                 Tmp_Varnum := Enter_Variable(Function_Name, To_V(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(Elem))), False);
                                 Current_Unit := Keep_Current_Unit;
                              end;

                              Enter_Functiondec(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1))),
                                          Gela_Ids.Create_Id(Elem),
                                          Args_Stack, Tmp_Varnum);
                           end;

--                                        To_V(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(Elem))), False );
                           Clear(Args_Stack);

                           Func_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Elem));
                        else
                           if Is_Debug_Mode then
                              Put_Line("function spec detected.");
                           end if;
                           Func_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Func_Decl));
                        end if;
                        Func_Item.Callee := Current_Unit.First_Stmt;
                        Add_Stacked_Call_Branch(Func_Item.all);
                        My_Args_Item := Func_Item.Args;
                        Current_Unit.Subprogram := Func_Item;
                     end;
                  when others => null;
               end case;
               -- formal-in, out ノードの追加
               declare
                  Decl : Asis.Declaration := Nil_Element;
               begin
                  if Asis.Elements.Declaration_Kind(Elem) /= An_Entry_Body_Declaration then
                     Decl := Asis.Declarations.Corresponding_Declaration(Elem);
                     --Ada.Wide_Text_IO.Put(Decl);
                     --new_line;
                  else
                     Decl := Prot_Entry_Decl_List.Search_Node(Protected_Entry_List, Search_Entry_Decl(To_V(
                     Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1)))));
                  end if;
--                if Decl = Nil_Element then
                  if Asis.Elements.Is_Nil(Decl) then
                     Decl := Elem;
                  end if;
                  declare
                     Plist : Asis.Parameter_Specification_List :=
                       Asis.Declarations.Parameter_Profile(Decl);
                  begin

                     if Plist /= Nil_Element_List then
                        if Is_Param_Node_Approach then
                           Set_Arg_Branch(Tmp_Nodenum);
                        end if;
                        for I in Plist'Range loop
                           --                              Put_Line(Asis.Elements.Debug_Image(Asis.Declarations.Declaration_Subtype_Mark(Plist(I))));
                           declare
                              DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                              Tmp_Varnum : Varnum_T;
                              Type_Ident : V_String := Get_Type_Ident(Plist(I));
```

```
                                    --                                  Decl_Is_Access : Boolean := Is_Access(Plist(I));
                                    Decl_Is_Access : Boolean;
                                 begin
                                    Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                                    for J in Dname_List'Range loop
                                       --                                  Tmp_Varnum := Enter_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))));
                                       Tmp_Varnum := Enter_Variable(Dname_List(J), Type_Ident, Decl_Is_Access);
                                       case Asis.Elements.Mode_Kind(Plist(I)) is
                                          when A_Default_In_Mode | An_In_Mode
                                            | An_In_Out_Mode =>
                                             if Is_Param_Node_Approach then
                                                Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Plist(I))));
                                                Set_Arg_Branch(Tmp_Nodenum);
                                                Add_Top_Arg_Branch(Tmp_Nodenum);
                                                Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
                                                Remove_Branch(Tmp_Nodenum-1, Tmp_Nodenum);
                                                Add_Branch(Current_Unit.First_Stmt, Tmp_Nodenum);
                                                Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                                                Set_Info(Plist(I), (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                                                Assign_Variable(Tmp_Varnum, Tmp_Nodenum);
                                                declare
                                                   Arg_Item : Args_Item_Link := Search_Args_Item(My_Args_Item, To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))));
                                                begin
                                                   if Arg_Item /= null then
                                                      Arg_Item.Formal_In := Tmp_Nodenum;
                                                      Add_Stacked_Arg_Branch_In(Arg_Item.all);
                                                   end if;
                                                end;
                                             else
                                                Add_Formal_In(Tmp_Varnum, Tmp_Nodenum);
                                             end if;
                                          when others =>
                                             null;
                                       end case;
                                       if not Is_Param_Node_Approach then
                                          case Asis.Elements.Mode_Kind(Plist(I)) is
                                             when An_In_Out_Mode | An_Out_Mode =>
                                                Add_Formal_Out(Tmp_Varnum, Tmp_Nodenum);
                                             when others =>
                                                null;
                                          end case;
                                       end if;
                                    end loop;
                                 end;
                              end loop;
                              Use_Arg_Stacks;
                           end if;
                        end;
                     end;



                  when A_Task_Body_Declaration =>
                     Enter_Task(To_V(Asis.Declarations.Defining_Name_Image
                                   (Asis.Declarations.Names (Elem) (1))));
                     First_Stmt;
                     Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                     Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                     Push(Taskhead_Stack, Tmp_Nodenum);
                  when others =>
                     null;

               end case;
               declare
                  Dlist : Asis.Element_List := Asis.Declarations.Body_Declarative_Items(Elem);
               begin
                  Dlist_Add(Dlist);
               end;

--             Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Asis.Declarations.Body_Block_Statement(Elem)))-1); -- for begin statement
               Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Asis.Declarations.Body_Statements(Elem)(1)))-1); -- for begin statement
               Set_Fork_Point;


            when A_Protected_Type_Declaration
              | A_Single_Protected_Declaration =>

               Enter_Protected_Frame(To_V(Asis.Declarations.Defining_Name_Image
                                   (Asis.Declarations.Names (Elem) (1))));
--             First_Stmt;
--             Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--             Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
               declare
                  Top_Node : Nodenum_T := Get_Info(Elem).Top;
               begin
                  Current_Unit.First_Stmt := Top_Node;
                  Remove_Branch(Top_Node, Top_Node+1);
                  Add_Branch(Top_Node, Num_Of_Nodes +1);
               end;


--          when A_Function_Body_Declaration =>
--             null;
            when A_Package_Declaration =>
               Enter_Package_Frame(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1))),
                                   Asis.Elements.Declaration_Kind(Elem));
               if Current_Unit.Parent = Main_Task then
                  declare
                     Vlist : Asis.Element_List := Asis.Declarations.Visible_Part_Declarative_Items(Elem);
                     Plist : Asis.Element_List := Asis.Declarations.Private_Part_Declarative_Items(Elem);
                  begin
                     Dlist_Add(Vlist);
                     if Asis.Declarations.Is_Private_Present(Elem) then
                        Dlist_Add(Plist);
                     end if;
                  end;
                  if Current_Unit.Nodes /= null then
                     Current_Unit.Fork_Point := Num_Of_Nodes;
                  end if;
```

```
            end if;

         when A_Package_Body_Declaration =>
            if Current_Unit /= Main_Task then
               Enter_Package(To_V(Asis.Declarations.Defining_Name_Image
                                  (Asis.Declarations.Names (Elem) (1))));
               First_Stmt;
               declare
                  Top_Node : Nodenum_T := Get_Info(Elem).Top;
                  Bottom_Node : Nodenum_T := Get_Info(Elem).Bottom;
                  Dlist : Asis.Element_List := Asis.Declarations.Body_Declarative_Items(Elem);
                  B_Statements : Asis.Statement_List := Asis.Declarations.Body_Statements(Elem);

               begin
                  Remove_Branch(Top_Node, Bottom_Node);
                  Add_Branch(Top_Node, Num_Of_Nodes +1);
                  Dlist_Add(Dlist);
--                Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Asis.Declarations.Body_Block_Statement(Elem)))-1); -- for begin statement
                  if B_Statements /= Nil_Element_List then
                     Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(B_Statements(1)))-1); -- for begin statement
                     Set_Fork_Point;
                  end if;
               end;
            else
               Enter_Package(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1))));
               Current_Unit.Subprogram := new Subprogram_Item;
               Current_Unit.Subprogram.Name := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1)));
               First_Stmt;
               Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
               Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
               declare
                  Dlist : Asis.Element_List := Asis.Declarations.Body_Declarative_Items(Elem);
               begin
                  Dlist_Add(Dlist);
               end;
               Set_Fork_Point;
            end if;
         when An_Object_Renaming_Declaration =>
            null;
         when A_Package_Renaming_Declaration =>
            null;
         when A_Procedure_Renaming_Declaration =>
            null;
         when A_Function_Renaming_Declaration =>
            null;
         when A_Generic_Package_Renaming_Declaration =>
            null;
         when A_Generic_Procedure_Renaming_Declaration =>
            null;
         when A_Generic_Function_Renaming_Declaration =>
            null;
         when An_Entry_Index_Specification =>
            null;
         when An_Entry_Declaration =>
            Clear(Args_Stack);
            declare
               Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Elem);
            begin
               if Plist /= Nil_Element_List then
                  for I in Plist'Range loop
                     declare
                        DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                        Type_Ident : V_String := Get_Type_Ident(Plist(I));
                        Tmp_Varnum : Varnum_T;
--                      Decl_Is_Access : Boolean := Is_Access(Plist(I));
                        Decl_Is_Access : Boolean;
                        Position : Natural := 1;
                     begin
                        Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                        for J in Dname_List'Range loop
                           -- Put_Line(Asis.Declarations.Defining_Name_Image(Dname_List(J)));
                           if Current_Unit.Decl_Kind /= A_Protected_Body_Declaration then
                              Tmp_Varnum := Enter_Variable(Dname_List(J), Type_Ident, Decl_Is_Access);
                           else
                              -- protected entry declaration
                              -- Asis.Declaration
                              Prot_Entry_Decl_List.Set_Node(Protected_Entry_List, Gela_Ids.Create_Id(Elem), Elem);
                           end if;
                        Push(Args_Stack, Args'(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Asis.Elements.Mode_Kind(Plist(I)), Position));
                        Position := Position + 1;
                        end loop;
                     end;
                  end loop;
               end if;
            end;

            Enter_Entrydec(To_V(Asis.Declarations.Defining_Name_Image
                    (Asis.Declarations.Names (Elem) (1))),
                           Gela_Ids.Create_Id(Elem), Args_Stack);
            Clear(Args_Stack);
         when A_Procedure_Body_Stub =>
            null;
         when A_Function_Body_Stub =>
            null;
         when A_Package_Body_Stub =>
            null;
         when A_Task_Body_Stub =>
            null;
         when A_Protected_Body_Stub =>
            null;
         when An_Exception_Declaration =>
            null;
         when A_Generic_Procedure_Declaration =>
            null;
         when A_Generic_Function_Declaration =>
            null;
         when A_Generic_Package_Declaration =>
            Enter_Package_Frame(To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1))),
                                Asis.Elements.Declaration_Kind(Elem));
```

```
            if Current_Unit.Parent = Main_Task then
               declare
                  Flist : Asis.Element_List := Asis.Declarations.Generic_Formal_Part(Elem);
                  Vlist : Asis.Element_List := Asis.Declarations.Visible_Part_Declarative_Items(Elem);
                  Plist : Asis.Element_List := Asis.Declarations.Private_Part_Declarative_Items(Elem);
               begin
                  if Is_Debug_Mode then
                     Put_Line("*** generic package formal part");
                  end if;
                  Dlist_Add(Flist);
                  if Is_Debug_Mode then
                     Put_Line("*** generic package visible part");
                  end if;
                  Dlist_Add(Vlist);
                  if Asis.Declarations.Is_Private_Present(Elem) then
                     if Is_Debug_Mode then
                        Put_Line("*** generic package private part");
                     end if;
                     Dlist_Add(Plist);
                  end if;
               end;
               if Current_Unit.Nodes /= null then
                  Current_Unit.Fork_Point := Num_Of_Nodes;
               end if;
            end if;

         when A_Package_Instantiation =>
            null;
         when A_Procedure_Instantiation =>
            null;
         when A_Function_Instantiation =>
            null;
         when A_Formal_Procedure_Declaration =>
            null;
         when A_Formal_Function_Declaration =>
            null;
         when A_Formal_Package_Declaration =>
            null;
         when A_Formal_Package_Declaration_With_Box =>
            null;
         when An_Ordinary_Type_Declaration =>
            declare
               Type_Def : Asis.Definition := Asis.Declarations.Type_Declaration_View(Elem);
               Temp_Exp : Asis.Expression;
               Task_Decl : Asis.Declaration;
               Task_Type : Task_Type_Info_Link;
               Keep_Next : Task_Type_Item_Link;
            begin
               if Asis.Elements.Type_Kind(Type_Def) = An_Access_Type_Definition then
                  Temp_Exp := Asis.Definitions.Subtype_Mark(Asis.Definitions.Access_To_Object_Definition(Type_Def));
                  Put_Line(Expression_Kinds'Wide_Image
                           (Asis.Elements.Expression_Kind(Temp_Exp)));
                  if (Asis.Elements.Expression_Kind(Temp_Exp)
                      /= An_Identifier) then
                     Temp_Exp
                       := Asis.Expressions.Selector(Temp_Exp);
                  end if;
                  Task_Decl := Asis.Expressions.Corresponding_Name_Declaration(Temp_Exp);
                  if Asis.Elements.Declaration_Kind(Task_Decl)
                    = A_Task_Type_Declaration then
                     Task_Type := Task_Type_Id_List.Search_Node(Task_Type_Decl_List, Gela_Ids.Create_Id(Task_Decl));
                     if Task_Type /= null then

                        Keep_Next
                          := Current_Unit.Decl_By_Task_Type;
                        Current_Unit.Decl_By_Task_Type := new Task_Type_Item;
                        Current_Unit.Decl_By_Task_Type.Info := Task_Type;
                        Current_Unit.Decl_By_Task_Type.Is_Access_Type := True;
                        Current_Unit.Decl_By_Task_Type.Next := Keep_Next;
                     end if;
                  end if;
               end if;
            end;

         when others =>
            null;
      end case;
   when A_Definition =>
      case Asis.Elements.Definition_Kind ( Elem ) is
         when A_Discrete_Range =>
            if Asis.Elements.Discrete_Range_Kind(Elem) = A_Discrete_Simple_Expression_Range then
               Disc_Range_Flag := True; -- ASIS bug
            end if;
         when others =>
            null;
      end case;

   when An_Expression =>
      case Asis.Elements.Expression_Kind ( Elem ) is
         when An_Identifier =>
            if Get_Info(Elem) /= Assigned_Element_Info then
--             Push(Vstring_Stack, To_V(Asis.Expressions.Name_Image(Elem)));
               declare
                  Tmp_Var : Varnum_T := Get_Varnum(Elem);
                  Parent_Exp : Asis.Element;
               begin
                  if Is_Debug_Mode then
                     Ada.Text_Io.Put_Line("get_varnum: " & Varnum_T'Image(Tmp_Var));
                  end if;
                  if Tmp_Var /= 0 then
                     Parent_Exp := Elem;
                     if (not Disc_Range_Flag) and then Asis.Elements.Element_Kind(Asis.Elements.Enclosing_Element(Parent_Exp)) = An_Expression then
                        declare
                           Iter : Asis.Expression := Asis.Elements.Enclosing_Element(Parent_Exp);
                           Target : Asis.Expression;
                           Norm_Stack : Vstring_Stacks.Stack := null;
                           Norm_Form : V_String := Null_Str;
                           Isnot_First : Boolean := False;
                           Is_Norm : Boolean := False;
```

```
                        begin
--                         Push(Norm_Stack, To_V(Asis.Expressions.Name_Image(Elem)));
                           Iter_Loop:
                           while Asis.Elements.Element_Kind(Iter) = An_Expression loop
                              if Is_Debug_Mode then
                                 Put_Line("iter: " & Asis.Text.Element_Image(Iter));
                              end if;
                              case Asis.Elements.Expression_Kind(Iter) is
                                 when A_Selected_Component =>
                                    Target := Asis.Expressions.Selector(Iter);
                                    if Is_Debug_Mode then
                                       Put_Line(Asis.Text.Element_Image(Target));
                                    end if;
--                                  if Target = Elem then
                                    if Asis.Elements.Is_Equal
                                      (Target,Elem) then
                                       exit Iter_Loop;
--                                  elsif Asis.Expressions.Corresponding_Expression_Type(Target) /= Asis.Expressions.Corresponding_Expression_Type(Elem) then
                                    elsif not Is_Type_Equal(Target, Elem) then
                                       if Is_Debug_Mode then
                                          Put_Line("pushed " & Asis.Expressions.Name_Image(Target));
                                       end if;
                                       Push(Norm_Stack, To_V(Asis.Expressions.Name_Image(Target)));
                                    else

                                       Is_Norm := True;
                                    end if;

                                    Parent_Exp := Iter;

                                    Iter := Asis.Elements.Enclosing_Element(Iter);

                                 when An_Explicit_Dereference =>
                                    exit Iter_Loop;
                                 when An_Identifier =>

                                    exit Iter_Loop;
                                 when An_Indexed_Component =>
--                                  if Asis.Expressions.Prefix(Iter) = Parent_Exp then
                                    if Asis.Elements.Is_Equal
                                      (Asis.Expressions.Prefix(Iter),
                                       Parent_Exp) then
                                       Parent_Exp := Iter;
                                    end if;
                                    exit Iter_Loop;
                                 when others =>
                                    exit Iter_Loop;
                              end case;

                           end loop Iter_Loop;
                           if Is_Debug_Mode then
                              Put_Line("iter loop exited.");
                           end if;
                           if not Isempty(Norm_Stack) then
                              while not Isempty(Norm_Stack) loop
                                 if Isnot_First then
                                    Norm_Form := Top(Norm_Stack) & "." & Norm_Form;
                                 else
                                    Isnot_First := True;
                                    Norm_Form := Top(Norm_Stack);
                                 end if;
                                 Remove(Norm_Stack);
                              end loop;
                              Norm_Form := To_V(Asis.Expressions.Name_Image(Elem)) & "." & Norm_Form;
                              if Get_Info(Parent_Exp) /= Assigned_Element_Info then
--                                 Put_Line("case norm:use");
                                    Push(Var_Item_Stack, (Tmp_Var, Is_Norm, Norm_Form, null));
                              else
--                                 Put_Line("case norm:assign");
                                    Push(A_Var_Item_Stack, (Tmp_Var, Is_Norm, Norm_Form, null));
                              end if;
                           else
                              if Get_Info(Parent_Exp) /= Assigned_Element_Info then
--                                 Put_Line("case :use");
                                    Push(Var_Stack, Tmp_Var);
                              else
--                                 Put_Line("case :assign");
                                    Push(A_Var_Stack, Tmp_Var);
                              end if;
                           end if;
                        end;
                     else
                        Push(Var_Stack, Tmp_Var);
                     end if;
                  end if;
               end;

--             use_variable(To_V(Asis.Expressions.Name_Image(Elem)), Num_Of_Nodes);
            end if;
         when A_Function_Call =>
            if Is_Debug_Mode then
               Put_line("function call : ");
               Put_Line(Asis.Text.Element_Image(Elem));
            end if;

            declare
               Called_Entity : Asis.Declaration;
               Subprog : Subprogram_Item_Link;
               New_Call_Branch : Call_Branch_Item_Link := null;
            begin
               begin
                  Put_Line("123");
                  --Asis.Expressions.Prefix(Elem);
                  Called_Entity := Asis.Elements.Enclosing_Element(Asis.Expressions.Corresponding_Name_Definition(
                  Asis.Expressions.Selector(Asis.Expressions.Prefix(Elem))));
                  Put_Line("124");
                  -- ASIS bug
               exception
                  when others =>
```

```
                        Called_Entity := Asis.Expressions.Corresponding_Called_Function(Elem);
                     end;
                     Subprog := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_entity));
--                   if Called_Entity /= Nil_Element and then Asis.Elements.Enclosing_Compilation_Unit(Called_Entity) = The_Unit then
                     if Subprog /= null then
--                      Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                      Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                        Set_Call_Tree(Current_Routine.Subprogram, Subprog, Num_Of_Nodes);

                        if Subprog.Callee = 0 then
                           New_Call_Branch :=
                             Add_Call_Branch(Caller_Nodenum, 0);
                           Call_Branch_Stack.Push(Subprog.Caller,
                                                  New_Call_Branch);
                        else
                           New_Call_Branch :=
                             Add_Call_Branch(Caller_Nodenum,
                                             Subprog.Callee);
                        end if;
                        if Is_Debug_Mode then
                           Ada.Text_Io.Put_Line("caller nodenum: " & Nodenum_T'Image(Caller_Nodenum));
                        end if;
                        Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Subprog.Args, A_Function_Call, True, False, Caller_Nodenum, null, New_Call_Branch));
                     else
                        Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, A_Function_Call, False, False, 0, null, null));
                     end if;

                  end;
--                   end if;

               when An_Allocation_From_Subtype =>
                  declare
                     Task_Decl : Asis.Declaration := Asis.Expressions.Corresponding_Name_Declaration(Asis.Definitions.Subtype_Mark(
                     Asis.Expressions.Allocator_Subtype_Indication(Elem)));
                     Task_Type : Task_Type_Info_Link;
                  begin
                     if Asis.Elements.Declaration_Kind(Task_Decl) = A_Task_Type_Declaration then
                        Task_Type := Task_Type_Id_List.Search_Node(Task_Type_Decl_List, Gela_Ids.Create_Id(Task_Decl));
                        if Task_Type /= null  then
                           Add_P_Branch_From_Allocate(Task_Type);
                        end if;
                     end if;
                  end;
               when others =>
                  null;
            end case;


         when An_Association =>
            case Asis.Elements.Association_Kind ( Elem ) is
               when A_Parameter_Association =>
                  declare
                     Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
                     Param_Name : V_String;
                     Temp_Exp : Asis.Expression;
                     Arg_Item : Args_Item_Link;
                  begin
                     if Current_Call.Is_Same_Compilation_Unit then
                        if Asis.Expressions.Is_Normalized(Elem) then
                           Param_Name := To_V(Asis.Declarations.Defining_Name_Image(Asis.Expressions.Formal_Parameter(Elem)));
                           Arg_Item := Search_Args_Item(Current_Call.Args, Param_Name);
                        else
                           Temp_Exp := Asis.Expressions.Formal_Parameter(Elem);
--                         if Temp_Exp = Nil_Element then
                           if Asis.Elements.Is_Nil(Temp_Exp) then
                              Arg_Item := Current_Call.Args;
                           else
                              Arg_Item := Search_Args_Item(Current_Call.Args, To_V(Asis.Expressions.Name_Image(Temp_Exp)));
                           end if;
                        end if;
                        if Is_Param_Node_Approach then
                           if Arg_Item.Mode /= An_Out_Mode then
                              if Current_Call.Callkind /= An_Entry_Call then
                                 Set_Arg_Branch(Current_Call.First_Node);
                                 Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                                 Remove_Branch(Tmp_Nodenum-1, Tmp_Nodenum);
                                 Add_Branch(Current_Call.First_Node, Tmp_Nodenum);
                                 if Arg_Item.Mode = A_Default_In_Mode or Arg_Item.Mode = An_In_Mode then
                                    Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                                    Set_Arg_Branch(Tmp_Nodenum);
                                    if Is_Debug_Mode then
                                       Ada.Text_Io.Put_Line("=== arg_join_stack : " & Nodenum_T'Image(Tmp_Nodenum));
                                    end if;
                                    Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                                 end if;
                                 Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                                 Caller_Nodenum := Tmp_Nodenum;
                                 if Is_Debug_Mode then
                                    Ada.Text_Io.Put_Line("caller nodenum: " & Nodenum_T'Image(Caller_Nodenum));
                                 end if;

                                 if Arg_Item.Formal_In = 0 then
                                    Nodenum_Stack.Push(Arg_Item.Actual_In,
                                                       Tmp_Nodenum);
                                 else
                                    Add_Arg_Branch(Tmp_Nodenum,
                                                   Arg_Item.Formal_In);
                                 end if;
                              end if;
                           end if;
                        end if;
                     end if;
                  end;
               when others =>
                  null;
            end case;

         when A_Statement =>
            if Is_Debug_Mode then
```

```
            Put_Line(Asis.Elements.Debug_Image(Elem));
            Put_Line(Asis.Text.Element_Image(Elem));
         end if;

         if Asis.Elements.Statement_Kind(Elem) /= A_Terminate_Alternative_Statement then

            Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
            Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
            Caller_Nodenum := Tmp_Nodenum;
            if Is_Debug_Mode then
               Ada.Text_Io.Put_Line("caller nodenum: " & Nodenum_T'Image(Caller_Nodenum));
            end if;
            Assign_Stacks;

--          Put("(");
--          Node_Io.Put(tmp_Nodenum);
--          Put(", ");
--          Node_Io.Put(tmp_Nodenum);
--          Put(", ");
--          Node_Io.Put(Tmp_Nodenum+1);
--          Put_line(")");
         end if;

         Clear(Vstring_Stack);
         Clear(Var_Stack);
         Clear(A_Var_Stack);

         Current_Statement := Tmp_Nodenum;
         declare
            package Statement_IO is new Enumeration_IO(Asis.Statement_Kinds);
            package Declaration_IO is new Enumeration_IO(Asis.Declaration_Kinds);
            package Expression_IO is new Enumeration_IO(Asis.Expression_Kinds);
            use Statement_IO;
            use Declaration_IO;
            use Expression_IO;
         begin
            case Asis.Elements.Statement_Kind ( Elem ) is
               when A_Null_Statement =>
                  null;
               when An_Assignment_Statement =>
                  --           Line_IO.Put(Asis.Text.First_Line_Number(Elem));
                  --           Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                  --           Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
--                Put(Asis.Elements.Expression_Kind(Asis.Expressions.Prefix(Asis.Statements.Assignment_Variable_Name(Elem))));
--                New_Line;
--                if Asis.Elements.Expression_Kind(Asis.Statements.Assignment_Variable_Name(Elem)) = A_Selected_Component then
--                   Put_Line(Asis.Text.Element_Image(Asis.Expressions.Prefix(Asis.Statements.Assignment_Variable_Name(Elem))));
--                   Put(Asis.Elements.Expression_Kind(Asis.Expressions.Prefix(Asis.Statements.Assignment_Variable_Name(Elem))));
--                   New_Line;
--                end if;
--                Put_Line("test name_definition:");
--                Put_Line(Asis.Text.Element_Image(Asis.Expressions.Corresponding_Name_Definition(Asis.Expressions.Selector(Asis.Statements.Assignment_Variable_Name(Elem)))));
--                Put_Line("test name_declaration:");
--                Put_Line(Asis.Text.Element_Image(Expressions.Corresponding_Name_Declaration(Asis.Expressions.Selector(Asis.Statements.Assignment_Variable_Name(Elem)))));
                  declare
                     Assign_Var : Asis.Expression := Asis.Statements.Assignment_Variable_Name(Elem);
                  begin
                     Set_Info(Assign_Var, Assigned_Element_Info);
                     if Asis.Elements.Expression_Kind(Assign_Var) = A_Selected_Component then
                        Assign_Var := Asis.Expressions.Selector(Assign_Var);
                     end if;
                     declare
                        Tmp_Var : Varnum_T := Get_Varnum(Assign_Var);
                     begin
                        if Is_Debug_Mode then
                           Ada.Text_Io.Put_Line("get_varnum: " & Varnum_T'Image(Tmp_Var));
                        end if;
                        if Tmp_Var /= 0 then
                           Assign_Variable(Get_Varnum(Assign_Var), Tmp_Nodenum);
                        end if;
                     end;
--                   Put_Line(Asis.Text.Element_Image(Asis.Expressions.Corresponding_Expression_Type(Assign_Var)));
--                   Assign_Variable(To_V(Asis.Expressions.Name_Image(Asis.Statements.Assignment_Variable_Name(Elem))), Tmp_Nodenum);

--                   Set_Info(Asis.Statements.Assignment_Variable_Name(Elem), Assigned_Element_Info);
                  end;
               when An_If_Statement =>
                  null;
                  --           Push(Ifnum_Stack, Tmp_Nodenum);
--                declare
--                   Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
--                begin
--                   for I in Plist'Range loop
--                      declare
--                         Pkind : Asis.Element_Kinds := Asis.Elements.Path_Kind(Plist(I));
--                         Cond : Asis.Expression;
--                      begin
--                      if Pkind = An_If_Path or Pkind = An_Elsif_Path then
--                         Cond := Condition_Expression(pkind

               when A_Case_Statement =>
                  null;

               when A_Loop_Statement =>
                  null;

               when A_While_Loop_Statement =>
                  Set_Info(Asis.Statements.While_Condition(Elem), Get_Info(Elem));

               when A_For_Loop_Statement =>
                  declare
                     Param : Asis.Declaration := Asis.Statements.For_Loop_Parameter_Specification(Elem);
                     Type_Ident : V_String := To_V(String'("For_Loop_Param"));
                     Tmp_Varnum : Varnum_T;
                  begin
--                   Type_Ident := Get_Type_Ident(Param);
                     if Is_Debug_Mode then
                        Put_Line("for_loop");
                        Put_Line(Asis.Text.Element_Image(Param));
```

```
                        end if;
                        Tmp_Varnum := Enter_Variable(Asis.Declarations.Names(Param)(1), Type_Ident, False);
                        Assign_Variable(Tmp_Varnum, Tmp_Nodenum);
                        Use_Variable(Tmp_Varnum, Tmp_Nodenum);
                     end;

                  when A_Block_Statement =>
                     if Asis.Statements.Is_Declare_Block( Elem ) then
                        declare
                           Block_Name : Asis.Defining_Name := Asis.Statements.Statement_Identifier(Elem);
                           Dlist : Asis.Declarative_Item_List := Asis.Statements.Block_Declarative_Items(Elem);
                        begin
--                         if Block_Name = Nil_Element then
                           if Asis.Elements.Is_Nil(Block_Name) then
                              Enter_Unit(To_V(Unique_Identifier), Not_A_Declaration);
                           else
                              Enter_Unit(To_V(Asis.Declarations.Defining_Name_Image(Block_Name)), Not_A_Declaration);
                           end if;
                           First_Stmt;
                           Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                           Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                           Dlist_Add(Dlist);
                        end;
                        Set_Fork_Point;
                     end if;

                  when An_Exit_Statement =>
--                   declare
--                      Loop_Name : Asis.Expression := Asis.Statements.Exit_Loop_Name(Elem);
--                      Iter : Asis.Element;
--                   begin
--                      if Loop_Name = Nil_Element then
                           Push_Exit_Node(Asis.Statements.Corresponding_Loop_Exited(Elem), Tmp_Nodenum);
--                      else
--                         if Asis.Elements.Expression_Kind(Loop_Name) =
--                           A_Selected_Component then
--                            Loop_Name := Asis.Expressions.Selector(Loop_Name);
--                         end if;
--                         Iter := Asis.Elements.Enclosing_Element(Elem);
--                         while Iter /= Nil_Element loop
--                            if Asis.Elements.Element_Kind(Iter) = A_Statement and then
--                              (Asis.Elements.Statement_Kind(Iter) = A_Loop_Statement or Asis.Elements.Statement_Kind(Iter) = A_For_Loop_Statement
--                               or Asis.Elements.Statement_Kind(Iter) = A_While_Loop_Statement or Asis.Elements.Statement_Kind(Iter) = A_Block_Statement) and then
--                              Equal_Insensitive(To_V(Asis.Expressions.Name_Image(Loop_Name)), To_V(Asis.Declarations.Defining_Name_Image(
Asis.Statements.Statement_Identifier(Iter)))) then
--                                 Push_Exit_Node(Iter, Tmp_Nodenum);
--                               end if;
--                               Iter := Asis.Elements.Enclosing_Element(Iter);
--                            end loop;
--                      end if;
--                   end;

                     --           Put_Line("done(corresponding_loop_exited)");
--                   if Asis.Statements.Exit_Condition(Elem) = Nil_Element then
                     if Asis.Elements.Is_Nil
                       (Asis.Statements.Exit_Condition(Elem)) then
                        Remove_Branch(Tmp_Nodenum, Tmp_Nodenum+1);
                        Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, 0));
                     end if;

                  when A_Goto_Statement =>
                     null;

                  when A_Procedure_Call_Statement =>
                     --           Line_IO.Put(Asis.Text.First_Line_Number(Elem));

                     declare
                        Called_Entity : Asis.Declaration := Asis.Statements.Corresponding_Called_Entity(Elem);
                        Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
                     begin
--                      if Called_Entity /= Nil_Element
--                        and then Asis.Elements.Enclosing_Compilation_Unit(Called_Entity) = The_Unit
--                        and then Asis.Elements.Declaration_Kind(Called_Entity) /= Not_A_Declaration then
                        if Subprog /= null then
                           if Is_Debug_Mode then
                              Put("procedure call : ");
                              Put_Line(Asis.Text.Element_Image(Called_Entity));
                           end if;
                           Set_Call_Tree(Current_Routine.Subprogram, Subprog, Tmp_Nodenum);
                           declare
--                            Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
                              New_Call_Branch : Call_Branch_Item_Link;
                           begin
                              if Subprog.Callee = 0 then
                                 New_Call_Branch :=
                                   Add_Call_Branch(Tmp_Nodenum, 0);
                                 Call_Branch_Stack.Push(Subprog.Caller,
                                                        New_Call_Branch);
                              else
                                 New_Call_Branch := Add_Call_Branch
                                   (Tmp_Nodenum, Subprog.Callee);
                              end if;

                              Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Subprog.Args, A_Procedure_Call, True, True, Tmp_Nodenum, null, New_Call_Branch));
                           end;
                        else
--                         if Called_Entity /= Nil_Element and then Asis.Elements.Declaration_Kind(Called_Entity) /= A_Procedure_Instantiation then
                           if not (Asis.Elements.Is_Nil(Called_Entity)) and then Asis.Elements.Declaration_Kind(Called_Entity) /= A_Procedure_Instantiation then
                              Clear(Args_Stack);
                              Make_Arg_From_Proc(Called_Entity);
                              declare
                                 Base_Args, New_Args : Args_Item_Link := null;
                              begin
                                 while not Isempty(Args_Stack) loop
                                    New_Args := new Args_Item;
                                    New_Args.Name := Top(Args_Stack).Name;
                                    New_Args.Mode := Top(Args_Stack).Mode;
                                    New_Args.Position := Top(Args_Stack).Position;
                                    New_Args.Next := Base_Args;
```

```
                        Base_Args := New_Args;
                        Remove(Args_Stack);
                     end loop;
                     Subprog := new Subprogram_Item;
                     Subprog.Args := Base_Args;
                     Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Base_Args, A_Procedure_Call, False, False, 0, null, null));
                     Clear(Args_Stack);
                  end;
               else
                  Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, A_Procedure_Call, False, False, 0, null, null));
               end if;

            end if;
         exception
            when others =>
               Put_Line("exception raised.");
               Put_Line(Asis.Elements.Debug_Image(Elem));
               Put(Asis.Elements.Declaration_Kind ( Called_Entity ));
               New_Line;
         end;

      when An_Entry_Call_Statement =>
         if Is_Debug_Mode then
            Put_Line(Asis.Text.Element_Image(Asis.Statements.Corresponding_Called_Entity(Elem)));
         end if;
         declare
            Entry_Decl : Entry_Item_Link := Entry_Decl_Id_List.Search_Node(Entry_Decl_List, Gela_Ids.Create_Id(Asis.Statements.Corresponding_Called_Entity(Elem)));
         begin
            if Entry_Decl /= null then -- ASIS bug
               if Entry_Decl.Task_Link.Decl_Kind /= A_Protected_Body_Declaration then
                  Enter_Entrycall(Entry_Decl, Tmp_Nodenum);
                  Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, Entry_Decl.Args, An_Entry_Call, False, False, 0, null, null));
               else
                  declare
                     Called_Entity : Asis.Declaration := Asis.Statements.Corresponding_Called_Entity(Elem);
                     Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
                     New_Call_Branch : Call_Branch_Item_Link
                       := null;
                  begin
                     if Subprog /= null then
                        if Is_Debug_Mode then
                           Put("entry body call : ");
                           Put_Line(Asis.Text.Element_Image(Called_Entity));
                        end if;
                        Set_Call_Tree(Current_Routine.Subprogram, Subprog, Tmp_Nodenum);

                        if Subprog.Callee = 0 then
                           New_Call_Branch :=
                             Add_Call_Branch(Tmp_Nodenum, 0);
                           Call_Branch_Stack.Push(Subprog.Caller,
                                                  New_Call_Branch);
                        else
                           New_Call_Branch := Add_Call_Branch
                             (Tmp_Nodenum, Subprog.Callee);
                        end if;

                        Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Subprog.Args, A_Protected_Entry_Call, True, True, Tmp_Nodenum, null, New_Call_Branch));
                     else
--                      if Called_Entity /= Nil_Element then
                        if not (Asis.Elements.Is_Nil(Called_Entity)) then
                           Clear(Args_Stack);
                           Make_Arg_From_Proc(Called_Entity);
                           declare
                              Base_Args, New_Args : Args_Item_Link := null;
                           begin
                              while not Isempty(Args_Stack) loop
                                 New_Args := new Args_Item;
                                 New_Args.Name := Top(Args_Stack).Name;
                                 New_Args.Mode := Top(Args_Stack).Mode;
                                 New_Args.Position := Top(Args_Stack).Position;
                                 New_Args.Next := Base_Args;
                                 Base_Args := New_Args;
                                 Remove(Args_Stack);
                              end loop;
                              Subprog := new Subprogram_Item;
                              Subprog.Args := Base_Args;
                              Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Base_Args, A_Protected_Entry_Call, False, False, 0, null, null));
                              Clear(Args_Stack);
                           end;
                        else
                           Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, An_Entry_Call, False, False, 0, null, null));
                        end if;
                     end if;
                  exception
                     when others =>
                        Put_Line("exception raised.");
                        Put_Line(Asis.Elements.Debug_Image(Elem));
                        Put(Asis.Elements.Declaration_Kind ( Called_Entity ));
                        New_Line;
                  end;

               end if;
            else
               declare
                  Called_Entity : Asis.Declaration := Asis.Statements.Corresponding_Called_Entity(Elem);
                  Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
               begin
                  if Subprog /= null then
                     if Is_Debug_Mode then
                        Put("procedure call : ");
                        Put_Line(Asis.Text.Element_Image(Called_Entity));
                     end if;

                     Set_Call_Tree(Current_Routine.Subprogram, Subprog, Tmp_Nodenum);
                     declare
                        New_Call_Branch : Call_Branch_Item_Link;
                     begin
                        if Subprog.Callee = 0 then
                           New_Call_Branch :=
```

```
                        Add_Call_Branch(Tmp_Nodenum, 0);
                     Call_Branch_Stack.Push(Subprog.Caller,
                                            New_Call_Branch);
                  else
                     New_Call_Branch :=
                       Add_Call_Branch(Tmp_Nodenum, Subprog.Callee);
                  end if;
                  Call_Stacks.Push(Call_Stack, Subprogram_Call'(Subprog, Subprog.Args, A_Procedure_Call, True, True, Tmp_Nodenum, null, New_Call_Branch));
               end;

            else
                  Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, A_Procedure_Call, False, False, 0, null, null));
            end if;

         exception
            when others =>
               Put_Line("exception raised.");
               Put_Line(Asis.Elements.Debug_Image(Elem));
               Put(Asis.Elements.Declaration_Kind ( Called_Entity ));
               New_Line;
         end;
      end if;
   end;

when A_Return_Statement =>
   if Current_Unit.Decl_Kind = A_Function_Body_Declaration then
      Nodenum_Stack.Push(Current_Unit.Subprogram.Return_Callee, Tmp_nodenum);
      if not Nodenum_Stack.Isempty(Current_Unit.Subprogram.Return_Caller) then
         declare
            Iterator : Nodenum_Stack.Stack := Current_Unit.Subprogram.Return_Caller;
            use Nodenum_Stack;
         begin
            while Iterator /= null loop

               Add_Arg_Branch(Tmp_Nodenum, Nodenum_Stack.Top(Iterator));

               Iterator := Iterator.Next;
            end loop;
         end;
      end if;
      if not Is_Param_Node_Approach then
         Add_Arg_Branch(Tmp_Nodenum, Tmp_Nodenum);
      end if;
   end if;


when An_Accept_Statement =>
   Enter_Accept(Gela_Ids.Create_Id(Asis.Statements.Corresponding_Entry(Elem)), Tmp_Nodenum);
   declare
      Plist : Asis.Parameter_Specification_List := Asis.Statements.Accept_Parameters(Elem);
   begin
      if Plist /= Nil_Element_List then
         for I in Plist'Range loop
            declare
               DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
            begin
               for J in Dname_List'Range loop

                  case Asis.Elements.Mode_Kind(Plist(I)) is
                     when A_Default_In_Mode | An_In_Mode
                       | An_In_Out_Mode =>
                          Assign_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Tmp_Nodenum);
                     when others => null;
                  end case;
               end loop;
            end;
         end loop;
      end if;
   end;
   Push(Accepthead_Stack,Tmp_Nodenum);
when A_Requeue_Statement |
  A_Requeue_Statement_With_Abort =>

   declare
      Entry_Decl : Entry_Item_Link := Entry_Decl_Id_List.Search_Node(Entry_Decl_List, Gela_Ids.Create_Id(
      Asis.Expressions.Corresponding_Name_Declaration(Asis.Statements.Requeue_Entry_Name(Elem))));
      Encl_Entry : Asis.Element := Asis.Elements.Enclosing_Element(Elem);
      Parent_Decl : Asis.Declaration;
      Keep_Unit : Unit_Item_Link := Current_Unit;
      Keep_Nodenum : Nodenum_T := Tmp_Nodenum;
   begin
      if Is_Debug_Mode then
         Put_Line("requeue statement");
      end if;

      if Asis.Elements.Statement_Kind(Elem) =
        A_Requeue_Statement_With_Abort then
         Set_Nondet_Branch(Tmp_Nodenum);
         Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
         Set_Info(Elem, (Keep_Nodenum, Tmp_Nodenum, Tmp_Nodenum +1));
      end if;
      Iter_Encl:
      loop
         case Asis.Elements.Element_Kind(Encl_Entry) is
            when A_Declaration =>
               if Asis.Elements.Declaration_Kind(Encl_Entry)
                 = An_Entry_Body_Declaration then
                  Parent_Decl := Encl_Entry;
--                Plist := Asis.Declarations.Parameter_Profile(Encl_Entry);
                  exit Iter_Encl;
               end if;

            when A_Statement =>
               if Asis.Elements.Statement_Kind(Encl_Entry)
                 = An_Accept_Statement then
                  Parent_Decl := Asis.Statements.Corresponding_Entry(Encl_Entry);
--                Plist := Asis.Statements.Accept_Parameters(Encl_Entry);
                  exit Iter_Encl;
               end if;
```

```
            when others =>
               null;
         end case;
         Encl_Entry := Asis.Elements.Enclosing_Element(Encl_Entry);
      end loop Iter_Encl;
      if Is_Debug_Mode then
         Put_Line(Asis.Text.Element_Image(Parent_Decl));
      end if;
      declare
         Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Parent_Decl);
         Parent_Decl_Spec : Asis.Declaration;
      begin

         if Entry_Decl = null or else Entry_Decl.Task_Link.Decl_Kind /= A_Protected_Body_Declaration then
            if Plist /= Nil_Element_List then
               for I in Plist'Range loop
                  declare
                     DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                     Type_Ident : V_String := Get_Type_Ident(Plist(I));
                     Decl_Is_Access : Boolean;
                  begin
                     Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                     for J in Dname_List'Range loop
                        case Asis.Elements.Mode_Kind(Plist(I)) is
                           when A_Default_In_Mode | An_In_Mode =>
                              Use_Variable(Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Dname_List(J))), Keep_Nodenum);
                           when An_In_Out_Mode =>
                              Use_Variable(Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Dname_List(J))), Keep_Nodenum);
                              Assign_Variable(Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Dname_List(J))), Tmp_Nodenum);
                           when others =>
                              Assign_Variable(Varnum_List.Search_Node(Defining_List, Gela_Ids.Create_Id(Dname_List(J))), Tmp_Nodenum);
                        end case;
                     end loop;
                  end;
               end loop;
            end if;
            if Entry_Decl /= null then
               Enter_RequeueEntry(Entry_Decl, Keep_Nodenum,Tmp_Nodenum);
--             Call_Stacks.Push(Call_Stack, Subprogram_Call'(null, null, An_Entry_Call, False, False, 0, null, null));

            end if;
         else
            declare
--             Called_Entity : Asis.Declaration := Asis.Statements.Corresponding_Called_Entity(Elem);
--             Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Called_Entity));
               Subprog : Subprogram_Item_Link := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Entry_Decl.Decl_Id);
               Arg_Item : Args_Item_Link;
               Mode_Kind : Asis.Mode_Kinds;
               New_Call_Branch : Call_Branch_Item_Link := null;
            begin
               Current_Unit := Search_Unit_Item(Search_Node(Get_Info(Parent_Decl).Top).Belong_Unit);
               Parent_Decl_Spec := Prot_Entry_Decl_List.Search_Node(Protected_Entry_List, Search_Entry_Decl(To_V(
               Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Parent_Decl) (1)))));
               Current_Unit := Keep_Unit;
               Plist := Asis.Declarations.Parameter_Profile(Parent_Decl_Spec);
               if Subprog /= null then
                  Set_Call_Tree(Current_Routine.Subprogram, Subprog, Tmp_Nodenum);
                  if Subprog.Callee = 0 then
                     New_Call_Branch :=
                       Add_Call_Branch(Tmp_Nodenum, 0);
                     Call_Branch_Stack.Push(Subprog.Caller,
                                            New_Call_Branch);
                  else
                     New_Call_Branch :=
                       Add_Call_Branch(Tmp_Nodenum,
                                       Subprog.Callee);
                  end if;

                  if Plist /= Nil_Element_List then
                     if Is_Param_Node_Approach then
                        Set_Arg_Branch(Tmp_Nodenum);
                     end if;
                     Arg_Item := Subprog.Args;
                     for I in Plist'Range loop
                        declare
                           DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                           Type_Ident : V_String := Get_Type_Ident(Plist(I));
                           Decl_Is_Access : Boolean;
                        begin
                           Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                           for J in Dname_List'Range loop
                              Mode_Kind := Asis.Elements.Mode_Kind(Plist(I));
                              if not Is_Param_Node_Approach then
                                 Push(Var_Stack, Get_Varnum(Dname_List(J)));
                                 case Mode_Kind is
                                    when A_Default_In_Mode | An_In_Mode =>
                                       Add_Actual_In(New_Call_Branch.Actual_In, Arg_Item.Position, False);
                                    when An_In_Out_Mode =>
                                       Add_Actual_In(New_Call_Branch.Actual_In, Arg_Item.Position, True);
                                       Add_Actual_Out(New_Call_Branch.Actual_Out, Arg_Item.Position, False);
                                    when An_Out_Mode =>
                                       Add_Actual_Out(New_Call_Branch.Actual_Out, Arg_Item.Position, True);
                                    when others =>
                                       null;
                                 end case;
                              else

                                 case Mode_Kind is
                                    when A_Default_In_Mode
                                      | An_In_Mode
                                      | An_In_Out_Mode =>
                                       Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                                       Remove_Branch(Tmp_Nodenum-1, Tmp_Nodenum);
                                       Add_Branch(Keep_Nodenum, Tmp_Nodenum);
                                    if Mode_Kind = A_Default_In_Mode or Mode_Kind = An_In_Mode then
                                       Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                                       Set_Arg_Branch(Tmp_Nodenum);
                                       if Is_Debug_Mode then
                                          Ada.Text_Io.Put_Line("=== arg_join_stack : " & Nodenum_T'Image(Tmp_Nodenum));
```

```
                                        end if;
                                        Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                                     end if;
                                     Set_Info(Elem, (Keep_Nodenum, Tmp_Nodenum, Tmp_Nodenum +1));
                                     -- set_info
                                     if Arg_Item.Formal_In = 0 then
                                        Nodenum_Stack.Push(Arg_Item.Actual_In,
                                                           Tmp_Nodenum);
                                     else
                                        Add_Arg_Branch(Tmp_Nodenum,
                                                       Arg_Item.Formal_In);
                                     end if;
                                     when others =>
                                        null;
                                  end case;
                               end if;
                               Arg_Item := Arg_Item.Next;
                            end loop;
                         end;
                      end loop;
                      if Is_Param_Node_Approach then
                         Arg_Item := Subprog.Args;

                         for I in Plist'Range loop
                            declare
                               DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                               Type_Ident : V_String := Get_Type_Ident(Plist(I));
                               Decl_Is_Access : Boolean;
                            begin
                               Set_Is_Access(Plist(I), Decl_Is_Access, Type_Ident);
                               for J in Dname_List'Range loop
                                  case Asis.Elements.Mode_Kind(Plist(I)) is
                                     when An_In_Out_Mode
                                       | An_Out_Mode =>
                                        Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                                        Set_Arg_Branch(Tmp_Nodenum);
                                        Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                                        Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                                        if Arg_Item.Mode = An_Out_Mode then
                                        Add_Branch(Keep_Nodenum, Tmp_Nodenum);
                                        end if;
                                        -- set_info
                                        if Arg_Item.Formal_Out = 0 then
                                           Nodenum_Stack.Push(Arg_Item.Actual_Out,
                                                              Tmp_Nodenum);
                                        else
                                           Add_Arg_Branch(Tmp_Nodenum,
                                                          Arg_Item.Formal_Out);
                                        end if;
                                     when others =>
                                        null;
                                  end case;
                                  Arg_Item := Arg_Item.Next;
                               end loop;
                            end;
                         end loop;
                      end if;
                   end if;

--                 else

                   end if;
                end;
             end if;
          end;

          Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
          if Asis.Elements.Element_Kind(Encl_Entry) = A_Statement then
             Make_Labeled_Branch(Tmp_Nodenum, Label_T(Top(Accepthead_Stack)));
          else
             Make_Labeled_Branch(Tmp_Nodenum, Label_T(Top(Subprogramhead_Stack)));
          end if;
          if Asis.Elements.Statement_Kind(Elem) =
            A_Requeue_Statement_With_Abort then
             if Asis.Elements.Element_Kind(Encl_Entry) = A_Statement then
                Make_Labeled_Branch(Keep_Nodenum, Label_T(Top(Accepthead_Stack)));
             else
                Make_Labeled_Branch(Keep_Nodenum, Label_T(Top(Subprogramhead_Stack)));
             end if;
          end if;
       end;


    when A_Delay_Until_Statement =>
       null;

    when A_Delay_Relative_Statement =>
       null;

    when A_Terminate_Alternative_Statement =>
       null;

    when A_Selective_Accept_Statement =>
       Set_Nondet_Branch(Tmp_Nodenum);

    when A_Conditional_Entry_Call_Statement =>
       Set_Nondet_Branch(Tmp_Nodenum);

    when A_Timed_Entry_Call_Statement =>
       Set_Nondet_Branch(Tmp_Nodenum);

    when An_Asynchronous_Select_Statement =>
--     Set_Nondet_Branch(Tmp_Nodenum);
       null;
    when An_Abort_Statement =>
       null;

    when A_Raise_Statement =>
       null;
```

```
                when A_Code_Statement =>
                   null;

                when others =>
                   null;
             end case;
          exception
             when Asis.Exceptions.Asis_Inappropriate_Element =>
                Put_Line("raised ASIS.EXCEPTIONS.ASIS_INAPPROPRIATE_ELEMENT");
                Put_Line(Asis.Text.Element_Image(Elem));
                Put(Asis.Elements.Statement_Kind ( Elem ));
                New_Line;
          end;
       when An_Exception_Handler =>
--        Put_Line("exception_handler:");
--        Put_Line(Asis.Text.Element_Image(Elem));
          if Is_Exception_Node then
             if Current_Unit.Exception_Point = 0 then
                Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                Remove_Branch(Num_Of_Nodes, Num_Of_Nodes + 1);
                Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                Remove_Branch(Num_Of_Nodes, Num_Of_Nodes + 1);
                Current_Unit.Exception_Point := Tmp_Nodenum;
             else
                Remove_Branch(Current_Unit.Exception_Point -1, Num_Of_Nodes +1);
             end if;
             Add_Branch(Current_Unit.Exception_Point, Num_Of_Nodes + 1);
          else
             Control := Abandon_Children;
          end if;
       when A_Path =>
          case Asis.Elements.Path_Kind ( Elem ) is
             when An_If_Path =>
                Set_Info(Asis.Statements.Condition_Expression(Elem), Get_Info(Asis.Elements.Enclosing_Element(Elem)));
             when An_Elsif_Path =>
                Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                Set_Info(Asis.Statements.Condition_Expression(Elem),
                         (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
             when An_Else_Path =>
                null;
             when A_Case_Path =>
                null;
             when A_Select_Path | An_Or_Path =>
                declare
                   Guard_Item : Asis.Element := Asis.Statements.Guard(Elem);
                begin
--                 if Guard_Item /= Nil_Element then
                   if not Asis.Elements.Is_Nil(Guard_Item) then
                      Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Guard_item)));
                      Set_Info(Guard_Item, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum +1));
                   end if;
                end;

             when A_Then_Abort_Path =>
                null;
             when others =>
                null;
          end case;
       when others =>
          null;
    end case;
          ------< end of the case >------
   exception
      when others =>
         Put_Line("Exception raised in Pre_Op");
         Ada.Text_Io.Put_Line("Line : " & Asis.Text.Line_Number'Image(Asis.Text.First_Line_Number(Elem)));
         Put_Line(Asis.Text.Element_Image(Elem));
         Put_Line(Asis.Elements.Debug_Image(Elem));
         raise;
   end Pre_Op;

   procedure Post_Op(Elem : in Asis.Element;
                     Control : in out Asis.Traverse_Control;
                     State : in out boolean) is
      use Asis;
      Tmp_Nodenum : Nodenum_T;

--      procedure Use_Stacks(Nodenum : in Nodenum_T :=0) is
      procedure Use_Stacks is
         Self_Info : Element_Info := Get_Info(Elem);
         Self_Node : Nodenum_T;

--       Inserted_Node : Nodenum_T;
      begin
         if Self_Info /= Null_Element_Info then
            Self_Node := Self_Info.Top;
--       if Nodenum /= 0 then
--          Self_Node := Nodenum;
--       end if;
            while not isempty(vstring_stack) loop
               use_variable(top(vstring_stack), Self_Node);
               remove(vstring_stack);
            end loop;

            while not isempty(var_stack) loop
               use_variable(top(var_stack), Self_Node);
               remove(var_stack);
            end loop;

            while not isempty(Var_item_stack) loop
               use_variable(top(Var_Item_stack).number, Self_Node, Top(Var_Item_Stack).Is_Normalized, Top(Var_Item_Stack).Normalized_Form);
               remove(Var_Item_stack);
            end loop;


         else
            Clear(Vstring_Stack);
```

```
            Clear(Var_Stack);
            Clear(Var_Item_Stack);
--       Inserted_Node := Self_Node;
--       while not Isempty(Fnode_stack) loop
--          Insert_Function(Top(Fnode_stack), Inserted_Node, Elem);
--          Remove(Fnode_stack);
--       end loop;

         end if;

      end Use_Stacks;


      procedure Assign_Stacks is
         Self_Node : Nodenum_T := Get_Info(Elem).Top;
      begin

         while not isempty(A_var_stack) loop
            assign_variable(top(A_var_stack), Self_Node);
            remove(A_var_stack);
         end loop;

         while not isempty(A_Var_item_stack) loop
            assign_variable(top(A_Var_Item_stack).number, Self_Node, Top(A_Var_Item_Stack).Is_Normalized, Top(A_Var_Item_Stack).Normalized_Form);
            remove(A_Var_Item_stack);
         end loop;

      end Assign_Stacks;


   begin
      case Asis.Elements.Element_Kind(Elem) is
         when Not_An_Element =>
            null;
         when A_Declaration =>
            case Asis.Elements.Declaration_Kind(Elem) is
               when A_Task_Type_Declaration | A_Single_Task_Declaration =>
                  Exit_Task_Frame;

               when A_Protected_Body_Declaration =>
                  Exit_Protected;

               when A_Package_Declaration | A_Generic_Package_Declaration =>
                  Exit_Package;

               when A_Package_Body_Declaration =>
                  Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
                  if Current_Unit.Fork_Point = 0 then
                     Set_Fork_Point;
                  end if;
                  Add_P_Branch_To_Child;
                  Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                  if Current_Unit.Parent /= Main_Task then
                     declare
                        Bottom_Node : Nodenum_T := Get_Info(Elem).Bottom;
                        Node_Item : Node_Item_Link;
                     begin
                        Add_Branch(Tmp_Nodenum, Bottom_Node);
                        if Current_Unit.Fork_Point = Tmp_Nodenum then
                           Node_Item := Search_Node(Bottom_Node);
                           Node_Item.Receive := new Dug_Channel_Item'(Current_Unit, null, False, Node_Item.Receive);
                        end if;
                     end;
                  end if;
                  Exit_Package;

               when A_Procedure_Body_Declaration
                 | A_Function_Body_Declaration
                 | An_Entry_Body_Declaration =>
                  declare
                     Plist : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Elem);
                     Subprog_Decl : Asis.Declaration;
                     Subprog_Item : Subprogram_Item_Link;
                     Keep_Nodenum : Nodenum_T;
                  begin
                     Connect_Labels(Num_Of_Nodes +1, Label_T(Top(subprogramhead_Stack)));
                     Remove(subprogramhead_Stack);

                     Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
                     Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                     Keep_Nodenum := Tmp_Nodenum;
                     if Is_Param_Node_Approach then
                        if Plist /= Nil_Element_List then
                           Set_Arg_Branch(Keep_Nodenum);
                           if Asis.Elements.Declaration_Kind(Elem) = An_Entry_Body_Declaration then
                              Subprog_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Search_Entry_Decl(To_V(
                              Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names (Elem) (1)))));
                           else
                              Subprog_Decl := Asis.Declarations.Corresponding_Declaration(Elem);
--                            if Subprog_Decl = Nil_Element then
                              if Asis.Elements.Is_Nil(Subprog_Decl) then
                                 Subprog_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Elem));
                              else
                                 Subprog_Item := Subprogram_Decl_Id_List.Search_Node(Subprogram_Decl_List, Gela_Ids.Create_Id(Subprog_Decl));
                              end if;
                           end if;

                           for I in Plist'Range loop
                              case Asis.Elements.Mode_Kind(Plist(I)) is
                                 when An_Out_Mode | An_In_Out_Mode =>
                                    declare
                                       Dname_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                                       First_Nodenum : Nodenum_T := 0;
                                    begin
                                       for J in Dname_List'Range loop
                                          Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Plist(I))));
                                          Add_Top_Arg_Branch(Tmp_Nodenum);
                                          Remove_Branch(Tmp_Nodenum -1, Tmp_Nodenum);
                                          Add_Branch(Keep_Nodenum, Tmp_Nodenum);
```

```
                                    Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
                                    if First_Nodenum = 0 then
                                       First_Nodenum := Tmp_Nodenum;
                                    end if;
                                    Set_Info(Plist(I), (First_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                                    Use_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Tmp_Nodenum);
                                    declare
                                       Arg_Item : Args_Item_Link := Search_Args_Item(Subprog_Item.Args, To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))));
                                    begin
                                       if Arg_Item /= null then
                                          Arg_Item.Formal_Out := Tmp_Nodenum;
                                          Add_Stacked_Arg_Branch_Out(Arg_Item.all);
                                       end if;
                                    end;

                                 end loop;
                              end;
                           when others =>
                              null;
                        end case;
                     end loop;
                  end if;
               end if;

               if Current_Unit.Parent /= null and then
                 Current_Unit.Parent.Decl_Kind
                 = Asis.A_Protected_Body_Declaration
               then
                  declare
                     First_Node : Node_Item_Link := Search_Node(Current_Unit.First_Stmt);
                     Last_Node : Node_Item_Link := Search_Node(Keep_Nodenum);
                  begin
               -- protected variable
               -- protected procedure/function/entry
--                   First_Node.Assign := Current_Unit.Protected_Use;
--                   Last_Node.Refer := Current_Unit.Protected_Def;

                     First_Node.Receive := new Dug_Channel_Item'(Current_Unit.Parent, null, False, First_Node.Receive);
                     Last_Node.Send := new Dug_Channel_Item'(Current_Unit.Parent, null, False, Last_Node.Send);
                  end;
               end if;
            end;

            Add_P_Branch_To_Child;
            Add_P_Branch_From_Child;
            Exit_Unit;

         when A_Task_Body_Declaration =>
            Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
            Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
            Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, 0));

            Connect_Labels(Tmp_Nodenum, Label_T(Top(Taskhead_Stack)));
            Remove(Taskhead_Stack);

            Add_P_Branch_To_Child;
            Add_P_Branch_From_Child;
            if Current_Unit.Decl_Kind = A_Task_Type_Declaration then
               Add_Stacked_P_Branch(Task_Type_Id_List.Search_Node(Task_Type_Decl_List, Gela_Ids.Create_Id(Asis.Declarations.Corresponding_Declaration(Elem))));
            end if;
            Exit_Task;

         when A_Single_Protected_Declaration
           | A_Protected_Type_Declaration =>
            -- protected object ∅
            -- protected variable
--             Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
--             Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
--             Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, 0));

--             Protected_Last_Node(Tmp_Nodenum);
            declare
               Top_Node : Nodenum_T := Get_Info(Elem).Top;
               Bottom_Node : Nodenum_T := Get_Info(Elem).Bottom;
            begin
               if Search_Branch(Top_Node, Num_Of_Nodes +1) then
                  Remove_Branch(Top_Node, Num_Of_Nodes +1);
                  Add_Branch(Top_Node, Bottom_Node);
               else
                  Remove_Branch(Num_Of_Nodes, Num_Of_Nodes + 1);
                  Add_Branch(Num_Of_Nodes, Bottom_Node);
               end if;
            end;

            Exit_Protected_Frame;

         when A_Variable_Declaration |
           A_Constant_Declaration  =>
            Use_Stacks;
         when others =>
            null;
      end case;

   when A_Definition =>
      case Asis.Elements.Definition_Kind ( Elem ) is
         when A_Discrete_Range =>
            if Asis.Elements.Discrete_Range_Kind(Elem) = A_Discrete_Simple_Expression_Range then
               Disc_Range_Flag := False; -- ASIS bug
            end if;
         when others =>
            null;
      end case;

   when An_Association =>
      case Asis.Elements.Association_Kind ( Elem ) is
         when A_Parameter_Association =>
            declare
               Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
               Param_Name : V_String;
```

```
                  Temp_Exp : Asis.Expression;
                  Arg_Item : Args_Item_Link;
               begin
--                if Current_Call.Is_Same_Compilation_Unit then
                  if Current_Call.Args /= null then
                     if Asis.Expressions.Is_Normalized(Elem) then
                        Param_Name := To_V(Asis.Declarations.Defining_Name_Image(Asis.Expressions.Formal_Parameter(Elem)));
                        Arg_Item := Search_Args_Item(Current_Call.Args, Param_Name);
                     else
                        Temp_Exp := Asis.Expressions.Formal_Parameter(Elem);
--                      if Temp_Exp = Nil_Element then
                        if Asis.Elements.Is_Nil(Temp_Exp) then
                           Arg_Item := Current_Call.Args;
                           Current_Call.Args := Current_Call.Args.Next;
                           Call_Stacks.Remove(Call_Stack);
                           Call_Stacks.Push(Call_Stack, Current_Call);
                        else
                           Arg_Item := Search_Args_Item(Current_Call.Subprogram_Called.Args, To_V(Asis.Expressions.Name_Image(Temp_Exp)));
                        end if;

                     end if;

                     if not Is_Param_Node_Approach and Current_Call.Is_Same_Compilation_Unit then
                        case Arg_Item.Mode is
                           when A_Default_In_Mode | An_In_Mode =>
                              Add_Actual_In(Current_Call.Call_Branch.Actual_In, Arg_Item.Position, False);
                           when An_In_Out_Mode =>
                              Add_Actual_In(Current_Call.Call_Branch.Actual_In, Arg_Item.Position, True);
                              Add_Actual_Out(Current_Call.Call_Branch.Actual_Out, Arg_Item.Position, False);
                           when An_Out_Mode =>
                              Add_Actual_Out(Current_Call.Call_Branch.Actual_Out, Arg_Item.Position, True);
                           when others =>
                              null;
                        end case;

                     end if;

                     if Arg_Item.Mode = An_In_Out_Mode
                       or Arg_Item.Mode = An_Out_Mode then
                        if Current_Call.Is_Same_Compilation_Unit then
                           if Is_Param_Node_Approach then
                              Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
                              Set_Arg_Branch(Tmp_Nodenum);
                              Nodenum_Stack.Push(Arg_Join_Stack, Tmp_Nodenum);
                              Remove_Branch(Tmp_Nodenum, Tmp_Nodenum +1);
                              if Arg_Item.Mode = An_Out_Mode then
                                 Add_Branch(Current_Call.First_Node, Tmp_Nodenum);
                              end if;
                              declare
                                 Self_Node : Nodenum_T := Get_Info(Elem).Top;
                              begin
                                 Set_Info(Elem, (Self_Node, Tmp_Nodenum, Tmp_Nodenum + 1));
                              end;
                              while not Isempty(Var_Stack) loop
                                 Assign_Variable(Top(Var_Stack),
                                                 Tmp_Nodenum);
                                 Remove(Var_Stack);
                              end loop;

                              while not Isempty(Var_Item_Stack) loop
                                 Assign_Variable(Top(Var_Item_Stack).Number,
                                                 Tmp_Nodenum,
                                                 Top(Var_Item_Stack).Is_Normalized,
                                                 Top(Var_Item_Stack).Normalized_Form
                                                 );
                                 Remove(Var_Item_Stack);
                              end loop;
                              if Arg_Item.Formal_Out = 0 then
                                 Nodenum_Stack.Push(Arg_Item.Actual_Out,
                                                    Tmp_Nodenum);
                              else
                                 Add_Arg_Branch(Arg_Item.Formal_Out,
                                                Tmp_Nodenum);
                              end if;
                           end if;

                        elsif Current_Call.CallKind = An_Entry_Call then
                           Out_Arg_Stacks.Push(Current_Call.Out_Arg_Stack, Out_Arg_Item'(Var_Stack, Var_Item_Stack, Arg_Item));
                           Call_Stacks.Remove(Call_Stack);
                           Call_Stacks.Push(Call_Stack, Current_Call);
                           if Arg_Item.Mode = An_Out_Mode then
                              Clear(Vstring_Stack);
                              Clear(Var_Stack);
                           end if;
                        else
                           Tmp_Nodenum := Caller_Nodenum;
                           Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
                           while not Isempty(Var_Stack) loop
                              Assign_Variable(Top(Var_Stack),
                                              Tmp_Nodenum);
                              if Arg_Item.Mode = An_In_Out_Mode then
                                 Use_Variable(Top(Var_Stack),
                                              Tmp_Nodenum);
                              end if;
                              Remove(Var_Stack);
                           end loop;

                           while not Isempty(Var_Item_Stack) loop
                              Assign_Variable(Top(Var_Item_Stack).Number,
                                              Tmp_Nodenum,
                                              Top(Var_Item_Stack).Is_Normalized,
                                              Top(Var_Item_Stack).Normalized_Form
                                              );
                              if Arg_Item.Mode = An_In_Out_Mode then
                                 Use_Variable(Top(Var_Item_Stack).Number,
                                              Tmp_Nodenum,
                                              Top(Var_Item_Stack).Is_Normalized,
                                              Top(Var_Item_Stack).Normalized_Form
                                              );
```

```
                                 end if;
                                 Remove(Var_Item_Stack);
                              end loop;


                           end if;

--                         if not Out_Arg_Stacks.Isempty(Current_Call.Out_Arg_Stack) then
--                            declare
--                               Out_Var_Stack : Varnum_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Stack;
--                               Out_Var_Item_Stack : Var_Item_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Item_Stack;
--                            begin

--                            declare
--                               Arg_Item : Args_Item_Link := Out_Arg_Stacks.Top(Current_Call.Out_Arg_stack).Arg_Item;
--                            begin
--                            end;
                           -- out_arg
--                         Out_Arg_Stacks.Push(Current_Call.Out_Arg_Stack, Out_Arg_Item'(Var_Stack, Var_Item_Stack, Arg_Item));
--                         Call_Stacks.Remove(Call_Stack);
--                         Call_Stacks.Push(Call_Stack, Current_Call);
                        end if;
                        Set_Info(Elem, (Caller_Nodenum, Caller_Nodenum, Caller_Nodenum + 1));
                        Use_Stacks;

                     end if;
                  end;
--                Put("parameter_association : ");
--                Put_Line(Asis.Text.Element_Image(Elem));
--                declare
--                   Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
--                   Param_Name : V_String;
--                   Arg_Item : Args_Item_Link;
--                begin
--                   if Current_Call.Callkind = A_Procedure_Call or
--                     Current_Call.Callkind = A_Function_Call or
--                     Current_Call.Callkind = An_Entry_Call then
--                      if Current_Call.Is_Same_Compilation_Unit then
--                         if not Current_Call.Has_Node then
--                            Tmp_Nodenum := Add_Out_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                            Current_Call.First_Node := Tmp_Nodenum;
--                            if Current_Call.Subprogram_Called.Callee = 0 then
--                               Nodenum_Stack.Push(Current_Call.Subprogram_Called.Caller, Tmp_Nodenum);
--                            else
--                               Add_Call_Branch(Tmp_Nodenum, Current_Call.Subprogram_Called.Callee);
--                            end if;
--                            Current_Call.Has_Node := True;
--                         end if;
--                         if Asis.Expressions.Is_Normalized(Elem) then
--
--                            Param_Name := To_V(Asis.Declarations.Defining_Name_Image(Asis.Expressions.Formal_Parameter(Elem)));
--                            Arg_Item := Search_Args_Item(Current_Call.Args, Param_Name);
--                         else
--
--                            Arg_Item := Current_Call.Args;
--                            Current_Call.Args := Current_Call.Args.Next;
--                            Call_Stacks.Remove(Call_Stack);
--                            Call_Stacks.Push(Call_Stack, Current_Call);
-- --                         Param_Name := To_V(Asis.Expressions.Name_Image(Asis.Expressions.Formal_Parameter(Elem)));
--                         end if;
--
--                         if Arg_Item = null then
--                            Put_Line("can't find arg_item...");
--                         end if;
--                         if Arg_Item.Mode /= An_Out_Mode then
--                            if Current_Call.Callkind = A_Procedure_Call then
--                               Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                            else
--                               Tmp_Nodenum := Add_Out_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                            end if;
--                            Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum + 1));
--                            Use_Stacks;
--                            if Arg_Item.Formal_In = 0 then
--                               Nodenum_Stack.Push(Arg_Item.Actual_In,
--                                                  Tmp_Nodenum);
--                            else
--                               Add_Arg_Branch(Tmp_Nodenum,
--                                              Arg_Item.Formal_In);
-- --                            Put("arg_item.name : ");
-- --                            Put_Line(To_S(Arg_Item.Name));
-- --                            Put("formal_in : ");
-- --                            Node_IO.Put(Arg_Item.Formal_In);
-- --                            New_Line;
-- --                            Put("next formal_in : ");
-- --                            Node_IO.Put(Arg_Item.Next.Formal_In);
-- --                            New_Line;

--                            end if;
--                         end if;

--                         if Arg_Item.Mode = An_In_Out_Mode
--                           or Arg_Item.Mode = An_Out_Mode then
--                            Push(Out_Arg_Stack, Out_Arg_Item'(To_V(Asis.Expressions.Name_Image(Asis.Expressions.Actual_Parameter(Elem))), Arg_Item));
--                         end if;
--                      else
--                         null;
--                      end if;
--                   end if;
--                end;

               when others =>
                  null;
            end case;
         when An_Expression =>
            case Asis.Elements.Expression_Kind ( Elem ) is
               when A_Function_Call =>
                  if Is_Debug_Mode then
                     Put("function call : ");
                     Put_Line(Asis.Text.Element_Image(Elem));
```

```
                  end if;
                  declare
                     Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
                     Ident : V_String := To_V(Unique_Identifier);
--                   Fnode : Function_Nodes;
                  begin
                     if Current_Call.Is_Same_Compilation_Unit then
                        if not Is_Param_Node_Approach then
                           Add_Left_Actual_Label(Current_Call.Subprogram_Called.Args, Current_Call.Call_Branch);
                        end if;

                        Push(Var_Stack, Current_Call.Subprogram_Called.Function_Varnum);
--                      if not Current_Call.Has_Node then
--                         Tmp_Nodenum := Add_Out_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                         Current_Call.First_Node := Tmp_Nodenum;
--                         if Current_Call.Subprogram_Called.Callee = 0 then
--                            Nodenum_Stack.Push(Current_Call.Subprogram_Called.Caller, Tmp_Nodenum);
--                         else
--                            Add_Call_Branch(Tmp_Nodenum, Current_Call.Subprogram_Called.Callee);
--                         end if;
--                         Current_Call.Has_Node := True;
--                      end if;

--                      Tmp_Nodenum := Add_Out_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                      Remove_Branch(Tmp_Nodenum, Tmp_Nodenum+1);
--                      Set_Info(Elem, (Tmp_Nodenum, Tmp_Nodenum, Tmp_Nodenum));
--                      Assign_Variable(Enter_Variable(Ident, To_V(Asis.Expressions.Name_Image(Asis.Declarations.Result_Profile(
Asis.Expressions.Corresponding_Called_Function(Elem)))), False), Tmp_Nodenum);
--                      Fnode.Start_Node := Current_Call.First_Node;
--                      Fnode.End_Node := Tmp_Nodenum;
--                      Push(Fnode_Stack, Fnode);
--                      Push(Vstring_Stack, Ident);
                        Nodenum_Stack.Push(Current_Call.Subprogram_Called.Return_Caller, Current_Call.First_Node);
                        if not Nodenum_Stack.Isempty(Current_Call.Subprogram_Called.Return_Callee) then
                           declare
                              Iterator : Nodenum_Stack.Stack := Current_Call.Subprogram_Called.Return_Callee;
                              use Nodenum_Stack;
                           begin
                              while Iterator /= null loop
                                 Add_Arg_Branch(Nodenum_Stack.Top(Iterator), Current_Call.First_Node);
                                 Iterator := Iterator.Next;
                              end loop;
                           end;

                        end if;
                        Caller_Nodenum := Current_Call.First_Node;
                        if Is_Debug_Mode then
                           Ada.Text_Io.Put_Line("caller nodenum: " & Nodenum_T'Image(Caller_Nodenum));
                        end if;
                     end if;
                  end;
                  Call_Stacks.Remove(Call_Stack);
               when others =>
                  null;
            end case;

--          declare
--             Exp_Info : Element_Info := Get_Info(Elem);
--          begin
--             if Exp_Info /= Nil_Element_Info then
--                Use_Stacks(Exp_Info.Top);
--             end if;
--          end;

            if Get_Info(Elem) /= Null_Element_Info and then
              Get_Info(Elem) /= Assigned_Element_Info then
               if Is_Debug_Mode then
                  Put_Line("get_info /= null !!");
                  Put_Line(Asis.Text.Element_Image(Elem));
               end if;
               Use_Stacks;
            end if;

         when An_Exception_Handler =>
            Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
            Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
            Add_Branch(Current_Unit.Exception_Point -1, Tmp_Nodenum + 1);

         when A_Path =>
            declare
               SList : Asis.Statement_List := Asis.Statements.Sequence_Of_Statements(Elem);
               Top_Node : Nodenum_T := Get_Info(Slist(Slist'First)).Top;
               Bottom_Node : Nodenum_T := Get_Info(Slist(Slist'Last)).Bottom;
               Next_Node : Nodenum_T := Get_Info(Slist(Slist'Last)).Next;
            begin
               case Asis.Elements.Path_Kind ( Elem ) is
                  when A_Select_Path | An_Or_Path =>
                     declare
                        Guard_Item : Asis.Element := Asis.Statements.Guard(Elem);
                     begin
--                      if Guard_Item /= Nil_Element then
                        if not (Asis.Elements.Is_Nil(Guard_Item)) then
                           Top_Node := Get_Info(Guard_Item).Top;
                        end if;
                     end;
                  when An_Elsif_Path =>
                     Top_Node := Get_Info(Elem).Top;
                  when others =>
                     null;
               end case;

               Set_Info(Elem, (Top_Node, Bottom_Node, Next_node));
--             Put("(");
--             Node_Io.Put(Top_Node);
--             Put(", ");
--             Node_Io.Put(Bottom_Node);
--             Put(", ");
--             Node_Io.Put(Next_Node);
--             Put_line(")");
```

```
--             Put_Line(Gela_Ids.Debug_Image(Gela_Ids.Create_Id(Slist(Slist'first))));
--             Put_Line(Gela_Ids.Debug_Image(Gela_Ids.Create_Id(Slist(Slist'last))));
--             if Gela_Ids.Is_Equal(Gela_Ids.Create_Id(Slist(Slist'first)), Gela_Ids.Create_Id(Slist(Slist'last))) then
--                Put_Line("same!!!");
--             end if;
--             if Slist(Slist'first) = Slist(Slist'last) then
--                Put_Line("same element!!!");
--             end if;

            end;
            case Asis.Elements.Path_Kind ( Elem ) is
               when An_If_Path =>
                  null;
               when An_Elsif_Path =>
                  null;
               when An_Else_Path =>
                  null;
               when A_Case_Path =>
                  null;
               when A_Select_Path =>
                  null;
               when An_Or_Path =>
                  null;
               when A_Then_Abort_Path =>
                  null;
               when others =>
                  null;
            end case;
         when A_Statement =>
            Use_Stacks;
            Assign_Stacks;
            Use_Arg_Stacks;
            case Asis.Elements.Statement_Kind ( Elem ) is
               when A_Null_Statement =>
                  null;
               when An_If_Statement =>
                  declare
                     Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
                     Self_Node : Nodenum_T := Get_Info(Elem).Top;
                  begin
                     Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
                     Remove_Branch(Self_Node, Self_Node + 1);
                     Add_Branch(Self_Node, Tmp_Nodenum);
                     for I in Plist'Range loop
                        declare
                           Pkind : Asis.Path_Kinds := Asis.Elements.Path_Kind ( Plist(I) );
                           Pinfo : Element_Info := Get_Info(Plist(I));
                        begin
                           Add_Branch(Self_Node, Pinfo.Top);
                           if Pkind = An_Else_Path then
                              Remove_Branch(Self_Node, Tmp_Nodenum);
                           elsif Pkind = An_Elsif_Path then
                              Remove_Branch(Self_Node, Tmp_Nodenum);
                              Self_Node := Pinfo.Top;
                              Add_Branch(Self_Node, Tmp_Nodenum);
                           end if;
                           if Pinfo.Next /= 0 then
                              Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                              Add_Branch(Pinfo.Bottom, Tmp_Nodenum);
                           end if;
                        end;
                     end loop;
                     declare
                        Top_Node : Nodenum_T := Get_Info(Elem).Top;
                        Bottom_Node : Nodenum_T := Tmp_Nodenum;
                        Next_Node : Nodenum_T := Num_Of_Nodes + 1;
                     begin
                        Set_Info(Elem, (Top_Node, Bottom_Node, Next_Node));
                     end;
                  end;
               when A_Case_Statement =>
                  declare
                     Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
                     Self_Node : Nodenum_T := Get_Info(Elem).Top;
                  begin
                     Remove_Branch(Self_Node, Self_Node + 1);
                     Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
                     for I in Plist'Range loop
                        Add_Branch(Self_Node, Get_Info(Plist(I)).Top);
                        declare
                           Pinfo : Element_Info := Get_Info(Plist(I));
                        begin
                           if Pinfo.Next /= 0 then
                              Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                              Add_Branch(Pinfo.Bottom, Tmp_Nodenum);
                           end if;
                        end;
                     end loop;

                     declare
                        Top_Node : Nodenum_T := Get_Info(Elem).Top;
                        Bottom_Node : Nodenum_T := Tmp_Nodenum;
                        Next_Node : Nodenum_T := Num_Of_Nodes + 1;
                     begin
                        Set_Info(Elem, (Top_Node, Bottom_Node, Next_Node));
                     end;

                  end;

               when A_Loop_Statement | A_While_Loop_Statement
                 | A_For_Loop_Statement =>
                  declare
                     Self_Node : Nodenum_T := Get_Info(Elem).Top;
                  begin
                     Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
                     Remove_Branch(Tmp_Nodenum, Tmp_Nodenum + 1);
                     Add_Branch(Tmp_Nodenum, Self_Node);
                     if Asis.Elements.Statement_Kind ( Elem )
                       /= A_Loop_Statement then
```

```
                  Add_Branch(Self_Node, Tmp_Nodenum + 1);
               end if;
               Set_Info(Elem, (Self_Node, Tmp_Nodenum, Tmp_Nodenum + 1));
               loop
                  Tmp_Nodenum := Pop_Exit_Node(Elem);
                  exit when Tmp_Nodenum = 0;
                  Add_Branch(Tmp_Nodenum, Num_Of_Nodes +1);
--                Node_Io.Put(Tmp_Nodenum);
               end loop;
            end;


         when A_Selective_Accept_Statement =>
            declare
               Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
               Self_Node : Nodenum_T := Get_Info(Elem).Top;
            begin
               Remove_Branch(Self_Node, Self_Node + 1);
               for I in Plist'Range loop
                  declare
                     Slist : Asis.Statement_List := Asis.Statements.Sequence_Of_Statements(Plist(I));
                  begin
                     if Slist'Length = 1 and then
                       Asis.Elements.Statement_Kind(Slist(Slist'First))
                       = A_Terminate_Alternative_Statement then
                        Make_Labeled_Branch(Self_Node, Label_T(Top(Taskhead_Stack)));
                        -- Add_Branch(Self_Node, Num_Of_Nodes +1);
                     else
                        declare
                           Pinfo : Element_Info := Get_Info(Plist(I));
                        begin
                           Add_Branch(Self_Node, Pinfo.Top);
                           if Pinfo.Next /= 0 then
                              Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                              Add_Branch(Pinfo.Bottom, Num_Of_Nodes +1);
                           end if;
                        end;
                     end if;
                  end;
               end loop;
            end;

         when A_Timed_Entry_Call_Statement | A_Conditional_Entry_Call_Statement =>
            declare
               Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
               Self_Node : Nodenum_T := Get_Info(Elem).Top;
            begin
               Remove_Branch(Self_Node, Self_Node + 1);
               for I in Plist'Range loop
                  declare
                     Pinfo : Element_Info := Get_Info(Plist(I));
                  begin
                     Add_Branch(Self_Node, Pinfo.Top);
                     if Pinfo.Next /= 0 then
                        Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                        Add_Branch(Pinfo.Bottom, Num_Of_Nodes +1);
                     end if;
                  end;
               end loop;
            end;
         when An_Asynchronous_Select_Statement =>
            declare
               Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Elem);
               Self_Node : Nodenum_T := Get_Info(Elem).Top;
            begin
               Remove_Branch(Self_Node, Self_Node + 1);
               for I in Plist'Range loop
                  declare
                     Pinfo : Element_Info := Get_Info(Plist(I));
                  begin
                     if Asis.Elements.Path_Kind(Plist(I)) = A_Select_Path then
                        Add_Fork_Branch(Self_Node, Pinfo.Top);
                        if Pinfo.Next /= 0 then
                           Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                           Add_Join_Branch(Pinfo.Bottom, Num_Of_Nodes +1);
                        end if;

                     else
                        Add_Branch(Self_Node, Pinfo.Top);
                        if Pinfo.Next /= 0 then
                           Remove_Branch(Pinfo.Bottom, Pinfo.Next);
                           Add_Branch(Pinfo.Bottom, Num_Of_Nodes +1);
                        end if;
                     end if;
                  end;
               end loop;
            end;

         when An_Assignment_Statement =>
            declare
               Self_Node : Nodenum_T := Get_Info(Elem).Top;
            begin
               Set_Info(Elem, (Self_Node, Num_Of_Nodes, Num_Of_Nodes + 1));
            end;

         when A_Block_Statement =>
            if Asis.Statements.Is_Declare_Block( Elem ) then
               Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
               Set_Info(Elem, (Get_Info(Elem).Top, Tmp_Nodenum, Tmp_Nodenum + 1));
               Add_P_Branch_To_Child;
               Add_P_Branch_From_Child;
               Exit_Unit;
            end if;
         when An_Accept_Statement =>
            declare
               Plist : Asis.Parameter_Specification_List := Asis.Statements.Accept_Parameters(Elem);
            begin
               if Asis.Statements.Accept_Body_Statements(Elem)
                 /= Nil_Element_List or else
```

```
                  Plist /= Nil_Element_List then
                   Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.Last_Line_Number(Elem)));
                   Set_Info(Elem, (Get_Info(Elem).top, Tmp_Nodenum, Tmp_Nodenum +1));
                   Connect_Labels(Tmp_Nodenum, Label_T(Top(Accepthead_Stack)));
                else
                   Tmp_Nodenum := 0;
                end if;
                if Plist /= Nil_Element_List then

                   for I in Plist'Range loop
                      declare
                         DName_List : Asis.Defining_Name_List := Asis.Declarations.Names(Plist(I));
                      begin
                         for J in Dname_List'Range loop

                            case Asis.Elements.Mode_Kind(Plist(I)) is
                               when An_Out_Mode
                                 | An_In_Out_Mode =>
                                  Use_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname_List(J))), Tmp_Nodenum);
                               when others => null;
                            end case;
                         end loop;
                      end;
                   end loop;
                end if;
             end;
             Remove(Accepthead_Stack);
             Exit_Accept(Gela_Ids.Create_Id(Asis.Statements.Corresponding_Entry(Elem)), Tmp_Nodenum);
          when A_Procedure_Call_Statement =>
             declare
                Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
             begin
                if Current_Call.Is_Same_Compilation_Unit then
                   if Is_Debug_Mode then
                      Put_Line(To_S(Current_Call.Subprogram_Called.Name));
                   end if;

                   if not Is_Param_Node_Approach then
                      Add_Left_Actual_Label(Current_Call.Subprogram_Called.Args, Current_Call.Call_Branch);
                   end if;

--                 while not Out_Arg_Stacks.Isempty(Current_Call.Out_Arg_Stack) loop
--                    Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                    declare
--                       Self_Node : Nodenum_T := Get_Info(Elem).Top;
--                    begin
--                       Set_Info(Elem, (Self_Node, Tmp_Nodenum, Tmp_Nodenum + 1));
--                    end;
--                    declare
--                       Out_Var_Stack : Varnum_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Stack;
--                       Out_Var_Item_Stack : Var_Item_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Item_Stack;
--                    begin
--                       while not Isempty(Out_Var_Stack) loop
--                          Assign_Variable(Top(Out_Var_Stack),
--                                          Tmp_Nodenum);
--                          Remove(Out_Var_Stack);
--                       end loop;
--                       while not Isempty(Out_Var_Item_Stack) loop
--                          Assign_Variable(Top(Out_Var_Item_Stack).Number,
--                                          Tmp_Nodenum,
--                                          Top(Out_Var_Item_Stack).Is_Normalized,
--                                          Top(Out_Var_Item_Stack).Normalized_Form
--                                          );
--                          Remove(Out_Var_Item_Stack);
--                       end loop;
--                    end;

--                    declare
--                       Arg_Item : Args_Item_Link := Out_Arg_Stacks.Top(Current_Call.Out_Arg_stack).Arg_Item;
--                    begin
--                       if Arg_Item.Formal_Out = 0 then
--                          Nodenum_Stack.Push(Arg_Item.Actual_Out,
--                                             Tmp_Nodenum);
--                       else
--                          Add_Arg_Branch(Arg_Item.Formal_Out,
--                                         Tmp_Nodenum);
--                       end if;
--                    end;

--                    Out_Arg_Stacks.Remove(Current_Call.Out_Arg_Stack);
--                 end loop;
                end if;
             end;
             Call_Stacks.Remove(Call_Stack);

          when An_Entry_Call_Statement =>
             declare
                Entry_Decl : Entry_Item_Link := Entry_Decl_Id_List.Search_Node(Entry_Decl_List, Gela_Ids.Create_Id(Asis.Statements.Corresponding_Called_Entity(Elem)));
                Current_Call : Subprogram_Call := Call_Stacks.Top(Call_Stack);
             begin
                if Entry_Decl = null or else
                  Entry_Decl.Task_Link.Decl_Kind = A_Protected_Body_Declaration then
                   if Current_Call.Is_Same_Compilation_Unit then
                      if not Is_Param_Node_Approach then
                         Add_Left_Actual_Label(Current_Call.Subprogram_Called.Args, Current_Call.Call_Branch);
                      end if;

--                    while not Out_Arg_Stacks.Isempty(Current_Call.Out_Arg_Stack) loop
--                       Tmp_Nodenum := Add_Node(Linenum_T(Asis.Text.First_Line_Number(Elem)));
--                       declare
--                          Self_Node : Nodenum_T := Get_Info(Elem).Top;
--                       begin
--                          Set_Info(Elem, (Self_Node, Tmp_Nodenum, Tmp_Nodenum + 1));
--                       end;
--                       declare
--                          Out_Var_Stack : Varnum_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Stack;
--                          Out_Var_Item_Stack : Var_Item_Stacks.Stack := Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Var_Item_Stack;
--                       begin
--                          while not Isempty(Out_Var_Stack) loop
```

```
--                                Assign_Variable(Top(Out_Var_Stack),
--                                                Tmp_Nodenum);
--                                Remove(Out_Var_Stack);
--                             end loop;
--                             while not Isempty(Out_Var_Item_Stack) loop
--                                Assign_Variable(Top(Out_Var_Item_Stack).Number,
--                                                Tmp_Nodenum,
--                                                Top(Out_Var_Item_Stack).Is_Normalized,
--                                                Top(Out_Var_Item_Stack).Normalized_Form);
--                                Remove(Out_Var_Item_Stack);
--                             end loop;
--                          end;

-- --                       Assign_Variable(Out_Arg_Stacks.Top(Current_Call.Out_Arg_Stack).Name,
-- --                                       Tmp_Nodenum);
--                          declare
--                             Arg_Item : Args_Item_Link := Out_Arg_Stacks.Top(Current_Call.Out_Arg_stack).Arg_Item;
--                          begin
--                             if Arg_Item.Formal_Out = 0 then
--                                Nodenum_Stack.Push(Arg_Item.Actual_Out,
--                                                   Tmp_Nodenum);
--                             else
--                                Add_Arg_Branch(Arg_Item.Formal_Out,
--                                               Tmp_Nodenum);
--                             end if;
--                          end;

--                          Out_Arg_Stacks.Remove(Current_Call.Out_Arg_Stack);
--                       end loop;
                      end if;
                   else
                      while not Out_Arg_Stacks.Isempty(Current_Call.Out_Arg_Stack) loop
                         declare
                            Tmp_Nextcall : Nextcall := Nextcall_Stacks.Top(Nextcall_Stack);
                         begin
                            Tmp_Nextcall.Arg_Item := Current_Call.Out_Arg_Stack;
                            Nextcall_Stacks.Remove(Nextcall_Stack);
                            Nextcall_Stacks.Push(Nextcall_Stack, Tmp_Nextcall);
                         end;
                         Out_Arg_Stacks.Remove(Current_Call.Out_Arg_Stack);
                      end loop;
                   end if;
                end;

                Call_Stacks.Remove(Call_Stack);

             when A_Requeue_Statement =>
--              Call_Stacks.Remove(Call_Stack);
                null;
             when A_Return_Statement =>
                declare
                   Self_Node : Nodenum_T := Get_Info(Elem).Top;
                begin
                   Set_Info(Elem, (Self_Node, Num_Of_Nodes, Num_Of_Nodes + 1));
                end;
                Remove_Branch(Num_Of_Nodes, Num_Of_Nodes+1);
                Make_Labeled_Branch(Num_Of_Nodes, Label_T(Top(Subprogramhead_Stack)));

             when others =>
                null;
          end case;
--        case Asis.Elements.Statement_Kind ( Elem ) is

        when others =>
           null;
     end case;
   exception
      when others =>
         Put_Line("Exception raised in Post_Op");
         Ada.Text_Io.Put_Line("Line : " & Asis.Text.Line_Number'Image(Asis.Text.First_Line_Number(Elem)));
         Put_Line(Asis.Text.Element_Image(Elem));
         Put_Line(Asis.Elements.Debug_Image(Elem));
         raise;
   end Post_Op;

   procedure Print_Usage is
   begin
--       Put_Line("USAGE: " & Ada.Command_Line.Command_Name & " -s|-b unitname");
--       Put_Line("  -s    processing unit specification.");
--       Put_Line("  -b    processing unit body.");
      Ada.Text_Io.Put_Line("USAGE: " & Ada.Command_Line.Command_Name & " unitname");
   end Print_Usage;

   Cl_Count : Natural := Ada.Command_Line.Argument_Count;
   Begin_Time : Ada.Calendar.Day_Duration;
begin

   -----------------------
   -- analysis--
   Begin_Time := (Ada.Calendar.Seconds(Ada.Calendar.Clock));
   -----------------------

--   if Ada.Command_Line.Argument_Count /= 2 then
   if Cl_Count /= 1 and Cl_Count /= 2 then
      Print_Usage;
      return;
   end if;

   if Cl_Count = 2 then
      if Ada.Command_Line.Argument(1) = "-d" then
         Is_Debug_Mode := True;
      elsif Ada.Command_Line.Argument(2) = "-d" then
         Is_Debug_Mode := True;
         Cl_Count := 1;
      else
         Print_Usage;
         return;
      end if;
   end if;
```

```
   -- Initialization of Asis environment.
   Asis.Implementation.Initialize (Wide_String'(""));
   Asis.Ada_Environments.Associate ( The_Context => My_Context,
                                     Name => "My_Context",
                                     Parameters => "-FS");
   Asis.Ada_Environments.Open ( My_Context );
   -----------------------------

   declare
      Unite : String := Ada.Command_Line.Argument(Cl_Count);
      Control : Asis.Traverse_Control := Asis.Continue;
      State : Boolean := True;
--      Is_Body : Boolean;
   begin

--       if Ada.Command_Line.Argument(1) = "-s" then
--       The_Unit := Asis.Compilation_Units.Library_Unit_Declaration(Unite, My_Context);
--       Is_Body := False;
--       elsif Ada.Command_Line.Argument(1) = "-b" then
--       The_Unit := Asis.Compilation_Units.Compilation_Unit_Body(Unite, My_Context);
--       Is_Body := True;
--       else
--       Print_Usage;
--       return;
--       end if;

      The_Unit_Spec := Asis.Compilation_Units.Library_Unit_Declaration(To_Wide_String(Unite), My_Context);
      The_Unit_Body := Asis.Compilation_Units.Compilation_Unit_Body(To_Wide_String(Unite), My_Context);

      -- If it's null, continuing makes no sense ...
      if ( Asis.Compilation_Units.Is_Nil ( The_Unit_Body ) ) then
         Ada.Text_Io.Put_Line ( "Unit " & Unite & " is Nil...");
         raise Asis.Exceptions.Asis_Inappropriate_Compilation_Unit ;
      end if ;
      if (not Asis.Compilation_Units.Is_Nil(The_Unit_Spec)) then
         Is_Body := False;
         Generate_DUN(Asis.Elements.Unit_Declaration(The_Unit_Spec), Control, State);
      end if;
      Is_Body := True;
      Generate_DUN(Asis.Elements.Unit_Declaration(The_Unit_Body), Control, State);
      Process_Nextcall;
      if Is_Debug_Mode then
         Output_Call_Tree(Main_Task);
      end if;
      Asis.Ada_Environments.Close(My_Context);
      Asis.Ada_Environments.Dissociate(My_Context);
      Asis.Implementation.Finalize(Wide_String'(""));
      Put_Line("done.");
--       if Is_Body then
--       Output_DUN(Unite & "_b");
--       else
--       Output_DUN(Unite & "_s");
--       end if;
      Output_Dun(Unite);
   end;

   -----------------------
   -- analysis--
   -----------------------
   Ada.Text_Io.Put("-- time" &
                   Ada.Calendar.Day_Duration'Image
                   (Begin_Time));
   Ada.Text_Io.Put_Line(" - " &
                        Ada.Calendar.Day_Duration'Image
                        (Ada.Calendar.Seconds(Ada.Calendar.Clock)));
   -----------------------



exception

   when Asis.Exceptions.Asis_Inappropriate_Compilation_Unit =>
      Ada.Text_Io.Put_Line ( "The unit " & Ada.Command_Line.Argument (1) &
                " is not Ada Unit.");
      New_Line ;
      Print_Usage;
      raise ;

end Ada2dun;
```

A.1.5 Asis_Application_Driver_1 Package

```
------------------------------------------------------------------------------
--                                                                          --
--                    ASIS APPLICATION TEMPLATE COMPONENTS                  --
--                                                                          --
--             A S I S _ A P P L I C A T I O N _ D R I V E R _ 1            --
--                                                                          --
--                                 S p e c                                  --
--                                                                          --
--           Copyright (c) 2000, Free Software Foundation, Inc.             --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
```

```
-- ASIS Application Templates were developed and are now maintained by Ada  --
-- Core Technologies Inc (http://www.gnat.com).                             --
--                                                                          --
------------------------------------------------------------------------------

--  This procedure implements the template for the top-level driver of an
--  ASIS application. This driver processes all the Compilation Units which
--  are located in a given ASIS Context. By default, the Context is made up
--  by all the tree files contained in the current directory.
--
--  The driver does not have parameter.
--
--  When working, this driver generates the trace containg the names of
--  compilation units being processed.

procedure ASIS_Application_Driver_1;
```



```
with Ada.Exceptions;
with Ada.Wide_Text_IO;
with Ada.Characters.Handling;
with DUN_Handler; use DUN_Handler;
with Asis;
with Asis.Ada_Environments;
with Asis.Implementation;
with Asis.Exceptions;
with Asis.Errors;

with Context_Processing;

procedure ASIS_Application_Driver_1 is

   My_Context            : Asis.Context;

   My_Context_Name       : Wide_String := Asis.Ada_Environments.Default_Name;
   --  The default name in case of the GNAT ASIS implementation is empty.
   --  If you would like to have some non-null name for your ASIS Context,
   --  change the initialization expression in this declaration. Note, that
   --  in the GNAT ASIS implementation the name of a Context does not have any
   --  special meaning or semantics associated with particular names.

   My_Context_Parameters : Wide_String :=
      Asis.Ada_Environments.Default_Parameters;
   --  The default COntext parameters in case of the GNAT ASIS implementation
   --  are an empty string. This corresponds to the following Context
   --  definition: "-CA -FT -SA", and has the following meaning:
   --  -CA - a Context is made up by all the tree files in the tree search
   --        path; the tree search path is not set, so the default is used,
   --        and the default is the current directory;
   --  -FT - only pre-created trees are used, no tree file can be created by
   --        ASIS;
   --  -SA - source files for all the Compilation Units belonging to the
   --        Context (except the predefined Standard package) are considered
   --        in the consistency check when opening the Context;
   --
   --  If you would like to use some other Context definition, you have to
   --  replace the initialization expression in this declaration by
   --  the corresponding Parameters string. See the ASIS Reference Manual for
   --  the full details of the Context definition for the GNAT ASIS
   --  implementation.

   Initialization_Parameters : Wide_String := "";
   Finalization_Parameters   : Wide_String := "";
   --  If you would like to use some specific initialization or finalization
   --  parameters, you may set them here as initialization expressions in
   --  the declarations above.

begin
   --  The code below corresponds to the basic sequencing of calls to ASIS
   --  queries which is mandatory for any ASIS application.
   --  First, the ASIS implementation should be initialized
   --  (Asis.Implementation.Initialize), then an application should define
   --  an ASIS Context to process by associating the Context with an external
   --  environment (Asis.Ada_Environments.Associate), then the Context should
   --  be opened (sis.Ada_Environments.Open ). After that all the ASIS queries
   --  may be used (the ASIS processing is encapsulated in
   --  Context_Processing.Process_Context. When the ASIS analysis is over, the
   --  application should release the system resources by closing the ASIS
   --  Context (Asis.Ada_Environments.Close), breaking the association of the
```

```
   --  Context with the external world (Asis.Ada_Environments.Dissociate) and
   --  finalizing the ASIS implementation (Asis.Implementation.Finalize).

   Asis.Implementation.Initialize     (Initialization_Parameters);

   Asis.Ada_Environments.Associate
     (The_Context => My_Context,
      Name        => My_Context_Name,
      Parameters  => My_Context_Parameters);

   Asis.Ada_Environments.Open         (My_Context);

   Context_Processing.Process_Context (The_Context => My_Context,
                                       Trace       => True);
   --Output_Dun("result.dun");
   Asis.Ada_Environments.Close        (My_Context);
   Asis.Ada_Environments.Dissociate   (My_Context);
   Asis.Implementation.Finalize       (Finalization_Parameters);

exception
   --  The exception handling in this driver is somewhat redundant and may
   --  need some reconsidering when using this driver in real ASIS tools

   when Ex : Asis.Exceptions.ASIS_Inappropriate_Context          |
             Asis.Exceptions.ASIS_Inappropriate_Container        |
             Asis.Exceptions.ASIS_Inappropriate_Compilation_Unit |
             Asis.Exceptions.ASIS_Inappropriate_Element          |
             Asis.Exceptions.ASIS_Inappropriate_Line             |
             Asis.Exceptions.ASIS_Inappropriate_Line_Number      |
             Asis.Exceptions.ASIS_Failed                         =>

      Ada.Wide_Text_IO.Put ("ASIS exception (");
      Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
               Ada.Exceptions.Exception_Name (Ex)));
      Ada.Wide_Text_IO.Put (") is raised");
      Ada.Wide_Text_IO.New_Line;

      Ada.Wide_Text_IO.Put ("ASIS Error Status is ");
      Ada.Wide_Text_IO.Put
        (Asis.Errors.Error_Kinds'Wide_Image (Asis.Implementation.Status));
      Ada.Wide_Text_IO.New_Line;

      Ada.Wide_Text_IO.Put ("ASIS Diagnosis is ");
      Ada.Wide_Text_IO.New_Line;
      Ada.Wide_Text_IO.Put (Asis.Implementation.Diagnosis);
      Ada.Wide_Text_IO.New_Line;

      Asis.Implementation.Set_Status;

   when Ex : others =>

      Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
               Ada.Exceptions.Exception_Name (Ex)));
      Ada.Wide_Text_IO.Put (" is raised (");
      Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
               Ada.Exceptions.Exception_Information (Ex)));
      Ada.Wide_Text_IO.Put (")");
      Ada.Wide_Text_IO.New_Line;

end ASIS_Application_Driver_1;
```

A.1.6 Context_Processing Package

```
------------------------------------------------------------------------------
--                                                                          --
--                    ASIS APPLICATION TEMPLATE COMPONENTS                  --
--                                                                          --
--                     C O N T E X T _ P R O C E S S I N G                  --
--                                                                          --
--                                 S p e c                                  --
--                                                                          --
--            Copyright (c) 2000, Free Software Foundation, Inc.            --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Application Templates were developed and are now maintained by Ada  --
-- Core Technologies Inc (http://www.gnat.com).                             --
--                                                                          --
------------------------------------------------------------------------------

--  This package contains routines for high-level processing of
--  (terating through) an ASIS Context

with Asis;

package Context_Processing is

   procedure Process_Context
     (The_Context : Asis.Context;
      Trace       : Boolean := False);
   --  This procedure iterates through the whole content of its argument
   --  Context and it calls a unit processing routine for those ASIS
   --  Compilation Units which are of An_Application_Unit origin (that is,
   --  user-defined units). If Trace parameter is set ON, it generate the
```

```
   --  simple trace of the unit processing (consisting of the names of the
   --  units in the Context being processed or skipped).

   function Get_Unit_From_File_Name
     (Ada_File_Name : String;
      The_Context   : Asis.Context)
      return          Asis.Compilation_Unit;
   --  Supposing that Ada_File_Name is the name of an Ada source file which
   --  follows the GNAT file naming rules (see the GNAT Users Guide), this
   --  function tries to get from The_Context the ASIS Compilation Unit
   --  contained in this source file. The source file name may contain the
   --  directory information in relative or absolute form.
   --
   --  If The_Context does not contain the ASIS Compilation Unit which
   --  may be the content of the argument file, Nil_Compilation_Unit is
   --  returned.
   --
   --  Note, that this function always return Nil_Compilation_Unit, if
   --  Ada_File_Name is a file name which is krunched. Nil_Compilation_Unit
   --  is also returned if Ada_File_Name correspond to any name of a child
   --  unit from the predefined or GNAT-specific hierarchy (children of
   --  System, Ada, Interfaces, and GNAT)

end Context_Processing;



------------------------------------------------------------------------------
--                                                                          --
--                    ASIS APPLICATION TEMPLATE COMPONENTS                  --
--                                                                          --
--                     C O N T E X T _ P R O C E S S I N G                  --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--            Copyright (c) 2000, Free Software Foundation, Inc.            --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Application Templates were developed and are now maintained by Ada  --
-- Core Technologies Inc (http://www.gnat.com).                             --
--                                                                          --
------------------------------------------------------------------------------
with DUN_Handler; use DUN_Handler;
with Ada.Wide_Text_IO;
with Ada.Characters.Handling;
with Ada.Exceptions;
with Ada.Text_IO;
--with Ada.Integer_Text_IO;
with gnat.Case_Util;
with Asis.Compilation_Units;
with Asis.Exceptions;
with Asis.Errors;
with Asis.Implementation;
with ada.Characters.Handling;
with Unit_Processing;

with Ada.Calendar; use Ada.Calendar;
with V_Strings; use V_Strings;
use Ada.Wide_Text_IO;
--use Ada.Text_IO, Ada.Integer_Text_IO;

package body Context_Processing is

   -----------------------------
   -- Get_Unit_From_File_Name --
   -----------------------------

   function Get_Unit_From_File_Name
     (Ada_File_Name : String;
      The_Context   : Asis.Context)
      return Asis.Compilation_Unit
   is
   begin

      return Asis.Nil_Compilation_Unit;

      --  To be completed....

   end Get_Unit_From_File_Name;

   --------------------
   -- Process_Context --
   --------------------

   procedure Process_Context
     (The_Context : Asis.Context;
      Trace       : Boolean := False)
   is
      Units : Asis.Compilation_Unit_List :=
         Asis.Compilation_Units.Compilation_Units (The_Context);
      Counter : integer := 1;
      Next_Unit        : Asis.Compilation_Unit := Asis.Nil_Compilation_Unit;
      Next_Unit_Origin : Asis.Unit_Origins     := Asis.Not_An_Origin;
      Next_Unit_Class  : Asis.Unit_Classes     := Asis.Not_A_Class;

      Turkey : FILE_TYPE;

      package Fix_IO is new Ada.Wide_Text_IO.Fixed_IO(DAY_DURATION);
```

```
   use Fix_IO;
   Year, Month, Day : INTEGER;
   Start, Seconds : DAY_DURATION;
   Time_And_Date : TIME;

begin
   Ada.Wide_Text_IO.Create(Turkey, Out_File, "METRICS2.TXT");
   Ada.Wide_Text_IO.Set_Output(Turkey);

   Time_And_Date :=Clock;
   Split(Time_And_Date, Year, Month, Day, Start);  --get start time
   Ada.Wide_Text_IO.Put("It is beginning time:");
   Fix_IO.Put(Start);
   Ada.Wide_Text_IO.New_Line;

   for J in Units'Range loop
      Next_Unit        := Units (J);
      Next_Unit_Class  := Asis.Compilation_Units.Unit_Class (Next_Unit);
      Next_Unit_Origin := Asis.Compilation_Units.Unit_Origin (Next_Unit);

      if Trace then
         Ada.Wide_Text_IO.Put ("Processing Unit: ");
         Ada.Wide_Text_IO.Put
           (Asis.Compilation_Units.Unit_Full_Name (Next_Unit));

         case Next_Unit_Class is
            when Asis.A_Public_Declaration |
                 Asis.A_Private_Declaration =>

               Ada.Wide_Text_IO.Put (" (spec)");

            when Asis.A_Separate_Body =>
               Ada.Wide_Text_IO.Put (" (subunit)");

            when Asis.A_Public_Body |
                 Asis.A_Public_Declaration_And_Body |
                 Asis.A_Private_Body =>

               Ada.Wide_Text_IO.Put_Line (" (body)");

            when others =>
               Ada.Wide_Text_IO.Put_Line (" (???)");
         end case;

         Ada.Wide_Text_IO.New_Line;

      end if;

      case Next_Unit_Origin is
         when Asis.An_Application_Unit =>
            Current_Unit := Main_Task;
            Unit_Processing.Process_Unit (Next_Unit);
            --  This is the call to the procedure which performs the
            --  analysis of a particular unit

            Output_Dun(ada.Characters.Handling.To_String(asis.Compilation_Units.Unit_Full_Name(Next_Unit)));
            Counter := Counter + 1;
            if Trace then
               Ada.Wide_Text_IO.Put ("Done ...");
            end if;

         when Asis.A_Predefined_Unit =>
            if Trace then
               Ada.Wide_Text_IO.Put ("Skipped as a predefined unit");
            end if;

         when Asis.An_Implementation_Unit =>

            if Trace then
               Ada.Wide_Text_IO.Put
                 ("Skipped as an implementation-defined unit");
            end if;

         when Asis.Not_An_Origin =>

            if Trace then
               Ada.Wide_Text_IO.Put
                 ("Skipped as nonexistent unit");
            end if;

      end case;

      if Trace then
         Ada.Wide_Text_IO.New_Line;
         Ada.Wide_Text_IO.New_Line;
      end if;

   end loop;


   Time_And_Date :=Clock;
   Split(Time_And_Date, Year, Month, Day, Seconds);  --get end time
   Ada.Wide_Text_IO.Put("It is endding time:");
   Fix_IO.Put(Seconds);
   Ada.Wide_Text_IO.New_Line;
   Ada.Wide_Text_IO.Put("The elapsed time is :");
   Fix_IO.Put(Seconds - Start);
   Ada.Wide_Text_IO.New_Line;

exception
   --  The exception handling in this procedure is somewhat redundant and
   --  may need some reconsidering when using this procedure as a template
   --  for a real ASIS tool

   when Ex : Asis.Exceptions.ASIS_Inappropriate_Context         |
             Asis.Exceptions.ASIS_Inappropriate_Container       |
             Asis.Exceptions.ASIS_Inappropriate_Compilation_Unit |
             Asis.Exceptions.ASIS_Inappropriate_Element         |
             Asis.Exceptions.ASIS_Inappropriate_Line            |
```

```
               Asis.Exceptions.ASIS_Inappropriate_Line_Number      |
               Asis.Exceptions.ASIS_Failed                         =>

         Ada.Wide_Text_IO.Put ("Process_Context : ASIS exception (");

         Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
                 Ada.Exceptions.Exception_Name (Ex)));

         Ada.Wide_Text_IO.Put (") is raised when processing unit ");

         Ada.Wide_Text_IO.Put
            (Asis.Compilation_Units.Unit_Full_Name (Next_Unit));

         Ada.Wide_Text_IO.New_Line;

         Ada.Wide_Text_IO.Put ("ASIS Error Status is ");

         Ada.Wide_Text_IO.Put
           (Asis.Errors.Error_Kinds'Wide_Image (Asis.Implementation.Status));

         Ada.Wide_Text_IO.New_Line;

         Ada.Wide_Text_IO.Put ("ASIS Diagnosis is ");
         Ada.Wide_Text_IO.New_Line;
         Ada.Wide_Text_IO.Put (Asis.Implementation.Diagnosis);
         Ada.Wide_Text_IO.New_Line;

         Asis.Implementation.Set_Status;

      when Ex : others =>

         Ada.Wide_Text_IO.Put ("Process_Context : ");

         Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
                 Ada.Exceptions.Exception_Name (Ex)));

         Ada.Wide_Text_IO.Put (" is raised (");

         Ada.Wide_Text_IO.Put (Ada.Characters.Handling.To_Wide_String (
                 Ada.Exceptions.Exception_Information (Ex)));

         Ada.Wide_Text_IO.Put (")");
         Ada.Wide_Text_IO.New_Line;

         Ada.Wide_Text_IO.Put ("when processing unit");

         Ada.Wide_Text_IO.Put
            (Asis.Compilation_Units.Unit_Full_Name (Next_Unit));

         Ada.Wide_Text_IO.New_Line;


    Close(Turkey);

   end Process_Context;

end Context_Processing;
```

A.1.7 Dun_Handler Package

```
with Asis;
with Gela_Ids;
with Asis.Declarations;
with V_Strings, Id_List, Stacks;
use V_Strings;

package DUN_Handler is

   Is_Param_Node_Approach : constant Boolean := False;


   Is_Debug_Mode : Boolean := False;

   Is_Exception_Node : Constant Boolean := False;

   Identlast : Integer := 0;


   type Unit_ID is new Natural;
   type Linenum_T is new Natural;
   type Label_T is new Natural;
   type Varnum_T is new Natural;
   type Nodenum_T is new Natural;
   type Entrynum_T is new Natural;
   Num_Of_Nodes: Nodenum_T:= 0;
   Num_Of_Units: Unit_ID := 1;
   Num_Of_Variables: Varnum_T:= 0;
   Num_Of_Entries : Entrynum_T := 0;
   Last_Node: Nodenum_T := 0;

   package Nodenum_Stack is new Stacks(Nodenum_T);
   Arg_Join_Stack : Nodenum_Stack.Stack := null;


   package varnum_stacks is new stacks(varnum_t);
   Var_Stack : Varnum_Stacks.Stack := null;
   A_Var_Stack : Varnum_Stacks.Stack := null;

   package vstring_stacks is new stacks(v_string);
   Vstring_Stack : Vstring_Stacks.Stack := null;
   Assign_Stack : Vstring_Stacks.Stack := null;

   type Element_Info is
      record
 Top : Nodenum_T := 0;
 Bottom : Nodenum_T := 0;
```

```
 Next : Nodenum_T := 0;
      end record;

   Null_Element_Info : Element_Info := (0, 0, 0);
   Assigned_Element_Info : Element_Info := (Nodenum_T'Last, Nodenum_T'Last, Nodenum_T'Last);
--   Discrete_Range_Element_Info : Element_Info := (Nodenum_T'Last, Nodenum_T'Last, 0);
   package Element_Id_List is new Id_List(Element_Info, Null_Element_Info);
   Info_List : Element_Id_List.A_NODE := null;

   package Node_Stack_List is new Id_List(Nodenum_Stack.Stack, null);
   Exit_Stack_List : Node_Stack_List.A_Node := null;

   package Varnum_List is new Id_List(Varnum_T, 0);
   Defining_List : Varnum_List.A_Node := null;


   package Prot_Entry_Decl_List is new Id_List(Asis.Declaration, Asis.Nil_Element);
   Protected_Entry_List : Prot_Entry_Decl_List.A_Node := null;

   type Var_Item;
   type Entry_Item;
   type Var_Item_Link is access Var_Item;
   type Entry_Item_Link is access Entry_Item;

   package Entry_Decl_ID_List is new Id_List(Entry_Item_Link, null);
   Entry_Decl_List : Entry_Decl_Id_List.A_Node := null;

   type Subprogram_Item;
   type Subprogram_Item_Link is access Subprogram_Item;

   package Subprogram_Decl_ID_List is new Id_List(Subprogram_Item_Link, null);
   Subprogram_Decl_List : Subprogram_Decl_Id_List.A_Node := null;

   Variable_Table_Tail: Var_Item_Link:= null;

   type Unit_Item;
   type Unit_Item_Link is access Unit_Item'Class;

   type Task_Type_Item;
   type Task_Type_Item_Link is access Task_Type_Item;

   type Dug_Channel_Item;
   type Dug_Channel_Item_Link is access Dug_Channel_Item;

   type Task_Type_Info is
      record
 Belong_Unit : Unit_Item_Link := null;
 Fork_Source : Nodenum_Stack.Stack := null;
 Join_Dest : Nodenum_Stack.Stack := null;
 Parent_Sync : Dug_Channel_Item_Link := null;
      end record;
   type Task_Type_Info_Link is access Task_Type_Info;
   type Task_Type_Item is
      record
 Info : Task_Type_Info_Link;
 Is_Access_Type : Boolean := False;
 Next : Task_Type_Item_Link := null;
      end record;



   package Task_Type_Id_List is new Id_List(Task_Type_Info_Link, null);
   Task_Type_Decl_List : Task_Type_Id_List.A_Node := null;



   type Entry_Calls;
   type Entry_Call_Link is access Entry_Calls;

   type Node_Item;
   type Branch_Item;
   type Node_Line_Item;
   type Node_Item_Link is access Node_Item;
   type Branch_Item_Link is access Branch_Item;
   type Node_Line_Item_Link is access Node_Line_Item;

   type Entry_Calls is
      record
         Task_Called: Unit_Item_Link:= null;
         Entry_Called: Entry_Item_Link:= null;
         Node: Node_Item_Link:= null;
         Next: Entry_Call_Link:= null;
      end record;

   type Accept_Node;
   type Accept_Node_Link is access Accept_Node;

   type Accept_Node is
      record
         Start_Node: Nodenum_T;
         End_Node: Nodenum_T;
         Next: Accept_Node_Link:= null;
      end record;

   Current_Accept: Entry_Item_Link:= null; -- not in accept stmt

   type Args is
      record
         Name: V_String;
         Mode: Asis.Mode_Kinds := Asis.A_Default_In_Mode;
 Position : Natural := 0;
      end record;

   package Args_Stacks is new Stacks(Args);
   Args_Stack : Args_Stacks.Stack := null;

   type Args_Item;
   type Args_Item_Link is access Args_Item;
--   type Arg_Mode is ( IN_M, OUT_M, INOUT_M );
```

```
   type Args_Item is
      record
         Name: V_String;
         Mode: Asis.Mode_Kinds := Asis.A_Default_In_Mode;
 Actual_In, Actual_Out : Nodenum_Stack.Stack := null;
 Formal_In, Formal_Out : Nodenum_T := 0;
 Position : Natural := 0;
         Next: Args_Item_Link;
      end record;



   type Dug_Var_Item;
   type Dug_Var_Item_Link is access Dug_Var_Item;

   type Dug_Var_Item is
      record
         Number: Varnum_T;
 Is_Normalized : Boolean := False;
 Normalized_Form : V_String;
         Next: Dug_Var_Item_Link;
      end record;

   Not_Found : constant Dug_Var_Item := (0, False, Null_Str, null);


   type Dug_Var_Group_Item;
   type Dug_Var_Group_Item_Link is access Dug_Var_Group_Item;

   type Dug_Var_Group_Item is
      record
 Contents: Dug_Var_Item_Link := null;
 Position : Natural := 0;
         Next: Dug_Var_Group_Item_Link := null;
      end record;

   package Var_Item_Stacks is new Stacks(Dug_Var_Item); -- for normalized form
   Var_Item_Stack : Var_Item_Stacks.Stack := null;
   A_Var_Item_Stack : Var_Item_Stacks.Stack := null;

   type Out_Arg_Item is
      record
-- Name : V_String;
 Var_Stack : Varnum_Stacks.Stack := null;
 Var_Item_Stack : Var_Item_Stacks.Stack := null;
 Arg_Item : Args_Item_Link := null;
      end record;

   package Out_Arg_Stacks is new Stacks(Out_Arg_Item);
   use Out_Arg_Stacks;
--   Out_Arg_Stack : Out_Arg_Stacks.Stack := null;


   type Dug_Channel_Item is
      record
         Unit_Link: Unit_Item_Link:= null;
         Entry_Link: Entry_Item_Link:= null;
         Ending: Boolean:= False;
 Next : Dug_Channel_Item_Link := null;
      end record;

   No_Channel: constant Dug_Channel_Item:= (null, null, False, null);
   Previous_Call: Dug_Channel_Item:= No_Channel;
--    First_Send : Dug_Channel_Item := No_Channel;
--

   type Nextcall is
      record
 Previous_Node : Nodenum_T;
 Channel_Item : Dug_Channel_Item_Link;
 Arg_Item : Out_Arg_Stacks.Stack := null;
 Is_Send : Boolean := False;
      end record;

   package Nextcall_Stacks is new Stacks(Nextcall);
   Nextcall_Stack : Nextcall_Stacks.Stack := null;


   type Unit_Item is tagged
     record
Name : V_String;
ID : Unit_ID;
Parent: Unit_Item_Link:= null;
Children: Unit_Item_Link:= null;
Brother: Unit_Item_Link:= null;
First_Stmt: Nodenum_T:= 0;
Begin_Stmt : Nodenum_T := 0;
Fork_Point : Nodenum_T := 0;
Vars: Var_Item_Link:= null;
Entry_Calls: Entry_Call_Link:= null;
Subprograms: Subprogram_Item_Link := null;
Subprogram : Subprogram_Item_Link := null;
Nodes: Node_Item_Link:= null;
Next: Unit_Item_Link:= null;
Decl_Kind: Asis.Declaration_Kinds := Asis.Not_A_Declaration;
Protected_Use : Dug_Var_Item_Link := null;
Protected_Def : Dug_Var_Item_Link := null;
Entries : Entry_Item_Link:= null;
Exception_Point : Nodenum_T := 0;
Decl_By_Task_Type : Task_Type_Item_Link := null;
     end record;

--    type Task_Item is new Unit_Item with
--       record
--       end record;
--    type Task_Item_Link is access Task_Item;
```

```
--   type Procedure_Item is new Unit_Item;
-- with
--       record
-- Entries : Entry_Item_Link:= null;
--       end record;

   Main_Task: Unit_Item_Link
     := new Unit_Item'(To_V(String'("MAIN")), 1, null, null, null, 0, 0, 0,
                       null, null, null, null, null, null,
       Asis.Not_A_Declaration, null, null, null, 0, null);
   Current_Unit : Unit_Item_Link := Main_Task;
   Last_Unit: Unit_Item_Link:= Main_Task;

   type Call_Branch_Item;
   type Call_Branch_Item_Link is access Call_Branch_Item;
   type Call_Branch_Item is
      record
         Head: Nodenum_T:= 0;
 Actual_In : Dug_Var_Group_Item_Link := null;
 Actual_Out : Dug_Var_Group_Item_Link := null;
         Next: Call_Branch_Item_Link:= null;
      end record;
   package Call_Branch_Stack is new Stacks(Call_Branch_Item_Link);

   Is_Body : Boolean := True;

   type Node_Item is
      record
         Belong_Unit : Unit_ID := 0;
         Number: Nodenum_T:= 0;
         Linenum: Linenum_T:= 0;
 Is_Body : Boolean := True; -- s0 or b0
         Assign: Dug_Var_Item_Link:= null; -- <def>
         Refer: Dug_Var_Item_Link:= null; -- <use>
--          Send: Dug_Channel_Item:= No_Channel; -- <send>
--          Receive: Dug_Channel_Item:= No_Channel; -- <receive>
 Send : Dug_Channel_Item_Link := null;
 Receive : Dug_Channel_Item_Link := null;
         Branch: Branch_Item_Link:= null; -- <connect>
         Det_Branch: Boolean:= True; -- <s-
 Arg_Branch: Boolean:= False; -- <a-
         Forks: Branch_Item_Link:= null; -- <fork>
         Joins: Branch_Item_Link:= null; -- <join>
 Calls: Call_Branch_Item_Link:= null; -- <call>
 Args: Branch_Item_Link:= null; -- <param>
 Top_Args : Branch_Item_Link:= null; -- <arg>
 Formal_In : Dug_Var_Item_Link := null; -- <f-in>
 Formal_Out : Dug_Var_Item_Link := null; -- <f-out>
         Next: Node_Item_Link:= null;
         Glob_Next: Node_Item_Link:= null;
         Is_Pre : Boolean := False;
         Is_Post : Boolean := False;
      end record;

   Node_Tail: Node_Item_Link:= null;

   type Node_Line_Item is
      record
         Linenum: Linenum_T;
         Node: Node_Item_Link;
         Next: Node_Line_Item_Link;
      end record;

   Current_Line: Node_Line_Item_Link:= null; -- first_line;

   type Branch_Item is
      record
         Head: Nodenum_T:= 0;
         Label: Label_T:= 0;
         Next: Branch_Item_Link:= null;
      end record;


   type Var_Item is
      record
         Unit_Link: Unit_Item_Link;
 Routine_Link : Unit_Item_Link;
         Name: V_String;
         Number: Varnum_T;
 Type_Ident : V_String;
 Is_Access : Boolean := False;
 Is_Protected : Boolean := False;
         Next: Var_Item_Link:= null;
         T_Next: Var_Item_Link:= null;
      end record;

   type Entry_Item is
      record
         Name: V_String;
         Number: Entrynum_T;
 Decl_Id : Gela_Ids.Id;
         Args: Args_Item_Link;
 Task_Link : Unit_Item_Link := null;
         Nodes: Accept_Node_Link:= null;
         Next: Entry_Item_Link:= null;
 Is_Protected : Boolean := False;
      end record;

   type Kind_Of_Call is (A_Procedure_Call, A_Function_Call, An_Entry_Call,
 A_Protected_Entry_Call);
   type Nodenum_List;
   type Nodenum_List_Link is access Nodenum_List;
   type Nodenum_List is
      record
 Content : Nodenum_T := 0;
 Next : Nodenum_List_Link := null;
      end record;
```

```
   type Call_Tree;
   type Call_Tree_Link is access Call_Tree;
   type Call_Tree is
      record
 Partner : Subprogram_Item_Link := null;
 Caller_Node : Nodenum_List_Link := null;
 Next : Call_Tree_Link;
      end record;

   type Subprogram_Item is
      record
         Name: V_String;
-- Id : Asis.Ids.Id;
         Args: Args_Item_Link := null;
 Belong_Unit : Unit_Item_Link := null;
--         Caller : Nodenum_Stack.Stack := null;
 Caller : Call_Branch_Stack.Stack := null;
 Callee : Nodenum_T := 0;
 Return_Caller : Nodenum_Stack.Stack := null;
 Return_Callee : Nodenum_Stack.Stack := null;
 Type_Ident : V_String;
 Is_Access : Boolean := False;
 Calling_Tree : Call_Tree_Link := null;
 Called_Tree : Call_Tree_Link := null;
 Function_Varnum : Varnum_T := 0;
         Next: Subprogram_Item_Link:= null;
         Precon_Nodenum : Nodenum_T := 0;
         Postcon_Nodenum : Nodenum_T := 0;
      end record;


   type Subprogram_Call is
      record
-- Name: V_String;
 Subprogram_Called : Subprogram_Item_Link := null;
 Args : Args_Item_Link := null;
 Callkind : Kind_Of_Call := A_Procedure_Call;
--         Args: Args_Item_Link := null;
 Is_Same_Compilation_Unit : Boolean := True;
 Has_Node : Boolean := False;
 First_Node : Nodenum_T := 0;
 Out_Arg_Stack : Out_Arg_Stacks.Stack := null;
 Call_Branch : Call_Branch_Item_Link := null;
      end record;

   type Function_Nodes is
      record
 Start_Node, End_Node : Nodenum_T;
      end record;

   function Current_Routine return Unit_Item_Link;

   function Search_Branch
     (Tail : Nodenum_T; Head : Nodenum_T) return Boolean;

   function Add_Node
     (Linenum : in Linenum_T)
      return Nodenum_T;

   procedure Add_Node
     (Linenum : in Linenum_T);

   function Add_Out_node
     (Linenum : Linenum_T) return Nodenum_T;

   procedure Add_Branch
     (Tail : in Nodenum_T;
      Head : in Nodenum_T);

   procedure Remove_Branch
     (Tail : in Nodenum_T;
      Head : in Nodenum_T);

   function Search_Node
     (Nodenum : in Nodenum_T)
      return Node_Item_Link;

   procedure Enter_Task_Frame
     (Task_Name : in V_String);

   function Enter_Task
     (Task_Name : in V_String)
      return Unit_ID;

   procedure Enter_Task
     (Task_Name : in V_String);

   procedure Enter_Protected_Frame
     (Protected_Name : in V_String);

   function Enter_Protected
     (Protected_Name : in V_String)
      return Unit_ID;

   procedure Enter_Protected
     (Protected_Name : in V_String);


   procedure Enter_Unit
     (Unit_Name : in V_String; Decl_Kind : in Asis.Declaration_Kinds);

   procedure Enter_Accept
     (Elem_ID : in Gela_Ids.Id;
      Node_Num : in Nodenum_T);

   procedure Exit_Accept
     (Elem_ID : in Gela_Ids.Id;
      Node_Num : in Nodenum_T);
```

```
  procedure First_Stmt;

  function Search_Entry_Decl(Entry_Name : in V_String ) return Gela_Ids.Id;

  procedure Exit_Unit;

  procedure Exit_Task_Frame renames Exit_Unit;

  procedure Exit_Task renames Exit_Unit;

  procedure Exit_Protected_Frame renames Exit_Unit;

  procedure Exit_Protected renames Exit_Unit;

  procedure Enter_Package_Frame(Unit_Name : in V_String; Decl_Kind : in Asis.Declaration_Kinds) renames Enter_Unit;

  procedure Enter_Package(Task_Name : in V_String) renames Enter_Task;

  procedure Exit_Package renames Exit_Unit;

  procedure Enter_Entrydec
    (Entry_Name : in V_String;
     Elem_Id : in Gela_Ids.Id;
     Arg_S      : in Args_Stacks.Stack);

  procedure Enter_Entrycall
    (Entry_Decl : in Entry_Item_Link;
     Node_Num : in Nodenum_T);

  procedure Enter_RequeueEntry
    (Entry_Decl : in Entry_Item_Link;
     Node_Num : in Nodenum_T;
     Out_Node : in Nodenum_T);

  procedure Enter_Proceduredec
    (Procedure_Name : in V_String;
     Elem_Id : in Gela_Ids.Id;
     Arg_S      : in Args_Stacks.Stack);

  procedure Enter_Functiondec
    (Function_Name : in V_String;
     Elem_Id : in Gela_Ids.Id;
     Arg_S      : in Args_Stacks.Stack;
     Function_Varnum : in Varnum_T);

  function Search_Unit_Item(Unit_Num : Unit_ID) return Unit_Item_Link;

  function Search_Args_Item(Call_Args : Args_Item_Link ; Arg_Name : V_String)
    return Args_Item_Link;

  function Add_Call_Branch
    (Tail : in Nodenum_T;
     Head : in Nodenum_T) return Call_Branch_Item_Link;

  procedure Add_Stacked_Call_Branch(Subprog : in out Subprogram_Item);

  procedure Add_Actual_In(Actual_In : in out Dug_Var_Group_Item_Link;
  Position : in Natural;
 Is_An_In_Out_Mode : in Boolean);

  procedure Add_Actual_Out(Actual_Out : in out Dug_Var_Group_Item_Link;
   Position : in Natural;
  Is_An_Out_Mode : in Boolean);

  procedure Add_Left_Actual_Label(Args : Args_Item_Link;
  Call_Item : Call_Branch_Item_Link);

  procedure Add_Formal_In
    (Var_Num  : in Varnum_T;
     Node_Num : in Nodenum_T);

  procedure Add_Formal_Out
    (Var_Num  : in Varnum_T;
     Node_Num : in Nodenum_T);



  procedure Add_Arg_Branch
    (Tail : in Nodenum_T;
     Head : in Nodenum_T);

  procedure Add_Top_Arg_Branch
    (Head : in Nodenum_T);

  procedure Add_Stacked_Arg_Branch_In
    (Arg_Item : in out Args_Item);

  procedure Add_Stacked_Arg_Branch_Out
    (Arg_Item : in out Args_Item);

  procedure Set_Fork_Point;

  procedure Add_P_Branch_To_Child;

  procedure Add_P_Branch_From_Child;

  procedure Add_Fork_Branch(Source, Dest : in Nodenum_T);
  procedure Add_Join_Branch(Source, Dest : in Nodenum_T);
  procedure Add_Stacked_P_Branch(Task_Type : in Task_Type_Info_Link);

  procedure Add_P_Branch_From_Allocate(Task_Type : in Task_Type_Info_Link);
  function Unique_Identifier return String;

  function Enter_Variable
    (Var_Name : in V_String; Type_Ident : in V_String;
     Is_Access : in Boolean)
     return Varnum_T;

  function Enter_variable
```

```
      (Dname : in Asis.Defining_Name; Type_Ident : in V_String;
      Is_Access : in Boolean)
       return Varnum_T;

   procedure Enter_Variable
     (Var_Name : in V_String; Type_Ident : in V_String;
      Is_Access : in Boolean);

   procedure Remove_Variable;

   procedure Assign_Variable
     (Var_Name : in V_String;
      Node_Num : in Nodenum_T);

   procedure Assign_Variable
     (Var_num : in Varnum_T;
      Node_Num : in Nodenum_T;
      Is_Normalized : in Boolean := False;
      Normal_Form : in V_String := Null_Str);

   procedure Use_Variable
     (Var_Name : in V_String;
      Node_Num : in Nodenum_T);

   procedure Use_Variable
     (Var_num : in Varnum_T;
      Node_Num : in Nodenum_T;
      Is_Normalized : in Boolean := False;
      Normal_Form : in V_String := Null_Str);

   procedure Set_Info(Id : in Gela_Ids.Id; Info : in Element_Info);
   procedure Set_Info(Elem : in Asis.Element ; Info : in Element_Info);
   function Get_Info(Id : Gela_Ids.Id) return Element_Info;
   function Get_Info(Elem : Asis.Element) return Element_Info;

   procedure Insert_Function(Fnode : in Function_Nodes; Node_Num : in out Nodenum_T; Elem : in Asis.Element);

   procedure Process_Nextcall;

   function Search_Function_Call(Function_Name : V_String)
 return Subprogram_Item_Link;

   procedure Set_Call_Tree(Caller, Callee : in out Subprogram_Item_Link ;
   Node : in Nodenum_T);
   procedure Output_Call_Tree(Unit : in Unit_Item_Link);

   function Find_Visible_Var_Link
     (Var_Name : in V_String; Use_Var : in Boolean)
      return Var_Item_Link;

--    function Fullname_Of_Var
--      (Var_Num : in Varnum_T)
--       return V_String;

--    function Fullname_Of_Var
--      (Var : in Dug_Var_Item_Link)
--       return V_String;

--    function Fullname_Of_Unit
--      (Unit_Num : in Unit_ID)
--       return V_String;

--    function Fullname_Of_Unit
--      (Unit_Item : in Unit_Item_Link)
--       return V_String;

   function Search_Var_Item
     (Var_Num : in Varnum_T)
      return Var_Item_Link;

   procedure Set_Nondet_Branch
     (Nodenum : in Nodenum_T);

   procedure Set_Arg_Branch
     (Nodenum : in Nodenum_T);

   procedure Make_Labeled_Branch
     (Tail  : in Nodenum_T;
      Label : in Label_T);

   procedure Connect_Labels
     (Head  : in Nodenum_T;
      Label : in Label_T);

   procedure Protected_Last_Node
     (Node_Num : in Nodenum_T);

   procedure Output_Dun
     (File_Name : in String);

end DUN_Handler;



with Stacks;
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Wide_Text_IO;
with Asis;
with Gela_Ids;
with Asis.Declarations;
with Asis.Elements;
with Asis.Text;
with V_Strings, Id_List, Stacks;
use V_Strings;

package body DUN_Handler is
```

```
   package N_IO is new Integer_IO(Nodenum_T);

   use Args_Stacks;

   -- call tree
   function Current_Routine return Unit_Item_Link is
      Result : Unit_Item_Link := Current_Unit;
      use Asis;
   begin
      Put_Line(to_s(Result.Name));
      if Result /= Main_Task then -- add code
         while Result.Parent /= Main_Task loop
         case Result.Decl_Kind is
            when A_Procedure_Body_Declaration
                 | A_Function_Body_Declaration
                 | An_Entry_Body_Declaration =>
                 --| A_Function_Declaration =>
               return Result;
            when others =>
               null;
         end case;
         Result := Result.Parent;
         end loop;
      end if;-- add code
      return Result;
   end Current_Routine;


   function Add_Node(Linenum : in Linenum_T)
      return Nodenum_T
   is
      New_Node: Node_Item_Link;
      New_Line: Node_Line_Item_Link;
   begin
      Num_Of_Nodes := Num_Of_Nodes + 1;
      Last_Node := Num_Of_Nodes;
--      Put_Line("in Add_Node,");

      New_Node:= new Node_Item;
      New_Node.Belong_Unit := Current_Unit.ID;
      New_Node.Number  := Num_Of_Nodes;
      New_Node.Linenum := Linenum;
      New_Node.Is_Body := Is_Body;
      New_Node.Assign  := null;
      New_Node.Refer   := null;
      New_Node.Branch  := null;
      New_Node.Det_Branch := True;
      New_Node.Arg_Branch := False;
      New_Node.Next    := Current_Unit.Nodes;
      Current_Unit.Nodes:= New_Node;
      New_Node.Glob_Next:= Node_Tail;
      Node_Tail        := New_Node;

--       if Previous_Call /= No_Channel then
--          New_Node.Receive:= Previous_Call;
--          Previous_Call:= No_Channel;
--       end if;

--       if First_Send /= No_Channel then
--          New_Node.Send = First_Send;
--          Previous_Call:= No_Channel;
--       end if;

      New_Line:= new Node_Line_Item;
      New_Line.Linenum:= Linenum;
      New_Line.Node   := New_Node;
      New_Line.Next   := Current_Line;
      Current_Line:= New_Line;

      Add_Branch(Num_Of_Nodes, Num_Of_Nodes+1);

--      while not Isempty(Last_Ecall_Outarg) loop
--         Assign_Variable(Top(Last_Ecall_Outarg), Num_Of_Nodes);
--         Remove(Last_Ecall_Outarg);
--      end loop;

      if Current_Unit.Fork_Point /= 0 and then Current_Unit.Begin_Stmt = 0 then
         Remove_Branch(Current_Unit.Fork_Point, Current_Unit.Fork_Point+1);
         Add_Branch(Current_Unit.Fork_Point, Num_Of_Nodes);
         Current_Unit.Begin_Stmt := Num_Of_Nodes;
         declare
            Begin_Node : Node_Item_Link := Search_Node(Num_Of_Nodes);
         begin
            Begin_Node.Receive := new Dug_Channel_Item'(Current_Unit, null, False, Begin_Node.Receive);
         end;

      end if;

      return(Num_Of_Nodes);
   end Add_Node;


   function Search_Branch
     (Tail : Nodenum_T; Head : Nodenum_T) return Boolean is
      Node : Node_Item_Link:= Search_Node(Tail);
      function Search_Branch_Iter(Iter_Branch : Branch_Item_Link)
                                  return Boolean is
      begin
         if Iter_Branch = null then
            return False;
         elsif Iter_Branch.Head = Head then
            return True;
         else
            return Search_Branch_Iter(Iter_Branch.Next);
         end if;
      end Search_Branch_iter;

   begin
      return Search_Branch_Iter(Node.Branch);
```

```
end Search_Branch;

-- Add_Node
procedure Add_Node
  (Linenum : in Linenum_T)
is
   Dummy: Nodenum_T;
begin
   Dummy:= Add_Node(Linenum);
end Add_Node;


function Add_Out_node
  (Linenum : Linenum_T) return Nodenum_T
is
   Dummy: Nodenum_T;
   Keep_Last_Node: Nodenum_T := Last_Node;
begin
   Dummy:= Add_Node(Linenum);
   Last_Node := Keep_Last_Node;
   Remove_Branch(Last_Node, Dummy);
   Add_Branch(Last_Node, Num_Of_Nodes + 1);
   return Dummy;
end Add_Out_Node;

procedure Add_Branch
  (Tail : in Nodenum_T;
   Head : in Nodenum_T)
is
   Node : Node_Item_Link:= Search_Node(Tail);
   New_Branch : Branch_Item_Link:= new Branch_Item;
begin
   if not Search_Branch(Tail, Head) then
      New_Branch.Head:= Head;
      New_Branch.Next:= Node.Branch;
      Node.Branch:= New_Branch;
   end if;
end Add_Branch;


function Search_Node
  (Nodenum : in Nodenum_T)
   return Node_Item_Link
is
   Now: Node_Item_Link:= Node_Tail;
begin
   while Now /= null loop
      if Now.Number = Nodenum then
         return Now;
      end if;
      Now:= Now.Glob_Next;
   end loop;

   return(null);
end Search_Node;


procedure Remove_Branch
  (Tail : in Nodenum_T;
   Head : in Nodenum_T)
is
   Node : Node_Item_Link:= Search_Node(Tail);
   procedure Remove_Branch_Iter(Iter_Branch : in out Branch_Item_Link) is
   begin
      if Iter_Branch = null then
         -- put_line("Warning: attempt to remove nonexistent node.");
         return;
      elsif Iter_Branch.Head = Head then
         Iter_Branch := Iter_Branch.Next;
      else
         Remove_Branch_Iter(Iter_Branch.Next);
      end if;
   end Remove_Branch_Iter;

begin
   Remove_Branch_Iter(Node.Branch);
end Remove_Branch;


procedure Enter_Task_Frame
  (Task_Name : in V_String)
is
   use Asis;
   New_Task: Unit_Item_Link;
   Parent: Unit_Item_Link:= Current_Unit;
   --Child: Unit_Item_Link;
begin
   Num_Of_Units := Num_Of_Units + 1;
   New_Task := new Unit_Item'(Task_Name, Num_Of_Units,
                              Parent, null, null, 0, 0, 0,
                              null, null, null, null, null, null,
                              A_Single_Task_Declaration, null, null, null, 0
                              , null);

   New_Task.Brother:= Parent.Children;
   Parent.Children:= New_Task;
   Current_Unit:= New_Task;
   New_Task.Next:= Last_Unit;
   Last_Unit:= New_Task;
   if Is_Debug_Mode then
      Put_Line("enter_task_frame : " & To_S(Task_Name));
      Put_Line("parent : " & To_S(Parent.Name));
   end if;

end Enter_Task_Frame;


function Enter_Task
  (Task_Name : in V_String)
```

```
      return Unit_ID
   is
      Child: Unit_Item_Link:= Current_Unit.Children;
   begin
      while Child /= null loop
         if Equal_Insensitive(Child.Name, Task_Name) then
            Current_Unit:= Child;
            if Is_Debug_Mode then
               Put_Line("enter_task : " & To_S(Current_Unit.Name));
            end if;
            return Current_Unit.ID;
         end if;
         Child:= Child.Brother;
      end loop;

      return 0;
   end Enter_Task;


   procedure Enter_Task
     (Task_Name : in V_String)
   is
      Dummy: Unit_ID;
   begin
      Dummy:= Enter_Task(Task_Name);
   end Enter_Task;


   procedure Enter_Protected_Frame
     (Protected_Name : in V_String)
   is
      use Asis;
      New_Protected: Unit_Item_Link;
      Parent: Unit_Item_Link:= Current_Unit;
      --Child: Unit_Item_Link;
   begin
      Num_Of_Units := Num_Of_Units + 1;
      New_Protected := new Unit_Item'(Protected_Name, Num_Of_Units,
                                 Parent, null, null, 0, 0, 0,
                                 null, null, null, null, null, null,
                                 A_Protected_Body_Declaration, null, null,
                                      null, 0, null);

      New_Protected.Brother:= Parent.Children;
      Parent.Children:= New_Protected;
      Current_Unit:= New_Protected;
      New_Protected.Next:= Last_Unit;
      Last_Unit:= New_Protected;
      if Is_Debug_Mode then
         Put_Line("enter_protected_frame : " & To_S(Protected_Name));
         Put_Line("parent : " & To_S(Parent.Name));
      end if;

   end Enter_Protected_Frame;


   function Enter_Protected
     (Protected_Name : in V_String)
      return Unit_ID
   is
      Child: Unit_Item_Link:= Current_Unit.Children;
   begin
      while Child /= null loop
         if Equal_Insensitive(Child.Name, Protected_Name) then
            Current_Unit:= Child;
            if Is_Debug_Mode then
               Put_Line("enter_protected : " & To_S(Current_Unit.Name));
            end if;
            return Current_Unit.ID;
         end if;
         Child:= Child.Brother;
      end loop;
--       Yyerror("Unspecificated task body.");
      return 0;
   end Enter_Protected;


   procedure Enter_Protected
     (Protected_Name : in V_String)
   is
      Dummy: Unit_ID;
   begin
      Dummy:= Enter_Protected(Protected_Name);
   end Enter_Protected;


   procedure Enter_Unit
     (Unit_Name : in V_String; Decl_Kind : in Asis.Declaration_Kinds)
   is
      New_Unit: Unit_Item_Link;
      Parent: Unit_Item_Link:= Current_Unit;
      --Child: Unit_Item_Link;
   begin
      Num_Of_Units := Num_Of_Units + 1;
      New_Unit := new Unit_Item'(Unit_Name, Num_Of_Units,
                                 Parent, null, null, 0, 0, 0,
                                 null, null, null, null, null, null, Decl_Kind,
                                  null, null, null, 0, null);
      --Put_Line("haha");
      --Put_Line(to_s(New_Unit.Name));
      --Put_Line(to_s(Current_Unit.Name));
      New_Unit.Brother:= Parent.Children;
      Parent.Children:= New_Unit;
      Current_Unit:= New_Unit;
      New_Unit.Next:= Last_Unit;
      Last_Unit:= New_Unit;
   end Enter_Unit;
```

```
   procedure Exit_Unit is
   begin
      Current_Unit := Current_Unit.Parent;
   end Exit_Unit;

   procedure Enter_Entrydec
     (Entry_Name : in V_String;
      Elem_Id : in Gela_Ids.Id;
      Arg_S      : in Args_Stacks.Stack)
   is
      New_Entry: Entry_Item_Link;
      New_Subprogram: Subprogram_Item_Link;
      Orig_Args: Args_Stacks.Stack:= Arg_S;
      Args_Stack: Args_Stacks.Stack;
      New_Args: Args_Item_Link;

--       procedure Set_Entry(Dummy : in out Unit_Item) is
--       begin
--       null;
--       end Set_Entry;

--       procedure Set_Entry(Current_Task : in out Task_Item) is
--       begin
--       New_Entry.Next := Current_Task.Entries;
--       Current_Task.Entries := New_Entry;
--       end Set_Entry;
      use Asis;
   begin
      Clear(Args_Stack);
      while not Isempty(Orig_Args) loop
         Push(Args_Stack, Top(Orig_Args));
         Remove(Orig_Args);
      end loop;
      Num_Of_Entries:= Num_Of_Entries + 1;
      New_Entry:= new Entry_Item;
      New_Entry.Name:= Entry_Name;
      New_Entry.Decl_Id := Elem_Id;
      New_Entry.Task_Link := Current_Unit;
      New_Entry.Number:= Num_Of_Entries;
      New_Entry.Nodes:= null;
      Entry_Decl_Id_List.Set_Node(Entry_Decl_List, Elem_Id, New_Entry);
--       New_Entry.Next:= Current_Unit.Entries
--       Current_Unit.Entries:= New_Entry;
--      Set_Entry(Current_Unit.all);
--      if Current_Unit.all in Task_Item then
--       declare
--          Current_Task : Task_Item := Task_Item(Current_Unit.all);
--       begin
      New_Entry.Next := Current_Unit.Entries;
      Current_Unit.Entries := New_Entry;
--       Put_Line("entry_decl:");
--       Put_Line(To_S(Current_Unit.Name));
--          Current_Unit.all := Current_Task;
--       end;
--      end if;

      if Current_Unit.Decl_Kind = A_Protected_Body_Declaration then
         New_Subprogram:= new Subprogram_Item;
         New_Subprogram.Name:= Entry_Name;
         --       New_Entry.Id := Elem_Id;
         Subprogram_Decl_Id_List.Set_Node(Subprogram_Decl_List, Elem_Id, New_Subprogram);
         New_Subprogram.Belong_Unit := Current_Unit;
         New_Subprogram.Caller := null;
         New_Subprogram.Callee := 0;
         New_Subprogram.Next:= Current_Unit.Subprograms;
         Current_Unit.Subprograms := New_Subprogram;
      end if;

      while not Isempty(Args_Stack) loop
         New_Args:= new Args_Item;
         New_Args.Name:= Top(Args_Stack).Name;
         New_Args.Mode:= Top(Args_Stack).Mode;
         New_Args.Position := Top(Args_Stack).Position;
         New_Args.Next:= New_Entry.Args;
         New_Entry.Args:= New_Args;
         if Current_Unit.Decl_Kind = A_Protected_Body_Declaration then
            New_Subprogram.Args := New_Args;
         end if;
         Remove(Args_Stack);
      end loop;


   end Enter_Entrydec;


   procedure Enter_Entrycall
     (Entry_Decl : in Entry_Item_Link;
      Node_Num : in Nodenum_T)
   is
      Node_Link : Node_Item_Link := Search_Node(Node_Num);
      New_Call : Entry_Call_Link;
   begin
      New_Call := new Entry_Calls;
      New_Call.Next := Current_Unit.Entry_Calls;
      New_Call.Task_Called := Entry_Decl.Task_Link;
      New_Call.Entry_Called := Entry_Decl;
      New_Call.Node := Node_Link;
      Current_Unit.Entry_Calls := New_Call;
      Node_Link.Send:= new Dug_Channel_Item'(Entry_Decl.Task_Link, Entry_Decl, False, Node_Link.Send);
--       Previous_Call:= Dug_Channel_Item'(Entry_Decl.Task_Link, Entry_Decl, True);
      Nextcall_Stacks.Push(Nextcall_Stack, (Node_Num, new Dug_Channel_Item'(Entry_Decl.Task_Link, Entry_Decl, True, null), null, False));

   end Enter_Entrycall;

   procedure Enter_RequeueEntry
     (Entry_Decl : in Entry_Item_Link;
      Node_Num : in Nodenum_T;
      Out_Node : in Nodenum_T)
   is
```

```
      Node_Link : Node_Item_Link := Search_Node(Node_Num);
      Out_Node_Link : Node_Item_Link := Search_Node(Out_Node);
      New_Call : Entry_Call_Link;
   begin
      New_Call := new Entry_Calls;
      New_Call.Next := Current_Unit.Entry_Calls;
      New_Call.Task_Called := Entry_Decl.Task_Link;
      New_Call.Entry_Called := Entry_Decl;
      New_Call.Node := Node_Link;
      Current_Unit.Entry_Calls := New_Call;
      Node_Link.Send:= new Dug_Channel_Item'(Entry_Decl.Task_Link, Entry_Decl, False, Node_Link.Send);
      Out_Node_Link.Receive := new Dug_Channel_Item'(Entry_Decl.Task_Link, Entry_Decl, True, Out_Node_Link.Receive);
--      Previous_Call:= Dug_Channel_Item'(Entry_Decl.Task_Link, Entry_Decl, True);
   end Enter_RequeueEntry;


   procedure Enter_Accept
     (Elem_ID : in Gela_Ids.Id;
      Node_Num : in Nodenum_T)
   is
      New_Acc_Node: Accept_Node_Link;
      Entry_Link: Entry_Item_Link := Entry_Decl_Id_List.Search_Node(Entry_Decl_List, Elem_ID);
      Node_Link: Node_Item_Link:= Search_Node(Node_Num);
   begin
      New_Acc_Node:= new Accept_Node;
      New_Acc_Node.Start_Node:= Node_Num;
      Current_Accept:= Entry_Link;
      New_Acc_Node.Next:= Entry_Link.Nodes;
      Entry_Link.Nodes:= New_Acc_Node;
      Node_Link.Receive:= new Dug_Channel_Item'(Current_Unit, Entry_Link, False, Node_Link.Receive);
   end Enter_Accept;


   procedure Exit_Accept
     (Elem_ID : in Gela_Ids.Id;
      Node_Num : in Nodenum_T)
   is
      Entry_Link: Entry_Item_Link := Entry_Decl_Id_List.Search_Node(Entry_Decl_List, Elem_ID);
      Node_Link: Node_Item_Link:= Search_Node(Node_Num);
   begin
      if Node_Num /= 0 then
         Current_Accept.Nodes.End_Node:= Node_Num;
         Node_Link.Send:= new Dug_Channel_Item'(Current_Unit, Entry_Link, True, Node_Link.Send);
      else
         Nextcall_Stacks.Push(Nextcall_Stack, (Num_Of_Nodes, new Dug_Channel_Item'(Current_Unit, Entry_Link, True, null), null, True));
      end if;
      Current_Accept:= null; -- some changes are needed for nested accept.
   end Exit_Accept;


   procedure Enter_Proceduredec
     (Procedure_Name : in V_String;
      Elem_Id : in Gela_Ids.Id;
      Arg_S      : in Args_Stacks.Stack)
   is
      New_Subprogram: Subprogram_Item_Link;
      Orig_Args: Args_Stacks.Stack:= Arg_S;
--      Args_Stack: Args_Stacks.Stack;
      New_Args: Args_Item_Link;

   begin
--       Clear(Args_Stack);
--       while not Isempty(Orig_Args) loop
--          Push(Args_Stack, Top(Orig_Args));
--          Remove(Orig_Args);
--       end loop;
      New_Subprogram:= new Subprogram_Item;
      New_Subprogram.Name:= Procedure_Name;
--      New_Entry.Id := Elem_Id;
      Subprogram_Decl_Id_List.Set_Node(Subprogram_Decl_List, Elem_Id, New_Subprogram);
      New_Subprogram.Belong_Unit := Current_Unit;
      New_Subprogram.Caller := null;
      New_Subprogram.Callee := 0;
      New_Subprogram.Next:= Current_Unit.Subprograms;
      Current_Unit.Subprograms := New_Subprogram;
      if Is_Debug_Mode then
         Put_Line("enter_proceduredec:");
      end if;
      while not Isempty(Orig_Args) loop
         New_Args:= new Args_Item;
         New_Args.Name:= Top(Orig_Args).Name;
         New_Args.Mode:= Top(Orig_Args).Mode;
         New_Args.Position := Top(Orig_Args).Position;
         if Is_Debug_Mode then
            Put_Line("pos :" & Natural'Image(Top(Orig_Args).Position));
         end if;

         New_Args.Next:= New_Subprogram.Args;
         New_Subprogram.Args:= New_Args;
         Remove(Orig_Args);
      end loop;
   end Enter_Proceduredec;


   procedure Enter_Functiondec
     (Function_Name : in V_String;
      Elem_Id : in Gela_Ids.Id;
      Arg_S      : in Args_Stacks.Stack;
      Function_Varnum : in Varnum_T)
--       Type_Ident : in V_String;
--      Is_Access : in Boolean)
   is
      New_Subprogram: Subprogram_Item_Link;
      Orig_Args: Args_Stacks.Stack:= Arg_S;
      New_Args: Args_Item_Link;
   begin
      New_Subprogram:= new Subprogram_Item;
      New_Subprogram.Name:= Function_Name;
--      New_Entry.Id := Elem_Id;
```

```
      Subprogram_Decl_Id_List.Set_Node(Subprogram_Decl_List, Elem_Id, New_Subprogram);
      New_Subprogram.Belong_Unit := Current_Unit;
      New_Subprogram.Caller := null;
      New_Subprogram.Callee := 0;
--       New_Subprogram.Type_Ident := Type_Ident;
--       New_Subprogram.Is_Access := Is_Access;
      New_Subprogram.Function_Varnum := Function_Varnum;
      New_Subprogram.Next:= Current_Unit.Subprograms;
      Current_Unit.Subprograms := New_Subprogram;

      while not Isempty(Orig_Args) loop
         New_Args:= new Args_Item;
         New_Args.Name:= Top(Orig_Args).Name;
         New_Args.Mode:= Top(Orig_Args).Mode;
         New_Args.Position := Top(Orig_Args).Position;
         New_Args.Next:= New_Subprogram.Args;
         New_Subprogram.Args:= New_Args;
         Remove(Orig_Args);
      end loop;
   end Enter_Functiondec;


   function Search_Entry_Decl(Entry_Name : in V_string) return Gela_Ids.Id is
      Entry_Link : Entry_Item_Link := Current_Unit.Parent.Entries;
   begin
      if Is_Debug_Mode then
         Put_Line("entry_search:");
         Put_Line(To_S(Current_Unit.Parent.Name));
      end if;
      while Entry_Link /= null loop
         if Entry_Link.Name = Entry_Name then
            return Entry_Link.Decl_Id;
         end if;
         Entry_Link := Entry_Link.Next;
      end loop;
      if Is_Debug_Mode then
         Put_Line("Cannot find an entry declaration corresponding to an entry body! : " &To_S(Entry_Name));
      end if;
      return Gela_Ids.Nil_Id ;
   end Search_Entry_Decl;


   procedure First_Stmt
   is
   begin
      if Current_Unit.First_Stmt = 0 then
         Current_Unit.First_Stmt:= Num_Of_Nodes+1;
      end if;
   end First_Stmt;


   function Search_Unit_Item
     (Unit_Num : Unit_ID)
      return Unit_Item_Link
   is
      Search_Unit: Unit_Item_Link:= Last_Unit;
   begin
      while Search_Unit /= null loop
         if Search_Unit.ID = Unit_Num then
            return Search_Unit;
         end if;
         Search_Unit:= Search_Unit.Next;
      end loop;

       return null;
   end Search_Unit_Item;


   function Search_Args_Item(Call_Args : Args_Item_Link ; Arg_Name : V_String)
                             return Args_Item_Link is
      Search_Args : Args_Item_Link := Call_Args;
   begin
      while Search_Args /= null loop
         if Equal_Insensitive(Search_Args.Name, Arg_Name) then
            return Search_Args;
         end if;
         Search_Args := Search_Args.Next;
      end loop;
      return null;
   end Search_Args_Item;


   function Add_Call_Branch
     (Tail : in Nodenum_T;
      Head : in Nodenum_T) return Call_Branch_Item_Link
   is
      Node : Node_Item_Link:= Search_Node(Tail);
      New_Branch : Call_Branch_Item_Link:= new Call_Branch_Item;
   begin
      New_Branch.Head:= Head;
      New_Branch.Next:= Node.Calls;
      Node.Calls := New_Branch;
      return New_Branch;
   end Add_Call_Branch;


   procedure Add_Stacked_Call_Branch(Subprog : in out Subprogram_Item) is
      use Call_Branch_Stack;
   begin
      while Subprog.Caller /= null loop
--       Add_Call_Branch(Top(Subprog.Caller), Subprog.Callee);
         Call_Branch_Stack.Top(Subprog.Caller).Head := Subprog.Callee;
         Call_Branch_Stack.Remove(Subprog.Caller);
      end loop;
   end Add_Stacked_Call_Branch;


   procedure Add_Actual_In(Actual_In : in out Dug_Var_Group_Item_Link;
                           Position : in Natural;
```

```
                     Is_An_In_Out_Mode : in Boolean) is
   use Vstring_Stacks;
   use Varnum_Stacks;
   use Var_Item_Stacks;

   Keep_Vstring_Stack : Vstring_Stacks.Stack;
   Keep_Var_Stack : Varnum_Stacks.Stack;
   Keep_Var_Item_Stack : Var_Item_Stacks.Stack;


   procedure Use_Variable
     (Var_Name : in V_String) is
      New_Var: Dug_Var_Item_Link;
      Var_Link_Found: Var_Item_Link;
      Trace: Dug_Var_Item_Link;
   begin
      Var_Link_Found:= Find_Visible_Var_Link(Var_Name, True);
      if Var_Link_Found = null then
         return;
      end if;
      Trace:= Actual_In.Contents;
      while Trace /= null loop
         if Trace.Number = Var_Link_Found.Number then
            return;                         -- duplicated
         end if;
         Trace:= Trace.Next;
      end loop;
      New_Var:= new Dug_Var_Item;
      New_Var.Number:= Var_Link_Found.Number;
      New_Var.Next:= Actual_In.Contents;
      Actual_In.Contents := New_Var;
   end Use_Variable;


   -- access type
   procedure Use_Variable
     (Var_num : in Varnum_T;
      Is_Normalized : in Boolean := False;
      Normal_Form : in V_String := Null_Str) is
      Var_Link : Var_Item_Link := Search_Var_Item(Var_Num);
      New_Var: Dug_Var_Item_Link;
      Trace: Dug_Var_Item_Link;
   begin
      if Var_Num /= 0 then
         Trace:= Actual_In.Contents;
         while Trace /= null loop
            if Trace.Number = Var_Num then
               return;                      -- duplicated
            end if;
            Trace:= Trace.Next;
         end loop;
         New_Var:= new Dug_Var_Item;
         New_Var.Number:= Var_Num;
         New_Var.Is_Normalized := Is_Normalized;
         New_Var.Normalized_Form := Normal_Form;
         New_Var.Next:= Actual_In.Contents;
         Actual_In.Contents := New_Var;
      end if;

   end Use_Variable;

begin
   if Actual_In = null then
      Actual_In := new Dug_Var_Group_Item;
      Actual_In.Position := Position;

      while not isempty(vstring_stack) loop
         use_variable(top(vstring_stack));
         Push(Keep_Vstring_Stack, Top(Vstring_Stack));
         remove(vstring_stack);
      end loop;

      while not isempty(var_stack) loop
         use_variable(top(var_stack));
         Push(Keep_Var_Stack, Top(Var_Stack));
         remove(var_stack);
      end loop;

      while not isempty(Var_item_stack) loop
         use_variable(top(Var_Item_stack).number, Top(Var_Item_Stack).Is_Normalized, Top(Var_Item_Stack).Normalized_Form);
         Push(Keep_Var_Item_Stack, Top(Var_Item_Stack));
         remove(Var_Item_stack);
      end loop;

      Actual_In.Next := null;
      if Is_An_In_Out_Mode then
         Vstring_Stack := Keep_Vstring_Stack;
         Var_Stack := Keep_Var_Stack;
         Var_Item_Stack := Keep_Var_Item_Stack;
      end if;
   else
      Add_Actual_In(Actual_In.Next, Position, Is_An_In_Out_Mode);
   end if;
end Add_Actual_In;


procedure Add_Actual_Out(Actual_Out : in out Dug_Var_Group_Item_Link;
                         Position : in Natural;
                        Is_An_Out_Mode : in Boolean) is


   procedure Assign_Variable
     (Var_Name : in V_String) is
      Var_Link: Var_Item_Link;
   begin
      Var_Link:= Find_Visible_Var_Link(Var_Name, False);
      if Var_Link = null then
         --         Yyerror("Identifier undefined.");
         return;
```

```
         else
            Actual_Out.Contents := new Dug_Var_Item;
            Actual_Out.Contents.Number := Var_Link.Number;
         end if;
      end Assign_Variable;


      procedure Assign_Variable
        (Var_Num  : in Varnum_T;
         Is_Normalized : in Boolean := False;
         Normal_Form : in V_String := Null_Str)
      is
      begin
         Actual_Out.Contents := new Dug_Var_Item;
         Actual_Out.Contents.Number:= Var_Num;
         Actual_Out.Contents.Is_Normalized := Is_Normalized;
         Actual_Out.Contents.Normalized_Form := Normal_Form;
      end Assign_Variable;

      use Varnum_Stacks;
      use Var_Item_Stacks;
   begin
      if Actual_Out = null then
         Actual_Out := new Dug_Var_Group_Item;
         Actual_Out.Position := Position;
         Actual_Out.Contents := new Dug_Var_Item;
         -- actual_out
         if not Isempty(Var_Stack) then
            Assign_Variable(top(Var_stack));
--          if Is_An_Out_Mode then
               Remove(Var_Stack);
--          end if;
         elsif not Isempty(Var_Item_Stack) then
            assign_variable(top(Var_Item_stack).number, Top(Var_Item_Stack).Is_Normalized, Top(Var_Item_Stack).Normalized_Form);
--          if Is_An_Out_Mode then
               Remove(Var_Item_Stack);
--          end if;
         else
            Actual_Out.Contents.Number := 0;
--          Put_Line("warning: no variable in actual-out.");
         end if;
         Actual_Out.Next := null;
      else
         Add_Actual_Out(Actual_Out.Next, Position, Is_An_Out_Mode);
      end if;
   end Add_Actual_Out;

   procedure Search_Actual_Contents(Position : in Natural;
                                    Actual : in Dug_Var_Group_Item_Link;
                                    Contents : out Dug_Var_Item_Link;
                                    Is_Last : out Boolean) is
      Iter : Dug_Var_Group_Item_Link := Actual;
      Tmp_Is_Last : Boolean := True;
   begin
      Contents := new Dug_Var_Item;
      Contents.all := Not_Found;
      while Iter /= null loop
         if Iter.Position > Position then
            Tmp_Is_Last := False;
         elsif Iter.Position = Position then
            Contents := Iter.Contents;
         end if;
         Iter := Iter.Next;
      end loop;
      Is_Last := Tmp_Is_Last;
   end Search_Actual_Contents;

   procedure Add_Left_Actual_Label(Args : Args_Item_Link;
                                   Call_Item : Call_Branch_Item_Link) is
      use Asis;
      Tmp_Contents : Dug_Var_Item_Link := null;
      Tmp_Is_Last : Boolean := False;
   begin
      if Args = null then
         return;
      end if;
      if Is_Debug_Mode then
         Put_Line("add_left_actual_label(pos) : " & Natural'Image(Args.Position));
      end if;
      if Args.Mode /= An_Out_Mode then
         Search_Actual_Contents(Args.Position, Call_Item.Actual_In,
                                Tmp_Contents, Tmp_Is_Last);
         if Tmp_Contents /= null and then Tmp_Contents.all = Not_Found then
            Add_Actual_In(Call_Item.Actual_In,
                          Args.Position,
                          False);
         end if;
      end if;
      if Args.Mode = An_Out_Mode or Args.Mode = An_In_Out_Mode then
         Search_Actual_Contents(Args.Position, Call_Item.Actual_Out,
                                Tmp_Contents, Tmp_Is_Last);
         if Tmp_Contents /= null and then Tmp_Contents.all = Not_Found then
            Add_Actual_Out(Call_Item.Actual_Out,
                          Args.Position,
                          False);
         end if;
      end if;
      Add_Left_Actual_Label(Args.Next, Call_Item);
   end Add_Left_Actual_Label;

   procedure Add_Formal_In
     (Var_Num  : in Varnum_T;
      Node_Num : in Nodenum_T)
   is
      Var_Link : Var_Item_Link := Search_Var_Item(Var_Num);
      Node_Link: Node_Item_Link:= Search_Node(Node_Num);
      New_Var: Dug_Var_Item_Link;
      Trace : Dug_Var_Item_Link;
   begin
```

```
      New_Var:= new Dug_Var_Item;
      New_Var.Number:= Var_Num;
      New_Var.Next:= null;
      if Node_Link.Formal_In = null then
         Node_Link.Formal_In := New_Var;
      else
         Trace := Node_Link.Formal_In;
         while Trace.Next /= null loop
            Trace := Trace.Next;
         end loop;
         Trace.Next := New_Var;
      end if;
   end Add_Formal_In;


   procedure Add_Formal_Out
     (Var_Num  : in Varnum_T;
      Node_Num : in Nodenum_T)
   is
      Var_Link : Var_Item_Link := Search_Var_Item(Var_Num);
      Node_Link: Node_Item_Link:= Search_Node(Node_Num);
      New_Var: Dug_Var_Item_Link;
      Trace : Dug_Var_Item_Link;
   begin
      New_Var:= new Dug_Var_Item;
      New_Var.Number:= Var_Num;
      New_Var.Next:= null;
      if Node_Link.Formal_Out = null then
         Node_Link.Formal_Out := New_Var;
      else
         Trace := Node_Link.Formal_Out;
         while Trace.Next /= null loop
            Trace := Trace.Next;
         end loop;
         Trace.Next := New_Var;
      end if;
   end Add_Formal_Out;

   procedure Add_Arg_Branch
     (Tail : in Nodenum_T;
      Head : in Nodenum_T)
   is
      Node : Node_Item_Link:= Search_Node(Tail);
      New_Branch : Branch_Item_Link:= new Branch_Item;
   begin
      New_Branch.Head:= Head;
      New_Branch.Next:= Node.Args;
      Node.Args := New_Branch;
   end Add_Arg_Branch;


   procedure Add_Top_Arg_Branch
     (Head : in Nodenum_T)
   is
      Node : Node_Item_Link:= Search_Node(Current_Unit.First_Stmt);
      New_Branch : Branch_Item_Link:= new Branch_Item;
   begin
      New_Branch.Head:= Head;
      New_Branch.Next:= Node.Top_Args;
      Node.Top_Args := New_Branch;
   end Add_Top_Arg_Branch;


   procedure Add_Stacked_Arg_Branch_In
     (Arg_Item : in out Args_Item) is
      use Nodenum_Stack;
   begin
      while Arg_Item.Actual_In /= null loop
         Add_Arg_Branch(Top(Arg_Item.Actual_In), Arg_Item.Formal_In);
         Remove(Arg_Item.Actual_In);
      end loop;
   end Add_Stacked_Arg_Branch_In;


   procedure Add_Stacked_Arg_Branch_Out
     (Arg_Item : in out Args_Item) is
      use Nodenum_Stack;
   begin
      while Arg_Item.Actual_Out /= null loop
         Add_Arg_Branch(Arg_Item.Formal_Out , Top(Arg_Item.Actual_Out));
         Remove(Arg_Item.Actual_Out);
      end loop;
   end Add_Stacked_Arg_Branch_Out;

   procedure Set_Fork_Point is
      use Asis;
      Last_Decl : Node_Item_Link;
      Send_Node : Nodenum_T;
   begin
      Current_Unit.Fork_Point := Num_Of_Nodes;
      if Is_Debug_Mode then
         Put_Line(To_S(Current_Unit.Name) & "'s fork point is " & Nodenum_T'Image(Num_Of_Nodes));
      end if;
      if Current_Unit.Decl_Kind = A_Single_Task_Declaration then
         if Num_Of_Nodes  < Current_Unit.First_Stmt then
            Send_Node := Current_Unit.First_Stmt;
         else
            Send_Node := Num_Of_Nodes ;
         end if;
--       First_Send := Dug_Channel_Item'(Current_Unit.Parent, null, False);
         Last_Decl := Search_Node(Send_Node);
         Last_Decl.Send := new Dug_Channel_Item'(Current_Unit.Parent, null, False, Last_Decl.Send);

      end if;
   end Set_Fork_Point;

   procedure Add_P_Branch_To_Child
   is
      Child: Unit_Item_Link:= Current_Unit.Children;
```

```
      Task_Type : Task_Type_Item_Link := Current_Unit.Decl_By_Task_Type;
      New_Branch: Branch_Item_Link;
      use ASIS;
      Fork_Node : Node_Item_Link := Search_Node(Current_Unit.Fork_Point);
      Last_Decl : Node_Item_Link;
   begin
      if Is_Debug_Mode then
         Put_Line("I am " & To_S(Current_Unit.Name));
      end if;
--       if Child = null then
--          return;
--       end if;
      while Child /= null loop
         if Is_Debug_Mode then
            Put("o");
         end if;
         if Child.Decl_Kind = A_Single_Task_Declaration then
            New_Branch:=
             new Branch_Item'(Search_Node(Child.First_Stmt).Number,
                              0,
                              Fork_Node.Forks);
            Fork_Node.Forks:= New_Branch;
            if Is_Debug_Mode then
               Put_Line("fork from " & To_S(Current_Unit.Name) & " to " & To_S(Child.Name));
            end if;
         end if;
         Child:= Child.Brother;
      end loop;

      while Task_Type /= null loop
         if not Task_Type.Is_Access_Type then
            if Task_Type.Info.Belong_Unit.First_Stmt /= 0 then
               New_Branch :=
                new Branch_Item'(Search_Node(Task_Type.Info.Belong_Unit.First_Stmt).Number,
                                 0,
                                 Fork_Node.Forks);
               Fork_Node.Forks:= New_Branch;
            else
               Nodenum_Stack.Push(Task_Type.Info.Fork_Source, Current_Unit.Fork_Point);
            end if;
            if Task_Type.Info.Belong_Unit.Fork_Point /= 0 then
               Last_Decl := Search_Node(Task_Type.Info.Belong_Unit.Fork_Point);
               Last_Decl.Send :=
                new Dug_Channel_Item'(Current_Unit, null, False, Last_Decl.Send);
            else
               Task_Type.Info.Parent_Sync := new Dug_Channel_Item'(Current_Unit, null, False, Task_Type.Info.Parent_Sync);
            end if;
         end if;
         Task_Type := Task_Type.Next;
      end loop;

--      Previous_Call:= Dug_Channel_Item'(Current_Unit, null, False);
   end Add_P_Branch_To_Child;


   procedure Add_Fork_Branch(Source, Dest : in Nodenum_T) is
      Fork_Node : Node_Item_Link := Search_Node(Source);
      New_Branch : Branch_Item_Link;
   begin
      New_Branch := new Branch_Item'(Dest, 0, Fork_Node.Forks);
      Fork_Node.Forks := New_Branch;
   end Add_Fork_Branch;


   procedure Add_Join_Branch(Source, Dest : in Nodenum_T) is
      Join_Node : Node_Item_Link := Search_Node(Source);
      New_Branch : Branch_Item_Link;
   begin
      New_Branch := new Branch_Item'(Dest, 0, Join_Node.Joins);
      Join_Node.Joins := New_Branch;
   end Add_Join_Branch;


   procedure Add_P_Branch_From_Child
   is
      Child: Unit_Item_Link:= Current_Unit.Children;
      Task_Type : Task_Type_Item_Link := Current_Unit.Decl_By_Task_Type;
      New_Branch : Branch_Item_Link;
      procedure Add_Iter(Tmp_Child : in Unit_Item_Link) is
         A_Child : Unit_Item_Link := Tmp_Child;
         New_Branch: Branch_Item_Link;
         use Asis;
      begin
         if A_Child = null then
            return;
         end if;
         while A_Child /= null loop
            if A_Child.Decl_Kind = A_Single_Task_Declaration then
               New_Branch:= new Branch_Item'(Current_Unit.Nodes.Number,
                                             0,
                                             A_Child.Nodes.Joins);
               A_Child.Nodes.Joins:= New_Branch;
            elsif A_Child.Decl_Kind = A_Package_Declaration or
              A_Child.Decl_Kind = A_Generic_Package_Declaration then
               Add_Iter(A_Child.Children);
            end if;
            A_Child:= A_Child.Brother;
         end loop;
      end Add_Iter;
   begin
      Add_Iter(Child);
      while Task_Type /= null loop
         if Task_Type.Info.Belong_Unit.Nodes /= null then
            New_Branch :=
             new Branch_Item'(Current_Unit.Nodes.Number,
                              0,
                              Task_Type.Info.Belong_Unit.Nodes.Joins);
            Task_Type.Info.Belong_Unit.Nodes.Joins := New_Branch;
         else
```

```
         Nodenum_Stack.Push(Task_Type.Info.Join_Dest, Current_Unit.Nodes.Number);
      end if;
      Task_Type := Task_Type.Next;
   end loop;

end Add_P_Branch_From_Child;


procedure Add_Stacked_P_Branch(Task_Type : in Task_Type_Info_Link) is
   use Nodenum_Stack;
   Last_Decl : Node_Item_Link;
begin
   if Task_Type = null then
      return;
   end if;
   while not Isempty(Task_Type.Fork_Source) loop
      Add_Fork_Branch(Top(Task_Type.Fork_Source), Current_Unit.First_Stmt);
      Remove(Task_Type.Fork_Source);
   end loop;
   while not Isempty(Task_Type.Join_Dest) loop
      Add_Join_Branch(Current_Unit.Nodes.Number, Top(Task_Type.Join_Dest));
      Remove(Task_Type.Join_Dest);
   end loop;
   Last_Decl := Search_Node(Current_Unit.Fork_Point);
   Last_Decl.Send := Task_Type.Parent_Sync;
end Add_Stacked_P_Branch;


procedure Add_P_Branch_From_Allocate(Task_Type : in Task_Type_Info_Link) is
   Last_Decl : Node_Item_Link;
   New_Send : Dug_Channel_Item_Link;
begin
   if Task_Type.Belong_Unit.First_Stmt /= 0 then
      Add_Fork_Branch(Num_Of_Nodes, Task_Type.Belong_Unit.First_Stmt);
   else
      Nodenum_Stack.Push(Task_Type.Fork_Source, Num_Of_Nodes);
   end if;
   New_Send :=
    new Dug_Channel_Item'(Current_Unit,
                          new Entry_Item'(To_V(Unique_Identifier), 0,
                                          Gela_Ids.Nil_Id, null,
                                          null, null, null, False),
                          False, null);

   if Task_Type.Belong_Unit.Fork_Point /= 0 then
      Last_Decl := Search_Node(Task_Type.Belong_Unit.Fork_Point);
      New_Send.Next := Last_Decl.Send;
      Last_Decl.Send := New_Send;
   else
      New_Send.Next := Task_Type.Parent_Sync;
      Task_Type.Parent_Sync := New_Send;
   end if;
   Nextcall_Stacks.Push(Nextcall_Stack,
                        (Num_Of_Nodes, New_Send, null, False));
end Add_P_Branch_From_Allocate;


function Unique_Identifier return String is
   package Int_IO is new Ada.Text_IO.Integer_IO(Integer);
   use Int_IO;
   Result : String(1..5);
begin
   Put(Result, IdentLast);
   IdentLast := IdentLast + 1;
   for I in Result'Range loop
      if Result(I) = ' ' then
         Result(I) := '0';
      end if;
   end loop;
   return "DECL_" & Result;
end Unique_Identifier;


function Enter_Variable
  (Var_Name : in V_String; Type_Ident : in V_String;
                           Is_Access : in Boolean)
   return Varnum_T
is
   New_Var: Var_Item_Link;
   --Temp: Var_Item_Link;
   Is_Protected : Boolean := False;
   use Asis;
begin
   if Current_Unit.Decl_Kind = A_Protected_Body_Declaration then
      Is_Protected := True;
      if Is_Debug_Mode then
         Put_Line("*** protected variable : " &  To_S(Var_Name));
      end if;
   end if;
   Num_Of_Variables:= Num_Of_Variables + 1;
   New_Var:= new Var_Item'(Current_Unit, Current_Routine,
                           Var_Name, Num_Of_Variables,
                           Type_Ident, Is_Access, Is_Protected,
                           Current_Unit.Vars, Variable_Table_Tail);
   Current_Unit.Vars:= New_Var;
   Variable_Table_Tail:= New_Var;
   return Num_Of_Variables;
end Enter_Variable;

function Enter_variable
  (Dname : in Asis.Defining_Name; Type_Ident : in V_String;
                                  Is_Access : in Boolean)
   return Varnum_T
is
   use Asis;
   Temp_Varnum : Varnum_T;
begin
   Temp_Varnum := Enter_Variable(To_V(Asis.Declarations.Defining_Name_Image(Dname)), Type_Ident, Is_Access);
   Varnum_List.Set_Node(Defining_List, Gela_Ids.Create_Id(Dname), Temp_Varnum);
```

```
      return Temp_Varnum;
   end Enter_Variable;



   procedure Enter_Variable
     (Var_Name : in V_String; Type_Ident : in V_String;
                              Is_Access : in Boolean)
   is
      Dummy: Varnum_T;
   begin
      Dummy:= Enter_Variable(Var_Name, Type_Ident, Is_Access);
   end Enter_Variable;


   procedure Remove_Variable
   is
   begin
      Current_Unit.Vars:= Current_Unit.Vars.Next;
   end Remove_Variable;


   procedure Assign_Variable
     (Var_Name : in V_String;
      Node_Num : in Nodenum_T)
   is
      Node_Link: Node_Item_Link:= Search_Node(Node_Num);
      Var_Link: Var_Item_Link;
      New_Var: Dug_Var_Item_Link;
   begin
      Var_Link:= Find_Visible_Var_Link(Var_Name, False);
      if Var_Link = null then
--         error("Identifier undefined.");
         return;
      else
         New_Var:= new Dug_Var_Item;
         New_Var.Number:= Var_Link.Number;
         New_Var.Next:= Node_Link.Assign;
         Node_Link.Assign:= New_Var;
      end if;
   end Assign_Variable;


   procedure Assign_Variable
     (Var_Num  : in Varnum_T;
      Node_Num : in Nodenum_T;
      Is_Normalized : in Boolean := False;
      Normal_Form : in V_String := Null_Str)
   is
      Var_Link : Var_Item_Link := Search_Var_Item(Var_Num);
      Node_Link: Node_Item_Link:= Search_Node(Node_Num);
      New_Var: Dug_Var_Item_Link;
      Last_Unit : Unit_Item_Link := null;
      Trace: Dug_Var_Item_Link;
      use Asis;
   begin
      New_Var:= new Dug_Var_Item;
      New_Var.Number:= Var_Num;
      New_Var.Is_Normalized := Is_Normalized;
      New_Var.Normalized_Form := Normal_Form;
      New_Var.Next:= Node_Link.Assign;
      Node_Link.Assign:= New_Var;
      if Var_Link.Unit_Link.Decl_Kind = A_Protected_Body_Declaration then
         Last_Unit := Current_Unit;
         while Last_Unit.Parent.ID /= Var_Link.Unit_Link.ID loop
--          Put_line(To_S(Last_Unit.Name) & ":" & To_S(Var_Link.Unit_Link.name));
--          Put_line(Unit_ID'Image(Last_Unit.ID) & ":" & Unit_ID'Image(Var_Link.Unit_Link.ID));
            Last_Unit := Last_Unit.Parent;
            if Last_Unit.ID = Main_Task.ID then
--             Put_Line("An assignment of a protected variable is invalid!");
               return;
            end if;
         end loop;
         Trace := Last_Unit.Protected_Def;
         while Trace /= null loop
            exit when Trace.Number = Var_Num;
            Trace := Trace.Next;
         end loop;
         if Trace = null then
            New_Var:= new Dug_Var_Item;
            New_Var.Number:= Var_Num;
            New_Var.Is_Normalized := Is_Normalized;
            New_Var.Normalized_Form := Normal_Form;
            New_Var.Next:= Last_Unit.Protected_Def;
            Last_Unit.Protected_Def := New_Var;
         end if;
      end if;
   end Assign_Variable;


   procedure Use_Variable
     (Var_Name : in V_String;
      Node_Num : in Nodenum_T)
   is
      Node_Link: Node_Item_Link:= Search_Node(Node_Num);
      New_Var: Dug_Var_Item_Link;
      Var_Link_Found: Var_Item_Link;
      Trace: Dug_Var_Item_Link;
   begin
      Var_Link_Found:= Find_Visible_Var_Link(Var_Name, True);
      if Var_Link_Found = null then
         return;
      end if;
      Trace:= Node_Link.Refer;
      while Trace /= null loop
         if Trace.Number = Var_Link_Found.Number then
            return;                       -- duplicated
         end if;
         Trace:= Trace.Next;
```

```
      end loop;
      New_Var:= new Dug_Var_Item;
      New_Var.Number:= Var_Link_Found.Number;
      New_Var.Next:= Node_Link.Refer;
      Node_Link.Refer:= New_Var;
   end Use_Variable;


   procedure Use_Variable
     (Var_num : in Varnum_T;
      Node_Num : in Nodenum_T;
      Is_Normalized : in Boolean := False;
      Normal_Form : in V_String := Null_Str)
   is
      Var_Link : Var_Item_Link := Search_Var_Item(Var_Num);
      Node_Link: Node_Item_Link:= Search_Node(Node_Num);
      New_Var: Dug_Var_Item_Link;
      Last_Unit : Unit_Item_Link := null;
      Trace: Dug_Var_Item_Link;
      use Asis;
   begin
      if Var_Num /= 0 then
         Trace:= Node_Link.Refer;
         while Trace /= null loop
            if Trace.Number = Var_Num then
               return;                       -- duplicated
            end if;
            Trace:= Trace.Next;
         end loop;
         New_Var:= new Dug_Var_Item;
         New_Var.Number:= Var_Num;
         New_Var.Is_Normalized := Is_Normalized;
         New_Var.Normalized_Form := Normal_Form;
         New_Var.Next:= Node_Link.Refer;
         Node_Link.Refer := New_Var;
      end if;

      if Var_Link.Unit_Link.Decl_Kind = A_Protected_Body_Declaration then
         Last_Unit := Current_Unit;
         while Last_Unit.Parent.ID /= Var_Link.Unit_Link.ID loop
            Last_Unit := Last_Unit.Parent;
            if Last_Unit.ID = Main_Task.ID then
               Put_Line("warning : A reference of a protected variable is invalid!");
               return;
            end if;
         end loop;
         Trace := Last_Unit.Protected_Use;
         while Trace /= null loop
            exit when Trace.Number = Var_Num;
            Trace := Trace.Next;
         end loop;
         if Trace = null then
            New_Var:= new Dug_Var_Item;
            New_Var.Number:= Var_Num;
            New_Var.Next:= Last_Unit.Protected_Use;
            Last_Unit.Protected_Use := New_Var;
         end if;
      end if;

   end Use_Variable;



   procedure Set_Info(Id : in Gela_Ids.Id; Info : in Element_Info) is
   begin

      Element_Id_List.Set_Node(Info_List, Id, Info);

   end Set_Info;


   procedure Set_Info(Elem : in Asis.Element ; Info : in Element_Info) is

   begin

        Ada.Wide_Text_IO.Put(Asis.Text.Element_Image(Elem));

        Set_Info(Gela_Ids.Create_Id(Elem), Info);


   end Set_Info;


   function Get_Info(Id : Gela_Ids.Id) return Element_Info is
   begin
      return Element_Id_List.Search_Node(Info_List, Id);
   end Get_Info;

   function Get_Info(Elem : Asis.Element) return Element_Info is
   begin
      return Get_Info(Gela_Ids.Create_Id(Elem));
   end Get_Info;

   procedure Insert_Function(Fnode : in Function_Nodes; Node_Num : in out Nodenum_T; Elem : in Asis.Element) is
      Node : Node_Item_Link := Search_Node(Node_Num);
      Start_N : Node_Item_Link := Search_Node(Fnode.Start_Node);
      End_N : Node_Item_Link := Search_Node(Fnode.End_Node);
   begin
      if Is_Debug_Mode then
         Put_Line("insert_function");
      end if;
--       N_Io.Put(Node_Num);
--       New_Line;
--       N_Io.Put(Fnode.Start_Node);
--       New_Line;
--       N_Io.Put(Fnode.End_Node);
--       New_Line;
      Node.Number := Start_N.Number;
      Start_N.Number := Node_Num;
```

```
      Node_Num := Node.Number;
      Set_Info(Elem, (Start_N.Number, Node.Number, Node.Number +1));
      Add_Branch(Fnode.End_Node, Node_Num);
   end Insert_Function;

   procedure Process_Nextcall is
      use Nextcall_Stacks;
      Branch : Branch_Item_Link;
      Node_Item : Node_Item_Link;
      Keep_Next : Dug_Channel_Item_Link;
   begin
      while not Isempty(Nextcall_Stack) loop
         Branch := Search_Node(Top(Nextcall_Stack).Previous_Node).Branch;
         while Branch /= null loop
            Node_Item := Search_Node(Branch.Head);
            if Top(Nextcall_Stack).Is_Send then
               Keep_Next := Node_Item.Send;
               Node_Item.Send := new Dug_Channel_Item;
               Node_Item.Send.all
                 := Top(Nextcall_Stack).Channel_Item.all;
               Node_Item.Send.Next := Keep_Next;
            else
               Keep_Next := Node_Item.Receive;
               Node_Item.Receive := new Dug_Channel_Item;
               Node_Item.Receive.all
                 := Top(Nextcall_Stack).Channel_Item.all;
               Node_Item.Receive.Next := Keep_Next;
            end if;
            declare
               use Varnum_Stacks;
               use Var_Item_Stacks;
               Out_Var_Stack : Varnum_Stacks.Stack;
-- := Top(Nextcall_Stack).Arg_Item.Var_Stack;
               Out_Var_Item_Stack : Var_Item_Stacks.Stack;
-- := Top(Nextcall_Stack).Arg_Item.Var_Item_Stack;
               Arg_Stack : Out_Arg_Stacks.Stack := Top(Nextcall_Stack).Arg_Item;
            begin
               while not Isempty(Arg_Stack) loop
                  Out_Var_Stack := Top(Arg_Stack).Var_Stack;
                  Out_Var_Item_Stack := Top(Arg_Stack).Var_Item_Stack;
                  while not Isempty(Out_Var_Stack) loop
                     Assign_Variable(Top(Out_Var_Stack),
                                     Branch.Head);
                     Remove(Out_Var_Stack);
                  end loop;
                  while not Isempty(Out_Var_Item_Stack) loop
                     Assign_Variable(Top(Out_Var_Item_Stack).Number,
                                     Branch.Head,
                                     Top(Out_Var_Item_Stack).Is_Normalized,
                                     Top(Out_Var_Item_Stack).Normalized_Form);
                     Remove(Out_Var_Item_Stack);
                  end loop;
                  Remove(Arg_Stack);
               end loop;
            end;
            Branch := Branch.Next;
         end loop;
         Remove(Nextcall_Stack);
      end loop;
   end Process_Nextcall;


   function Search_Function_Call(Function_Name : V_String)
                                 return Subprogram_Item_Link is
      Iter_Unit : Unit_Item_Link := Current_Unit;
      Iter_Name : V_String := Function_Name;
      Temp_Prefix : V_String;
      Iter_Child : Unit_Item_Link;

      function Prefix return V_String is
         Temp_Str : String := To_S(Iter_Name);
         Prt : Integer := 1;
      begin
         if Temp_Str'Length = 0 then
            return Null_Str;
         end if;
         for I in Temp_Str'Range loop
            if Temp_Str(I) = '.' then
               Iter_Name := To_V(Temp_Str(I+1..Temp_Str'Last));
               return To_V(Temp_Str(Temp_Str'First..I-1));
            end if;
         end loop;
         Iter_Name := Null_Str;
         return To_V(Temp_Str);
      end Prefix;

   begin
      while Iter_Name /= Null_Str loop
         Temp_Prefix := Prefix;
         Iter_Child := Iter_Unit.Children;
         loop
            if Iter_Child.Name = Temp_Prefix then
               Iter_Unit := Iter_Child;
               exit;
            end if;
            Iter_Child := Iter_Child.Next;
            if Iter_Child = null then
               return null;
            end if;
         end loop;
      end loop;
      return Iter_Unit.Subprogram;
   end Search_Function_Call;


   procedure Set_Call_Tree(Caller, Callee : in out Subprogram_Item_Link ;
                           Node : in Nodenum_T) is
      procedure Add_Tree(Target : in out Call_Tree_Link;
                         Subprog : in Subprogram_Item_Link) is
```

```
      Iter : Nodenum_List_Link;
   begin
      if Target = null then
         Target := new Call_Tree;
         Target.Partner := Subprog;
         Target.Caller_Node := new Nodenum_List;
         Target.Caller_Node.Content := Node;
         Target.Caller_Node.Next := null;
      elsif Target.Partner = Subprog then
         Iter := Target.Caller_Node;
         loop
            if Iter.Next = null then
               Iter.Next := new Nodenum_List;
               Iter.Next.Content := Node;
               Iter.Next.Next := null;
               exit;
            elsif Iter.Next.Content = Node then
               exit;
            end if;
            Iter := Iter.Next;
         end loop;
      else
         Add_Tree(Target.Next, Subprog);
      end if;
   end Add_Tree;

begin
   if Caller /= null then
      Add_Tree(Caller.Calling_Tree, Callee);
   end if;
   if Callee /= null then
      Add_Tree(Callee.Called_Tree, Caller);
   end if;
end Set_Call_Tree;


procedure Output_Call_Tree(Unit : in Unit_Item_Link) is
   Iter_Call : Call_Tree_Link;
   Iter_Node : Nodenum_List_Link;
begin
   Put_Line("[unit " & To_S(Unit.Name) &"]");
   if Unit.Subprogram /= null then
      Put_Line("[procedure/function " & To_S(Unit.Name) &"]");
      Iter_Call := Unit.Subprogram.Calling_Tree ;
      while Iter_Call /= null loop
         Put("<call to> " & To_S(Iter_Call.Partner.Name) & " <nodes> ");
         Iter_Node := Iter_Call.Caller_Node;
         while Iter_Node /= null loop
            Put(Nodenum_T'Image(Iter_Node.Content) & " ");
            Iter_Node := Iter_Node.Next;
         end loop;
         New_Line;
         Iter_Call := Iter_Call.Next;
      end loop;
   end if;

   if Unit.Children /= null then
      Output_Call_Tree(Unit.Children);
   end if;
   if Unit.Brother /= null then
      Output_Call_Tree(Unit.Brother);
   end if;
end Output_Call_Tree;


function Find_Visible_Var_Link
  (Var_Name : in V_String; Use_Var : in Boolean)
   return Var_Item_Link
is
   Search_Unit: Unit_Item_Link;
   Variable: Var_Item_Link;
   Last_Unit : Unit_Item_Link := null;
   Trace: Dug_Var_Item_Link;
   New_Var: Dug_Var_Item_Link;
   use Asis;
begin
   Search_Unit:= Current_Unit;
   while Search_Unit /= null loop
      Variable:= Search_Unit.Vars;
      while Variable /= null loop
         if Equal_Insensitive(Variable.Name, Var_Name) then
            if Search_Unit.Decl_Kind = Asis.A_Protected_Body_Declaration
              and then Last_Unit /= null then
               if Use_Var then
                  Trace := Last_Unit.Protected_Use;
               else
                  Trace := Last_Unit.Protected_Def;
               end if;
               while Trace /= null loop
                  exit when Trace.Number = Variable.Number;
                  Trace := Trace.Next;
               end loop;
               if Trace = null then
                  New_Var:= new Dug_Var_Item;
                  New_Var.Number:= Variable.Number;
                  if Use_Var then
                     New_Var.Next:= Last_Unit.Protected_Use;
                     Last_Unit.Protected_Use := New_Var;
                  else
                     New_Var.Next:= Last_Unit.Protected_Def;
                     Last_Unit.Protected_Def := New_Var;
                  end if;
               end if;
            end if;

            return Variable;
         end if;
         Variable:= Variable.Next;
      end loop;
```

```
         Last_Unit := Search_Unit;
         Search_Unit:= Search_Unit.Parent;
      end loop;
      return null;
   end Find_Visible_Var_Link;

--    function Find_Visible_Var_Link
--      (Var_Name : in Asis.Expression; Use_Var : in Boolean)
--       return Var_Item_Link
--    is
--      Target : Asis.Expression := Var_Name;
--      use Asis;

--    begin
--       if Asis.Elements.Expression_Kind(Target) = A_Selected_Component then


--    end Find_Visible_Var_Link;



   function Search_Var_Item
     (Var_Num : in Varnum_T)
      return Var_Item_Link
   is
      Current_Var: Var_Item_Link:= Variable_Table_Tail;
   begin
      while Current_Var /= null loop
         if Current_Var.Number = Var_Num then
            return Current_Var;
         end if;
         Current_Var:= Current_Var.T_Next;
      end loop;
--      error("No such var_num");
      return Current_Var;
   end Search_Var_Item;


   procedure Set_Nondet_Branch
     (Nodenum : in Nodenum_T)
   is
      Node_Link: Node_Item_Link:= Search_Node(Nodenum);
   begin
      Node_Link.Det_Branch:= False;
   end Set_Nondet_Branch;


   procedure Set_Arg_Branch
     (Nodenum : in Nodenum_T)
   is
      Node_Link: Node_Item_Link:= Search_Node(Nodenum);
   begin
      Node_Link.Arg_Branch:= True;
   end Set_Arg_Branch;


   procedure Make_Labeled_Branch
     (Tail  : in Nodenum_T;
      Label : in Label_T)
   is
      Node : Node_Item_Link:= Search_Node(Tail);
      New_Branch : Branch_Item_Link:= new Branch_Item;
   begin
      New_Branch.Head:= 0;
      New_Branch.Label:= Label;
      New_Branch.Next:= Node.Branch;
      Node.Branch:= New_Branch;
   end Make_Labeled_Branch;

   procedure Connect_Labels
     (Head  : in Nodenum_T;
      Label : in Label_T)
   is

      procedure Iter(Unit : in Unit_Item_Link; Is_Myself : in Boolean) is
         Node : Node_Item_Link;
         Branch : Branch_Item_Link;
      begin
         if Unit /= null then
            Node := Unit.Nodes;
            while Node /= null loop
               Branch:= Node.Branch;
               while Branch /= null loop
                  if Branch.Label = Label then
                     Branch.Head:= Head;
                     Branch.Label:= 0;
                  end if;
                  Branch:= Branch.Next;
               end loop;
               Node:= Node.Next;
            end loop;
            Iter(Unit.Children, False);
            if not Is_myself then
               Iter(Unit.Brother, False);
            end if;
         end if;
      end Iter;

   begin
      Iter(Current_Unit, True);
   end Connect_Labels;


   procedure Protected_Last_Node
     (Node_Num : in Nodenum_T) is
      Node_Link: Node_Item_Link:= Search_Node(Node_Num);
      New_Var: Dug_Var_Item_Link;
      Vars : Var_Item_Link := Current_Unit.Vars;
```

```
begin
   while Vars /= null loop
      New_Var := new Dug_Var_Item;
      New_Var.Number := Vars.Number;
      New_Var.Next := Node_Link.Refer;
      Node_Link.Refer := New_Var;
      Vars := Vars.Next;
   end loop;
   Node_Link.Send := new Dug_Channel_Item'(Current_Unit, null, False, Node_Link.Send);
end Protected_Last_Node;
-- DUN
I : Unit_ID := 1;
File_Count : integer := 1;
procedure Output_Dun
  (File_Name : in String)
is
   Declared_Flag : array(0..Num_Of_Variables) of Boolean
     := (others => False);
   -- protected variable


   function Fullname_Of_Unit
     (Unit_Num : in Unit_ID)
      return V_String;


   function Fullname_Of_Unit
     (Unit_Item : in Unit_Item_Link)
      return V_String;


   function Fullname_Of_Var
     (Var_Num : in Varnum_T)
      return V_String
is
      Current_Var: Var_Item_Link:= Variable_Table_Tail;
      Temp_Name: V_String;
   begin
      Current_Var:= Search_Var_Item(Var_Num);
      Temp_Name:= Fullname_Of_Unit(Current_Var.Unit_Link);
      Temp_Name:= Temp_Name & "." & Current_Var.Name;
      return Temp_Name;
   end Fullname_Of_Var;


   function Fullname_Of_Var
     (Var : in Dug_Var_Item_Link)
      return V_String
   is
      Current_Var: Var_Item_Link:= Variable_Table_Tail;
      Temp_Name: V_String;
   begin
      Current_Var:= Search_Var_Item(Var.Number);
      Temp_Name:= Fullname_Of_Unit(Current_Var.Unit_Link);
      if Var.Normalized_Form /= Null_Str then
         Temp_Name:= Temp_Name & "." & Var.Normalized_Form;
      else
         Temp_Name:= Temp_Name & "." & Current_Var.Name;
      end if;
      if Var.Is_Normalized then
         Temp_Name := "Normal," & Temp_Name;
      end if;
      return Temp_Name;
   end Fullname_Of_Var;

   function Fullname_Of_Unit
     (Unit_Num : in Unit_ID)
      return V_String
   is
   begin
      return Fullname_Of_Unit(Search_Unit_Item(Unit_Num));
   end Fullname_Of_Unit;

   function Fullname_Of_Unit
     (Unit_Item : in Unit_Item_Link)
      return V_String
   is
      Temp_Name: V_String;
      Parent: Unit_Item_Link;
   begin
      Temp_Name:= Unit_Item.Name;
      Parent:= Unit_Item.Parent;
      while Parent /= null loop
         Temp_Name:= Parent.Name & "." & Temp_Name;
         Parent:= Parent.Parent;
      end loop;
      return Temp_Name;
   end Fullname_Of_Unit;


   function Fullname_Of_Type
     (Var_Num : in Varnum_T)
      return V_String
   is
      Current_Var: Var_Item_Link:= Variable_Table_Tail;
      Temp_Name: V_String;
   begin
      Current_Var:= Search_Var_Item(Var_Num);
      Temp_Name:= Current_Var.Type_Ident;
      if Current_Var.Is_Access then
         Temp_Name := "ptr," & Temp_Name;
      end if;
      if Current_Var.Is_Protected then
         Temp_Name := Fullname_Of_Unit(Current_Var.Unit_Link)
           & ")," & Temp_Name;
         if Declared_Flag(Var_Num) then
            Temp_Name := "protected(" & Temp_Name;
         else
            Temp_Name := "declared(" & Temp_Name;
```

```
               Declared_Flag(Var_Num) := True;
            end if;
         end if;
         return Temp_Name;
      end Fullname_Of_Type;


      Output_File: constant String:= Integer'Image(File_Count) & File_Name &".dun";

      type Integer_Item;
      type Integer_Link is access Integer_Item;

      type Integer_Item is
         record
            Value: Integer;
            Next: Integer_Link;
         end record;

--    type Call_Cell;
--    type Call_Cell_Link is access Call_Cell;
--    type Call_Cell is
--    record
--       Call : Integer;
--       Actual_In :
--       Actual_Out : Dug_Var_Item_Link;

      type Output_Cell is
         record
            Line: Linenum_T;
            Number: Nodenum_T;
            Is_Body : Boolean := True;
            Connects: Integer_Link;
            Det_Branch: Boolean:= True;
            Arg_Branch: Boolean:= False;
            Forks: Integer_Link;
            Joins: Integer_Link;
--          Calls : Integer_Link;
            Calls : Call_Branch_Item_Link;
            Args : Integer_Link;
--          Param : Varnum_T;
            Top_Args : Integer_Link;
--             Assigns: Integer_Link;
--             Refers: Integer_Link;
            Assigns : Dug_Var_Item_Link;
            Refers : Dug_Var_Item_Link;
            Formal_In : Dug_Var_Item_Link;
            Formal_Out : Dug_Var_Item_Link;
            Send: Dug_Channel_Item_Link;
            Receive: Dug_Channel_Item_Link;
            Is_Pre : Boolean := False;
            Is_Post : Boolean := False;
         end record;

      package Output_Stacks is new Stacks(Output_Cell);
      use Output_Stacks;

      Output_Stack: Output_Stacks.Stack;
      Current_Unit: Unit_Item_Link;
      Current_Node: Node_Item_Link;
      Branches: Branch_Item_Link;
      Forks: Branch_Item_Link;
      Joins: Branch_Item_Link;
      Calls: Call_Branch_Item_Link:= null;
      Args: Branch_Item_Link:= null;
      Top_Args: Branch_Item_Link:= null;
      Assigns: Dug_Var_Item_Link;
      Refers: Dug_Var_Item_Link;
      New_Output: Output_Cell;
      New_Cell: Integer_Link;
--    Value: Integer;
      Fd: File_Type;

      Position : Natural;
      Contents : Dug_Var_Item_Link;
      Is_Last : Boolean;

      Num_Of_Connect, Num_Of_Sconnect, Num_Of_Fork, Num_Of_Join,
        Num_Of_Call : Integer := 0;

   begin
      File_Count := File_Count + 1;
      Create(Fd, Out_File, Output_File(Output_File'range));
      while I in I .. Num_Of_Units loop
         Clear(Output_Stack);
         Current_Unit:= Search_Unit_Item(I);
--          Put_Line("");
         Current_Node:= Current_Unit.Nodes;

         while Current_Node /= null loop
            New_Output:= Output_Cell'(0, 0, True, null, True, False, null,
                                      null,
                                      null, null, null, null, null, null,
                                      null, null, null, False, False);
            New_Output.Line:= Current_Node.Linenum;
            New_Output.Number:= Current_Node.Number;
            New_Output.Is_Body := Current_Node.Is_Body;
            New_Output.Det_Branch:= Current_Node.Det_Branch;
            New_Output.Arg_Branch:= Current_Node.Arg_Branch;
            New_Output.Is_Pre := Current_Node.Is_Pre;
            New_Output.Is_Post := Current_Node.Is_Post;
            Branches:= Current_Node.Branch;

            while Branches /= null loop
               New_Cell:= new Integer_Item;
               New_Cell.Value:= Integer(Branches.Head);
               if Branches.Label /= 0 then
                  Put_Line(Standard_Output,
                           "unconnected label detected:" &
                           Nodenum_T'Image(Current_Node.Number) &
```

```
                        Label_T'Image(Branches.Label));
                  raise PROGRAM_ERROR;
               end if;
               New_Cell.Next:= New_Output.Connects;
               New_Output.Connects:= New_Cell;
               Branches:= Branches.Next;
            end loop;
            Forks:= Current_Node.Forks;
            while Forks /= null loop
               New_Cell:= new Integer_Item;
               New_Cell.Value:= Integer(Forks.Head);
               New_Cell.Next:= New_Output.Forks;
               New_Output.Forks:= New_Cell;
               Forks:= Forks.Next;
            end loop;
            Joins:= Current_Node.Joins;
            while Joins /= null loop
               New_Cell:= new Integer_Item;
               New_Cell.Value:= Integer(Joins.Head);
               New_Cell.Next:= New_Output.Joins;
               New_Output.Joins:= New_Cell;
               Joins:= Joins.Next;
            end loop;
            Calls:= Current_Node.Calls;
            New_Output.Calls := Calls;
--             while Calls /= null loop
--                New_Cell:= new Integer_Item;
--                New_Cell.Value:= Integer(Calls.Head);
--                New_Cell.Next:= New_Output.Calls;
--                New_Output.Calls:= New_Cell;
--                Calls:= Calls.Next;
--             end loop;
            Args:= Current_Node.Args;
            while Args /= null loop
               New_Cell:= new Integer_Item;
               New_Cell.Value:= Integer(Args.Head);
               New_Cell.Next:= New_Output.Args;
               New_Output.Args:= New_Cell;
               Args:= Args.Next;
            end loop;
--          if Current_Unit.Subprogram /= null then
--             New_Output.Param := Current_Unit.Subprogram.Function_Varnum;
--          end if;
            Top_Args:= Current_Node.Top_Args;
            while Top_Args /= null loop
               New_Cell:= new Integer_Item;
               New_Cell.Value:= Integer(Top_Args.Head);
               New_Cell.Next:= New_Output.Top_Args;
               New_Output.Top_Args:= New_Cell;
               Top_Args:= Top_Args.Next;
            end loop;
            Assigns:= Current_Node.Assign;
            New_Output.Assigns := Assigns;
--             while Assigns /= null loop
--                New_Cell:= new Integer_Item;
--                New_Cell.Value:= Integer(Assigns.Number);
--                New_Cell.Next:= New_Output.Assigns;
--                New_Output.Assigns:= New_Cell;
--                Assigns:= Assigns.Next;
--             end loop;
            Refers:= Current_Node.Refer;
            New_Output.Refers := Refers;
--             while Refers /= null loop
--                New_Cell:= new Integer_Item;
--                New_Cell.Value:= Integer(Refers.Number);
--                New_Cell.Next:= New_Output.Refers;
--                New_Output.Refers:= New_Cell;
--                Refers:= Refers.Next;
--             end loop;
            New_Output.Formal_in := Current_Node.Formal_In;
            New_Output.Formal_Out := Current_Node.Formal_Out;
--           if Current_Node.Send /= null then
--             New_Output.Send:=Current_Node.Send;
--                  Fullname_Of_Unit(Current_Node.Send.Unit_Link);
--                if Current_Node.Send.Entry_Link /= null then
--                   New_Output.Send:= New_Output.Send
--                     & "." & Current_Node.Send.Entry_Link.Name;
--                end if;
--                if Current_Node.Send.Ending = True then
--                   New_Output.Send:= New_Output.Send & "!END";
--                end if;
--             end if;
--           if Current_Node.Receive /= No_Channel then
--             New_Output.Receive:=Current_Node.Receive;
--                  Fullname_Of_Unit(Current_Node.Receive.Unit_Link);
--                if Current_Node.Receive.Entry_Link /= null then
--                   New_Output.Receive:= New_Output.Receive
--                     & "." & Current_Node.Receive.Entry_Link.Name;
--                end if;
--                if Current_Node.Receive.Ending = True then
--                   New_Output.Receive:= New_Output.Receive & "!END";
--                end if;
--             end if;
            Current_Node:= Current_Node.Next;
            Push(Output_Stack, New_Output);
         end loop;
         while not Isempty(Output_Stack) loop
            New_Output:= Top(Output_Stack);
            Put(Fd, Nodenum_T'Image(New_Output.Number) & ": ");
            Put(Fd, "<line> ");
            if New_Output.Is_Body then
               Put(Fd, "b");
            else
               Put(Fd, "s");
            end if;
            Put(Fd, Linenum_T'Image(-New_Output.Line) & " ");
            if New_Output.Connects /= null then
               if New_Output.Det_Branch = False then
                  Put(Fd, "<s-");
```

```
            elsif New_Output.Arg_Branch = True then
               Put(Fd, "<a-");
            elsif New_Output.Is_Pre = True then
               Put(Fd, "<Pre-");
            elsif New_Output.Is_Post = True then
               Put(Fd, "<Post-");
            else
               Put(Fd, "<");
            end if;
            Put(Fd, "connect>");
            while New_Output.Connects /= null loop
               Put(Fd, Integer'Image(New_Output.Connects.Value));
               if New_Output.Det_Branch = False then
                  Num_Of_Sconnect := Num_Of_Sconnect + 1;
               elsif New_Output.Arg_Branch = False then
                  Num_Of_Connect := Num_Of_Connect + 1;
               end if;
               New_Output.Connects:= New_Output.Connects.Next;
            end loop;
            Put(Fd, " ");
         end if;
         if New_Output.Assigns /= null then
            Put(Fd, "<def> ");
            while New_Output.Assigns /= null loop
--                Value:= New_Output.Assigns.Value;
--                Put(Fd, To_S(Fullname_Of_Type(Varnum_T(Value))) & "," &
--                 To_S(Fullname_Of_Var(Varnum_T(Value)))
--                    & Integer'Image(-Value) & " ");
               Put(Fd, To_S(Fullname_Of_Type(New_Output.Assigns.Number)) & "," &
                    To_S(Fullname_Of_Var(New_Output.Assigns))
                    & Varnum_T'Image(-New_Output.Assigns.Number) & " ");
               New_Output.Assigns:= New_Output.Assigns.Next;
            end loop;
         end if;
         if New_Output.Refers /= null then
            Put(Fd, "<use> ");
            while New_Output.Refers /= null loop
--                Value:= New_Output.Refers.Value;
--                Put(Fd, To_S(Fullname_Of_Type(Varnum_T(Value))) & "," &
--                 To_S(Fullname_Of_Var(Varnum_T(Value)))
--                    & Integer'Image(-Value) & " ");
--                New_Output.Refers:= New_Output.Refers.Next;
               Put(Fd, To_S(Fullname_Of_Type(New_Output.Refers.Number)) & "," &
                    To_S(Fullname_Of_Var(New_Output.Refers))
                    & Varnum_T'Image(-New_Output.Refers.Number) & " ");
               New_Output.Refers:= New_Output.Refers.Next;
            end loop;
         end if;
         if New_Output.Joins /= null then
            Put(Fd, "<join>");
            while New_Output.Joins /= null loop
               Put(Fd, Integer'Image(New_Output.Joins.Value));
               Num_Of_Join := Num_Of_Join + 1;
               New_Output.Joins:= New_Output.Joins.Next;
            end loop;
            Put(Fd, " ");
         end if;
         if New_Output.Forks /= null then
            Put(Fd, "<fork>");
            while New_Output.Forks /= null loop
               Put(Fd, Integer'Image(New_Output.Forks.Value));
               Num_Of_Fork := Num_Of_Fork + 1;
               New_Output.Forks:= New_Output.Forks.Next;
            end loop;
            Put(Fd, " ");
         end if;
         if New_Output.Send /= null then
            Put(Fd, "<send> ");
            while New_Output.Send /= null loop
               Put(Fd, To_S(Fullname_Of_Unit(New_Output.Send.Unit_Link)));
               if New_Output.Send.Entry_Link /= null then
                  Put(Fd, "." & To_S(New_Output.Send.Entry_Link.Name));
               end if;
               if New_Output.Send.Ending then
                  Put(Fd, "!END");
               end if;
               Put(Fd, " ");
               New_Output.Send := New_Output.Send.Next;
            end loop;
         end if;
         if New_Output.Receive /= null then
            Put(Fd, "<receive> ");
            while New_Output.Receive /= null loop
               Put(Fd, To_S(Fullname_Of_Unit(New_Output.Receive.Unit_Link)));
               if New_Output.Receive.Entry_Link /= null then
                  Put(Fd, "." & To_S(New_Output.Receive.Entry_Link.Name));
               end if;
               if New_Output.Receive.Ending then
                  Put(Fd, "!END");
               end if;
               Put(Fd, " ");
               New_Output.Receive := New_Output.Receive.Next;
            end loop;
         end if;
         if New_Output.Args /= null then
            if Is_Param_Node_Approach then
               Put(Fd, "<param>");
               while New_Output.Args /= null loop
                  Put(Fd, Integer'Image(New_Output.Args.Value));
                  New_Output.Args:= New_Output.Args.Next;
               end loop;
            else
               Put(Fd, "<return> ");
               if Current_Unit.Subprogram /= null and then Current_Unit.Subprogram.Function_Varnum /= 0 then
                  Put(Fd, To_S(Fullname_Of_Type(Current_Unit.Subprogram.Function_Varnum)) & "," &
                       To_S(Fullname_Of_Var(Current_Unit.Subprogram.Function_Varnum))
                       & Varnum_T'Image(-Current_Unit.Subprogram.Function_Varnum) & " ");
               end if;
            end if;
```

```
            Put(Fd, " ");
         end if;
         if New_Output.Top_Args /= null then
            Put(Fd, "<arg>");
            while New_Output.Top_Args /= null loop
               Put(Fd, Integer'Image(New_Output.Top_Args.Value));
               New_Output.Top_Args:= New_Output.Top_Args.Next;
            end loop;
            Put(Fd, " ");
         end if;
         if New_Output.Calls /= null then
            while New_Output.Calls /= null loop
               Put(Fd, "<call>");
               Put(Fd, Nodenum_T'Image(New_Output.Calls.Head));
               Num_Of_Call := Num_Of_Call + 1;
               if New_Output.Calls.Actual_In /= null then
                  Put(Fd, " <a-in> ");
                  Position := 1;
                  loop
                     Search_Actual_Contents(Position, New_Output.Calls.Actual_In, Contents, Is_Last);
                     if Contents = null or else Contents.all /= Not_Found then
                        while Contents /= null loop
                           Put(Fd, To_S(Fullname_Of_Type(Contents.Number)) & "," &
                               To_S(Fullname_Of_Var(Contents))
                               & Varnum_T'Image(-Contents.Number) & " ");
                           Contents := Contents.Next;
                        end loop;
                        if not Is_Last then
                           Put(Fd, " | ");
                        end if;
                     end if;

                     exit when Is_Last;
                     Position := Position + 1;
                  end loop;
               end if;
               if New_Output.Calls.Actual_Out /= null then
                  Put(Fd, " <a-out> ");
                  Position := 1;
                  loop
                     Search_Actual_Contents(Position, New_Output.Calls.Actual_Out, Contents, Is_Last);
                     if Contents = null or else Contents.all /= Not_Found then
                        while Contents /= null loop
                           Put(Fd, To_S(Fullname_Of_Type(Contents.Number)) & "," &
                               To_S(Fullname_Of_Var(Contents))
                               & Varnum_T'Image(-Contents.Number) & " ");
                           Contents := Contents.Next;
                        end loop;
                        if not Is_Last then
                           Put(Fd, " | ");
                        end if;
                     end if;

                     exit when Is_Last;
                     Position := Position + 1;
                  end loop;

               end if;

               New_Output.Calls:= New_Output.Calls.Next;
               Put(Fd, " ");
            end loop;
            Put(Fd, " ");
         end if;
         if New_Output.Formal_In /= null then
            Put(Fd, "<f-in> ");
            while New_Output.Formal_In /= null loop
               Put(Fd, To_S(Fullname_Of_Type(New_Output.Formal_In.Number)) & "," &
                   To_S(Fullname_Of_Var(New_Output.Formal_In))
                   & Varnum_T'Image(-New_Output.Formal_In.Number) & " ");
               New_Output.Formal_In:= New_Output.Formal_In.Next;
            end loop;
         end if;
         if New_Output.Formal_Out /= null then
            Put(Fd, "<f-out> ");
            while New_Output.Formal_Out /= null loop
               Put(Fd, To_S(Fullname_Of_Type(New_Output.Formal_Out.Number)) & "," &
                   To_S(Fullname_Of_Var(New_Output.Formal_Out))
                   & Varnum_T'Image(-New_Output.Formal_Out.Number) & " ");
               New_Output.Formal_Out:= New_Output.Formal_Out.Next;
            end loop;
         end if;

         Remove(Output_Stack);
         New_Line(Fd);
      end loop;
      New_Line(Fd);
      I := I+1;
   end loop;


   Ada.Text_Io.Put_Line("-- name " & Ada.Text_Io.Name(Fd) & Ascii.Ht);


--       Ada.Text_IO.Put_Line("<connect> x " & Integer'IMAGE(Num_Of_Connect));
--       Ada.Text_IO.Put_Line("<s-connect> x " & Integer'IMAGE(Num_Of_Sconnect));
--       Ada.Text_IO.Put_Line("<fork> x " & Integer'IMAGE(Num_Of_Fork));
--       Ada.Text_IO.Put_Line("<join> x " & Integer'IMAGE(Num_Of_Join));
--       Ada.Text_IO.Put_Line("<call> x " & Integer'IMAGE(Num_Of_Call));
   Close(Fd);

   --      if Exec_Shell_Command("sort -n " & output_file
   --                            & " -o " & output_file) /= 0 then
   --          Set_Exit_Status(1);
   --          return;
   --      end if;
exception
   when others=>
     Put_Line("Exception raised in Output_Dun");
```

```
   end Output_Dun;


begin
   null;
end DUN_Handler;
```

A.1.8 Element_Processing Package

```
------------------------------------------------------------------------------
--                                                                          --
--                    ASIS APPLICATION TEMPLATE COMPONENTS                  --
--                                                                          --
--                     E L E M E N T _ P R O C E S S I N G                  --
--                                                                          --
--                                S p e c                                   --
--                                                                          --
--            Copyright (c) 2000, Free Software Foundation, Inc.            --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Application Templates were developed and are now maintained by Ada  --
-- Core Technologies Inc (http://www.gnat.com).                             --
--                                                                          --
------------------------------------------------------------------------------

--  This package contains routines for ASIS Elements processing.

with Asis;

package Element_Processing is

   procedure Process_Construct (The_Element : Asis.Element);
   --  This is the template for the procedure which is supposed to
   --  perform the analysis of its argument Element (The_Element) based on
   --  the recursive traversing of the Element hierarchy rooted by
   --  The_Element. It calls the instantiation of the ASIS Traverse_Element
   --  generic procedure for The_Element.
   --
   --  This procedure should not be called for Nil_Element;
   --
   --  Note, that the instantiation of Traverse_Element and the way how it is
   --  called is no more then a template. It uses a dummy enumeration type
   --  as the actual type for the state of the traversal, and it uses
   --  dummy procedures which do nothing as actual procedures for Pre- and
   --  Post-operations. The Control parameter of the traversal is set
   --  to Continue and it is not changed by actual Pre- or Post-operations.
   --  All this things will definitely require revising when using this
   --  set of templates to build any real application. (At least you will
   --  have to provide real Pre- and/or Post-operation)
   --
   --  See the package body for more details.

end Element_Processing;



------------------------------------------------------------------------------
--                                                                          --
--                    ASIS APPLICATION TEMPLATE COMPONENTS                  --
--                                                                          --
--                     E L E M E N T _ P R O C E S S I N G                  --
--                                                                          --
--                                B o d y                                   --
--                                                                          --
--            Copyright (c) 2000, Free Software Foundation, Inc.            --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Application Templates were developed and are now maintained by Ada  --
-- Core Technologies Inc (http://www.gnat.com).                             --
--                                                                          --
------------------------------------------------------------------------------

with Asis.Iterator;

with Actuals_For_Traversing;

package body Element_Processing is

   -----------------------
   -- Local subprograms --
   -----------------------

   procedure Recursive_Construct_Processing is new
```

```
      Asis.Iterator.Traverse_Element
        (State_Information => Actuals_For_Traversing.Traversal_State,
         Pre_Operation     => Actuals_For_Traversing.Pre_Op,
         Post_Operation    => Actuals_For_Traversing.Post_Op);
   --  This instantiation of Traverse_Element actually performs the recursive
   --  analysis of ASIS Elements. In this set of ASIS application templates
   --  it does nothing, but you may get a number of simple, but useful ASIS
   --  applications by providing some real code for Pre_Op and/or Post_Op,
   --  and keeping the rest of the template code unchanged. For example,
   --  see ASIS tutorials included in the ASIS distribution.

   -----------------------
   -- Process_Construct --
   -----------------------

   procedure Process_Construct (The_Element : Asis.Element) is
      Process_Control : Asis.Traverse_Control := Asis.Continue;

      Process_State   : Actuals_For_Traversing.Traversal_State :=
         Actuals_For_Traversing.Initial_Traversal_State;

   begin

      Recursive_Construct_Processing
        (Element => The_Element,
         Control => Process_Control,
         State   => Process_State);

   end Process_Construct;

end Element_Processing;
```

A.1.9 Gela_Ids Package

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
--  21  package Asis.Ids
------------------------------------------------------------------------------
------------------------------------------------------------------------------

with Ada.Strings.Wide_Unbounded;
with Asis; use Asis;
with Ada.Strings.Wide_bounded;
package gela_ids is

------------------------------------------------------------------------------
------------------------------------------------------------------------------
--  Asis.Ids provides support for permanent unique Element "Identifiers" (Ids).
--  An Id is an efficient way for a tool to reference an ASIS element.  The Id
--  is permanent from session to session provided that the Ada compilation
--  environment is unchanged.
------------------------------------------------------------------------------
------------------------------------------------------------------------------
--  This package encapsulates a set of operations and queries that implement
--  the ASIS Id abstraction.  An Id is a way of identifying a particular
--  Element, from a particular Compilation_Unit, from a particular Context.
--  Ids can be written to files.  Ids can be read from files and converted into
--  an Element value with the use of a suitable open Context.
--
------------------------------------------------------------------------------
------------------------------------------------------------------------------
--  21.1 type Id
------------------------------------------------------------------------------
--  The Ada element Id abstraction (a private type).
--
--  The Id type is a distinct abstract type representing permanent "names"
--  that correspond to specific Element values.
--
--  These names can be written to files, retrieved at a later time, and
--  converted to Element values.
------------------------------------------------------------------------------
--  ASIS Ids are a means of identifying particular Element values obtained from
--  a particular physical compilation unit.  Ids are "relative names".  Each Id
--  value is valid (is usable, makes sense, can be interpreted) only in the
--  context of an appropriate open ASIS Context.
--
--  Id shall be an undiscriminated private type, or, shall be derived from an
--  undiscriminated private type.  It shall be declared as a new type or as a
--  subtype of an existing type.
------------------------------------------------------------------------------

   type Id is private;
   Nil_Id : constant Id;

   function "=" (Left  : in Id;
                 Right : in Id)
                 return Boolean is abstract;


------------------------------------------------------------------------------
--  21.2        function Hash
------------------------------------------------------------------------------

   function Hash (The_Id : in Id) return Asis.ASIS_Integer;

------------------------------------------------------------------------------
--  21.3        function "<"
------------------------------------------------------------------------------

   function "<" (Left  : in Id;
                 Right : in Id) return Boolean;

------------------------------------------------------------------------------
--  21.4        function ">"
------------------------------------------------------------------------------
```

```
   function ">" (Left  : in Id;
                 Right : in Id) return Boolean;


------------------------------------------------------------------------------
--  21.5        function Is_Nil
------------------------------------------------------------------------------

   function Is_Nil (Right : in Id) return Boolean;

------------------------------------------------------------------------------
--  Right   - Specifies the Id to check
--
--  Returns True if the Id is the Nil_Id.
--
------------------------------------------------------------------------------
--  21.6        function Is_Equal
------------------------------------------------------------------------------

   function Is_Equal (Left  : in Id;
                      Right : in Id) return Boolean;

------------------------------------------------------------------------------
--  Left    - Specifies the left Id to compare
--  Right   - Specifies the right Id to compare
--
--  Returns True if Left and Right represent the same physical Id, from the
--  same physical compilation unit.  The two Ids convert
--  to Is_Identical Elements when converted with the same open ASIS Context.
--
------------------------------------------------------------------------------
--  21.7        function Create_Id
------------------------------------------------------------------------------

   function Create_Id (Element : in Asis.Element) return Id;

------------------------------------------------------------------------------
--  Element - Specifies any Element value whose Id is desired
--
--  Returns a unique Id value corresponding to this Element, from the
--  corresponding Enclosing_Compilation_Unit and the corresponding
--  Enclosing_Context.  The Id value will not be equal ("=") to the Id value
--  for any other Element value unless the two Elements are Is_Identical.
--
--  Nil_Id is returned for a Nil_Element.
--
--  All Element_Kinds are appropriate.
--
------------------------------------------------------------------------------
--  21.8        function Create_Element
------------------------------------------------------------------------------

   function Create_Element (The_Id      : in Id;
                            The_Context : in Asis.Context)
                           return Asis.Element;

------------------------------------------------------------------------------
--  The_Id      - Specifies the Id to be converted to an Element
--  The_Context - Specifies the Context containing the Element with this Id
--
--  Returns the Element value corresponding to The_Id.  The_Id shall
--  correspond to an Element available from a Compilation_Unit contained by
--  (referencible through) The_Context.
--
--  Raises ASIS_Inappropriate_Element if the Element value is not available
--  though The_Context.  The Status is Value_Error and the Diagnosis
--  string will attempt to indicate the reason for the failure.  (e.g., "Unit
--  is inconsistent", "No such unit", "Element is inconsistent (Unit
--  inconsistent)", etc.)
--

------------------------------------------------------------------------------
--  21.9        function Debug_Image
------------------------------------------------------------------------------

   function Debug_Image (The_Id : in Id) return Wide_String;

------------------------------------------------------------------------------
--  The_Id  - Specifies an Id to convert
--
--  Returns a string value containing implementation-defined debug
--  information associated with the Id.
--
--  The return value uses Asis.Text.Delimiter_Image to separate the lines
--  of multi-line results.  The return value does not end with
--  Asis.Text.Delimiter_Image.
--
--  These values are intended for two purposes.  They are suitable for
--  inclusion in problem reports sent to the ASIS implementor.  They can
--  be presumed to contain information useful when debugging the
--  implementation itself. They are also suitable for use by the ASIS
--  application when printing simple application debugging messages during
--  application development.  They are intended to be, to some worthwhile
--  degree, intelligible to the user.
--
------------------------------------------------------------------------------

private
   package W renames Ada.Strings.Wide_Unbounded;

   type Id is record
      Hash : Asis.ASIS_Integer;
      Unit : W.Unbounded_Wide_String;
   end record;

   Nil_Id : constant Id := (0, W.Null_Unbounded_Wide_String);

end gela_ids;
```

```
------------------------------------------------------------------------------
--  Copyright (c) 2006, Maxim Reznik
--  All rights reserved.
--
--  Redistribution and use in source and binary forms, with or without
--  modification, are permitted provided that the following conditions are met:
--
--     * Redistributions of source code must retain the above copyright notice,
--     * this list of conditions and the following disclaimer.
--     * Redistributions in binary form must reproduce the above copyright
--     * notice, this list of conditions and the following disclaimer in the
--     * documentation and/or other materials provided with the distribution.
--
--  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
--  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
--  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
--  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
--  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
--  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
--  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
--  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
--  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
--  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
--  POSSIBILITY OF SUCH DAMAGE.
------------------------------------------------------------------------------


------------------------------------------------------------------------------
--                            G E L A   A S I S                             --
--       ASIS implementation for Gela project, a portable Ada compiler      --
--                     http://www.ten15.org/wiki/Ada                        --
--                     - - - - - - - - - - - - - - -                        --
--            Read copyright and license at the end of this file            --
------------------------------------------------------------------------------
--  $TenDRA$
--  Purpose:
--  Implement permanent element Id on unmodified sources.
--  For Id we use hash of an element concatenated with unit unique name.

with Asis.Errors;
with Asis.Elements;
with Asis.Iterator;
with Asis.Exceptions;
with Asis.Implementation;
with Asis.Compilation_Units;

package body gela_ids is
   use type W.Unbounded_Wide_String;

   type State_Information is record
      Result : Asis.Element;
      Hash   : Asis.ASIS_Integer;
   end record;

   procedure Pre_Operation
     (Element : in     Asis.Element;
      Control : in out Traverse_Control;
      State   : in out State_Information);

   procedure Post_Operation
     (Element : in     Asis.Element;
      Control : in out Traverse_Control;
      State   : in out State_Information);

   ---------
   -- "<" --
   ---------

   function "<"
     (Left  : in Id;
      Right : in Id)
      return Boolean
   is
   begin
      return Left.Hash < Right.Hash or else
        (Left.Hash = Right.Hash and then Left.Unit < Right.Unit);
   end "<";

   ---------
   -- ">" --
   ---------

   function ">"
     (Left  : in Id;
      Right : in Id)
      return Boolean
   is
   begin
      return Left.Hash > Right.Hash or else
        (Left.Hash = Right.Hash and then Left.Unit > Right.Unit);
   end ">";

   -------------------
   -- Pre_Operation --
   -------------------

   procedure Pre_Operation
     (Element : in     Asis.Element;
      Control : in out Traverse_Control;
      State   : in out State_Information)
   is
   begin
      if Asis.Elements.Hash (Element) = State.Hash then
```

```
      State.Result := Element;
      Control := Terminate_Immediately;
   end if;
end Pre_Operation;

--------------------
-- Post_Operation --
--------------------

procedure Post_Operation
  (Element : in     Asis.Element;
   Control : in out Traverse_Control;
   State   : in out State_Information) is
begin
   null;
end Post_Operation;

--------------------
-- Create_Element --
--------------------

function Create_Element
  (The_Id      : in Id;
   The_Context : in Asis.Context)
   return Asis.Element
is
   use Asis.Errors;
   use Asis.Compilation_Units;
   use Asis.Elements;

   procedure Raise_Error (Text : Wide_String);

   procedure Search is
     new Asis.Iterator.Traverse_Element (State_Information);

   procedure Raise_Error (Text : Wide_String) is
   begin
      Asis.Implementation.Set_Status
        (Value_Error, Text);
      raise Asis.Exceptions.ASIS_Inappropriate_Element;
   end Raise_Error;

   Unit    : Asis.Compilation_Unit;
   State   : State_Information;
   Control : Traverse_Control := Continue;
   List    : constant Asis.Compilation_Unit_List :=
     Asis.Compilation_Units.Compilation_Units (The_Context);
begin
   for I in List'Range loop
      if Unique_Name (List (I)) = The_Id.Unit then
         Unit := List (I);
         exit;
      end if;
   end loop;

   if Is_Nil (Unit) then
      Raise_Error ("No such unit");
   else
      State.Hash := The_Id.Hash;
   end if;

   declare
      Clause : constant Asis.Element_List :=
        Context_Clause_Elements (Unit, True);
   begin
      for I in Clause'Range loop
         Search (Clause (I), Control, State);

         if not Is_Not_Null_Return (State.Result) then
            return State.Result;
         end if;
      end loop;
   end;

   Search (Unit_Declaration (Unit), Control, State);

   if not Is_Not_Null_Return (State.Result) then
      return State.Result;
   else
      Raise_Error ("No element for this id");
      return Nil_Element;
   end if;
end Create_Element;

---------------
-- Create_Id --
---------------

function Create_Id (Element : in Asis.Element) return Id is
   use Asis.Elements;
   use Asis.Compilation_Units;

   Result   : Asis.ASIS_Integer;
   The_Unit : Asis.Compilation_Unit :=
     Enclosing_Compilation_Unit (Element);
begin
   if Is_Not_Null_Return (Element) then
      return Nil_Id;
   end if;

   The_Unit := Enclosing_Compilation_Unit (Element);
   Result := Hash (Element);

   return (Result, W.To_Unbounded_Wide_String (Unique_Name (The_Unit)));
end Create_Id;

-----------------
-- Debug_Image --
-----------------
```

```
   function Debug_Image (The_Id : in Id) return Wide_String is
   begin
      return ASIS_Integer'Wide_Image (The_Id.Hash) & "/"
        & W.To_Wide_String (The_Id.Unit);
   end Debug_Image;

   ----------
   -- Hash --
   ----------

   function Hash (The_Id : in Id) return Asis.ASIS_Integer is
   begin
      return The_Id.Hash;
   end Hash;

   --------------
   -- Is_Equal --
   --------------

   function Is_Equal
     (Left  : in Id;
      Right : in Id)
      return Boolean
   is
   begin
      return Left.Hash = Right.Hash and then Left.Unit = Right.Unit;
   end Is_Equal;

   ------------
   -- Is_Nil --
   ------------

   function Is_Nil (Right : in Id) return Boolean is
   begin
      return Right.Hash = 0 and then Right.Unit = "";
   end Is_Nil;

end gela_ids;



------------------------------------------------------------------------------
--  Copyright (c) 2006, Maxim Reznik
--  All rights reserved.
--
--  Redistribution and use in source and binary forms, with or without
--  modification, are permitted provided that the following conditions are met:
--
--     * Redistributions of source code must retain the above copyright notice,
--     * this list of conditions and the following disclaimer.
--     * Redistributions in binary form must reproduce the above copyright
--     * notice, this list of conditions and the following disclaimer in the
--     * documentation and/or other materials provided with the distribution.
--
--  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
--  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
--  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
--  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
--  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
--  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
--  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
--  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
--  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
--  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
--  POSSIBILITY OF SUCH DAMAGE.
------------------------------------------------------------------------------
```

A.1.10 Id_List Package

```
with Asis, Gela_Ids;
use Asis, Gela_Ids;
-- with Types; use Types;

generic
type ITEM is private;
Null_ITEM : ITEM;

package ID_List is
   type NODE;
   type A_NODE is access NODE;

   type NODE is
      record
 Index : Gela_Ids.Id := Gela_Ids.Nil_Id;
 Object : ITEM;
 Next : A_NODE := null;
      end record;

   Null_Node : NODE := (Gela_Ids.Nil_Id , Null_ITEM, null);

   function SEARCH_NODE(TOP : A_NODE; CHILD : Gela_Ids.Id) return ITEM;
   procedure SET_NODE(TOP : in out A_NODE ; CHILD : in Gela_Ids.Id ;
    PARENT : in ITEM);
   function GET_NEXT_NODE(TOP : NODE) return NODE;

   procedure DELETE_NODE(TOP : in out A_NODE; CHILD : in Gela_Ids.Id);
   function IsNULL(TOP : A_NODE) return Boolean;
   function IsNULLItem(TOP : ITEM) return Boolean;
end ID_List;



with Ada.Text_Io; use Ada.Text_Io;
with Asis, Gela_Ids; use Asis, Gela_Ids;
```

```
package body Id_List is

--   procedure FREE_NODE is new UNCHECKED_DEALLOCATION(NODE, A_NODE);

   function SEARCH_NODE(TOP : A_NODE; CHILD : Gela_Ids.Id) return ITEM is
      function SEARCH_ITER(CURRENT_NODE : A_NODE) return ITEM is
      begin
--  Put("o");
--  New_Line;
--  New_Line;
--  Put_Line(Image(CHILD));
--  New_Line;
  if CURRENT_NODE = null then
--     Put_Line("null!");
     return Null_ITEM;
  end if;
--  if CURRENT_NODE.Index = CHILD then
  if Gela_Ids.Is_Equal(CURRENT_NODE.Index, CHILD) then
--     Put_Line("GET!");
--     New_Line;
     return CURRENT_NODE.Object;
  end if;
  return SEARCH_ITER(CURRENT_NODE.Next);
      end SEARCH_ITER;

    begin
      return SEARCH_ITER(TOP);
    end SEARCH_NODE;


   function GET_NEXT_NODE(TOP : NODE) return NODE is
   begin
      if TOP.Next = null then
 return Null_NODE;
      end if;
      return TOP.Next.all;
   end GET_NEXT_NODE;

   procedure SET_NODE(TOP : in out A_NODE ; CHILD : in Gela_Ids.Id ; PARENT : in ITEM) is
      procedure SET_ITER(CURRENT_NODE : in out A_NODE) is
      begin
  if CURRENT_NODE = null then
     CURRENT_NODE := new NODE;
     CURRENT_NODE.Index := CHILD;
     CURRENT_NODE.Object := PARENT;
     CURRENT_NODE.Next := null;
--  elsif CURRENT_NODE.Index = CHILD then
  elsif Gela_Ids.Is_Equal(CURRENT_NODE.Index, CHILD) then
     CURRENT_NODE.Object := PARENT;
  else
           SET_ITER(CURRENT_NODE.Next);
  end if;
      end SET_ITER;
    begin
      SET_ITER(TOP);
    end SET_NODE;

   procedure DELETE_NODE(TOP : in out A_NODE ; CHILD : in Gela_Ids.Id) is
      procedure DELETE_ITER(CURRENT_NODE : in out A_NODE) is
      begin
-- if CURRENT_NODE.Index = CHILD then
 if Gela_Ids.Is_Equal(CURRENT_NODE.Index, CHILD) then
--    FREE_NODE(CURRENT_NODE);
    CURRENT_NODE := CURRENT_NODE.Next;
 elsif CURRENT_NODE /= null then
    DELETE_ITER(CURRENT_NODE.Next);
 end if;
      end DELETE_ITER;
   begin
      DELETE_ITER(TOP);
   end DELETE_NODE;

   function IsNULL(TOP : A_NODE) return Boolean is
   begin
     return (TOP = null);
   end IsNull;

   function IsNULLItem(TOP : ITEM) return Boolean is
   begin
      return (TOP = Null_ITEM);
   end IsNULLItem;
end Id_List;
```

A.1.11 Metrics_Utilities Package

```
------------------------------------------------------------------------------
--                                                                          --
--                       ASIS TUTORIAL COMPONENTS                           --
--                                                                          --
--                    M E T R I C S _ U T I L I T I E S                     --
--                                                                          --
--                                S p e c                                   --
--                                                                          --
--           Copyright (c) 2000, Free Software Foundation, Inc.             --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
```

```
--                                                                          --
-- ASIS Tutorial was developed and are now maintained by Ada Core           --
-- Technologies Inc (http://www.gnat.com).                                  --
--                                                                          --
------------------------------------------------------------------------------

--  This package defines the data structures and the utility programs needed
--  to implement an ASIS Metrics tool in the framework of the ASIS tutorial
--  provided by ASIS-for-GNAT (Task 2).

package Metrics_Utilities is

   --  Metric counters:
   Total_Statements   : Natural := 0;
   Total_Declarations : Natural := 0;

   --  Metric counters added for Task 2:
   Simple_Statements   : Natural := 0;
   Compound_Statements : Natural := 0;
   Defining_Names      : Natural := 0;

   procedure Reset_Metric_Counters;
   --  Sets all the metric counters to zero.

   procedure Print_Metric_Counterts;
   --  Outputs the values of the metric counters.

end Metrics_Utilities;



------------------------------------------------------------------------------
--                                                                          --
--                       ASIS TUTORIAL COMPONENTS                           --
--                                                                          --
--                   M E T R I C S _ U T I L I T I E S                      --
--                                                                          --
--                                B o d y                                   --
--                                                                          --
--            Copyright (c) 2000, Free Software Foundation, Inc.            --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Tutorial was developed and are now maintained by Ada Core           --
-- Technologies Inc (http://www.gnat.com).                                  --
--                                                                          --
------------------------------------------------------------------------------

with Ada.Wide_Text_IO;

package body Metrics_Utilities is

   ----------------------------
   -- Print_Metric_Counterts --
   ----------------------------

   procedure Print_Metric_Counterts is
      package Natural_IO is new Ada.Wide_Text_IO.Integer_IO (Natural);
   begin
      Ada.Wide_Text_IO.Put ("Total statements    :");
      Natural_IO.Put       (Total_Statements);
      Ada.Wide_Text_IO.New_Line;

      --   Added for Task 2:
      Ada.Wide_Text_IO.Put ("Simple statements   :");
      Natural_IO.Put       (Simple_Statements);
      Ada.Wide_Text_IO.New_Line;

      --   Added for Task 2:
      Ada.Wide_Text_IO.Put ("Compound statements :");
      Natural_IO.Put       (Compound_Statements);
      Ada.Wide_Text_IO.New_Line;


      Ada.Wide_Text_IO.Put ("Total declarations  :");
      Natural_IO.Put       (Total_Declarations);
      Ada.Wide_Text_IO.New_Line;

      --   Added for Task 2:
      Ada.Wide_Text_IO.Put ("Defining names      :");
      Natural_IO.Put       (Defining_Names);
      Ada.Wide_Text_IO.New_Line;
   end Print_Metric_Counterts;

   ---------------------------
   -- Reset_Metric_Counters --
   ---------------------------

   procedure Reset_Metric_Counters is
   begin
      Total_Statements    := 0;
      Total_Declarations  := 0;

      --   Added for Task 2:
      Simple_Statements   := 0;
      Compound_Statements := 0;
      Defining_Names      := 0;

   end Reset_Metric_Counters;
```

```
end Metrics_Utilities;
```

A.1.12 Stacks Package

```
-- $Id: stacks.ads,v 1.1.1.1 2000/05/26 01:11:18 yoshimi Exp $
--
-- Generic Stack Handling Package Spec.
--
--  by Yoshiaki kasahara, 1992, 1993.

generic
    type item is private;
package Stacks is

    type cell;
    type stack is access cell;

    type cell is
    record
content: item;
next: stack:= null;
    end record;

    procedure push(s: in out stack; x: in item);
    function top(s: in stack) return item;
    procedure remove(s: in out stack);
    procedure pop(s: in out stack; x: out item);
    procedure clear(s: in out stack);
    function isempty(s: in stack) return boolean;
    procedure concat(s1: in out stack; s2: in out stack; s3: out stack);
end Stacks;



-- $Id: stacks.adb,v 1.1.1.1 2000/05/26 01:11:18 yoshimi Exp $
--
-- Generic Stack Handling Package Body
--
--  by Yoshiaki Kasahara, 1992, 1993.

package body Stacks is
    procedure push(s: in out stack; x: in item) is
    begin
s:= new cell'(x, s);
    end push;

    function top(s: in stack) return item is
    begin
return s.content;
    end top;

    procedure remove(s: in out stack) is
    begin
if s = null then
    return;
end if;
s:= s.next;
    end remove;

    procedure pop(s: in out stack; x: out item) is
    begin
x:= s.content;
s:= s.next;
    end pop;

    procedure clear(s: in out stack) is
    begin
s:= null;
    end clear;

    function isempty(s: in stack) return boolean is
    begin
if s = null then
    return true;
else
    return false;
end if;
    end isempty;

    procedure concat(s1: in out stack; s2: in out stack; s3: out stack) is
top: stack:= s2;
    begin
if s1 /= s2 then
    if top = null then
s2:= s1;
    else
while top.next /= null loop
    top:= top.next;
    if top = s1 then
raise constraint_error;
    end if;
end loop;
top.next:= s1;
    end if;
end if;
s3:= s2;
s1:= null;
s2:= null;
    end concat;
end Stacks;
```

A.1.13 Unit_Processing Package

```
--                                                                          --
--                  ASIS APPLICATION TEMPLATE COMPONENTS                    --
--                                                                          --
--                     U N I T _ P R O C E S S I N G                        --
--                                                                          --
--                                 S p e c                                  --
--                                                                          --
--           Copyright (c) 2000, Free Software Foundation, Inc.             --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Application Templates were developed and are now maintained by Ada  --
-- Core Technologies Inc (http://www.gnat.com).                             --
--                                                                          --
------------------------------------------------------------------------------

--  This package contains routines for ASIS Compilation Units processing.

with Asis;

package Unit_Processing is

   procedure Process_Unit (The_Unit : Asis.Compilation_Unit);
   --  This procedure decomposes its argument unit and calls the element
   --  processing routine for all the top-level components of the unit
   --  element hierarchy. This element processing routine is the instance
   --  of Traverse_Element which performs the recursive traversing of
   --  the unit structural components. In the given set of the ASIS
   --  implementation templates, the element processing routine is used
   --  "empty" actual Pre- and Post-Operations to instantiate
   --  Traverse_Element, they should be replaced by real subprograms
   --  when using this set of templates to build a real ASIS application
   --  from it.
   --
   --  It the argument Unit is Nil or nonexistent unit, this procedure does
   --  nothing.

end Unit_Processing;
```

```
------------------------------------------------------------------------------
--                                                                          --
--                          ASIS TUTORIAL COMPONENTS                        --
--                                                                          --
--                       U N I T _ P R O C E S S I N G                      --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--           Copyright (c) 2000, Free Software Foundation, Inc.             --
--                                                                          --
-- ASIS  Application  Templates are  free software; you can redistribute it --
-- and/or  modify it under  terms  of the  GNU  General  Public  License as --
-- published by the Free Software Foundation; either version 2, or (at your --
-- option) any later version. ASIS Application Templates are distributed in --
-- the hope that they will be useful, but  WITHOUT  ANY  WARRANTY; without  --
-- even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR --
-- PURPOSE. See the GNU General Public License for more details. You should --
-- have  received a copy of the GNU General Public License distributed with --
-- distributed  with  GNAT;  see  file  COPYING. If not, write to the Free  --
-- Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, --
-- USA.                                                                     --
--                                                                          --
-- ASIS Tutorial was developed and are now maintained by Ada Core           --
-- Technologies Inc (http://www.gnat.com).                                  --
--                                                                          --
------------------------------------------------------------------------------

--  This is the modification of the file with the same name from the ASIS
--  Application Templates to be used in the solution of the metrics tool
--  (for Task 1 and Task 2 the same version of this file is used)

with Asis.Elements;

with Element_Processing;

with Metrics_Utilities;
--  Added for the solution of the metrics tool

package body Unit_Processing is

   ------------------
   -- Process_Unit --
   ------------------

   procedure Process_Unit (The_Unit : Asis.Compilation_Unit) is
      Cont_Clause_Elements : constant Asis.Element_List :=
         Asis.Elements.Context_Clause_Elements (Compilation_Unit => The_Unit,
                                                Include_Pragmas  => True);
      --  This is the list of the context clauses, including pragmas, if any.
      --  If you do not want to process pragmas, set Include_Pragmas OFF when
      --  calling Asis.Elements.Context_Clause_Elements

      Unit_Decl : Asis.Element := Asis.Elements.Unit_Declaration (The_Unit);
      --  The top-level structural element of the library item or subunit
      --  contained in The_Unit.

   begin
```

```
      Metrics_Utilities.Reset_Metric_Counters;
      --  Added for the solution of the metrics tool

      for J in Cont_Clause_Elements'Range loop
         Element_Processing.Process_Construct (Cont_Clause_Elements (J));
      end loop;
      --  Many applications are not interested in processing the context
      --  clause of the compilation units. If this is the case for your
      --  application, simply remove this loop statement.

      Element_Processing.Process_Construct (Unit_Decl);

      --  This procedure does not contain any exception handler because it
      --  supposes that Element_Processing.Process_Construct should handle
      --  all the exceptions which can be raised when processing the element
      --  hierarchy

      Metrics_Utilities.Print_Metric_Counterts;
      --  Added for the solution of the metrics tool

   end Process_Unit;

end Unit_Processing;
```

A.1.14 V_Strings Package

```
-- $Id: v_strings.ads,v 1.1.1.1 2000/05/26 01:11:18 yoshimi Exp $
--
-- Variable-length strings handler package specification.
--
--  by Yoshiaki Kasahara, 1992, 1993.

package V_Strings is

type v_string is private;

null_str: constant v_string;

function to_v(s: in string) return v_string;
function to_s(s: in v_string) return string;
function to_v(s: in Wide_string) return v_string;
function to_s(s: in v_string) return Wide_string;

function equal(s1, s2: v_string) return boolean;
function Equal_Insensitive(s1, s2: v_string) return Boolean;
function "&"(s1: v_string;  s2: v_string ) return v_string;
function "&"(s1: v_string;  s2: string   ) return v_string;
function "&"(s1: string;    s2: v_string ) return v_string;
function "&"(s1: v_string;  s2: character) return v_string;
function "&"(s1: character; s2: v_string ) return v_string;

private

MAX_LENGTH: constant natural := 1024;

type v_string is
record
str: string(1 .. MAX_LENGTH)
 := (1 .. MAX_LENGTH => ascii.nul);
len: natural range 0 .. MAX_LENGTH:= 0;
end record;

null_str: constant v_string:=( (1 .. MAX_LENGTH => ascii.nul), 0);

end V_Strings;
```

```
-- $Id: v_strings.adb,v 1.1.1.1 2000/05/26 01:11:18 yoshimi Exp $
--
-- Variable-length strings handler package body.
--
--  by Yoshiaki Kasahara, 1992, 1993.

with Ada.Characters.Handling;
use Ada.Characters.Handling;
package body V_Strings is

function to_v(s: in string) return v_string is
buffer: v_string;
begin
buffer.len := s'Last - s'First + 1;
buffer.str(1..buffer.len) := s;
return buffer;
end to_v;

function to_v(s: in Wide_string) return v_string is
buffer: v_string;
begin
buffer.len := s'Last - s'First + 1;
buffer.str(1..buffer.len) := To_String(S);
return buffer;
end to_v;

function to_s(s: in v_string) return string is
begin
return s.str(1..s.len);
end to_s;

function to_s(s: in v_string) return Wide_string is
begin
return To_Wide_String(s.str(1..s.len));
end to_s;
```

```
function equal(s1, s2: v_string) return boolean is
begin
return ((s1.str = s2.str) and (s1.len = s2.len));
end equal;

function Equal_Insensitive(s1, s2: v_string) return boolean is
begin
return ((To_Lower(s1.Str) = To_Lower(s2.Str)) and (s1.len = s2.len));
end Equal_Insensitive;

function "&"(s1: v_string; s2: v_string) return v_string is
buffer: v_string;
begin
buffer.len:= s1.len + s2.len;
buffer.str(1..buffer.len)
:= s1.str(1..s1.len)&s2.str(1..s2.len);
return buffer;
end "&";

function "&"(s1: v_string; s2: string) return v_string is
buffer: v_string;
begin
buffer.len:= s1.len + s2'length;
buffer.str(1..buffer.len):= s1.str(1..s1.len)&s2;
return buffer;
end "&";

function "&"(s1: string; s2: v_string) return v_string is
buffer: v_string;
begin
buffer.len:= s1'length + s2.len;
buffer.str(1..buffer.len):= s1&s2.str(1..s2.len);
return buffer;
end "&";

function "&"(s1: v_string; s2: character) return v_string is
buffer: v_string;
begin
buffer.len:= s1.len + 1;
buffer.str(1..buffer.len):= s1.str(1..s1.len)&s2;
return buffer;
end "&";

function "&"(s1: character; s2: v_string) return v_string is
buffer: v_string;
begin
buffer.len:= 1 + s2.len;
buffer.str(1..buffer.len):= s1&s2.str(1..s2.len);
return buffer;
end "&";

end V_Strings;
```

Appendix B

A System Dependence Net Generator for Ada 2012 Programs

B.1 System Dependence Net Generator

B.1.1 DUN2SDN Package

```
#!/usr/local/bin/perl

# dun2SDN v1.5 -- wangbo@aise.ics.saitama-u.ac.jp  Exp(2016.2.17)
# dug2pdg [-c -a -s -d] foo.dug
#
# -c @)8fOMB8$r5a$a$k
# -d %G!<%?OMB8$r5a$a$k
# -a @)8fOMB8!"%G!<%?OMB8$NN>J}$r5a$a$k!#
#
# $?$@$7!"%3%~%s%IL>$NKvHx$,!"-c -d $N$H$-$O!"$=$l$>$l%3%s%H%m!<%kOMB8!"
# %G!<%?OMB8OJ3O$O5a$a$J$$!#


# -s F14|OMB8$r5a$a$k!#
# $?$@$7!"%3%~%s%IL>$NKvHx$, -s $N$H$-$O!"F14|OMB8$7$+5a$a$J$$!#
#
# -m DL?.OMB8$r5a$a$k!#
# $?$@$7!"%3%~%s%IL>$NKvHx$, -m $N$H$-$O!"DL?.OMB8$7$+5a$a$J$$!#
#
# -sl A*BrOMB8$r5a$a$k!#
# $?$@$7!"%3%~%s%IL>$NKvHx$, -sl $N$H$-$O!"A*BrOMB8$7$+5a$a$J$$!#
#
# -p 8F=POMB8;^$H!"JQ?tF~=PNO;^$r5a$a$k!#
# $?$@$7!"%3%~%s%IL>$NKvHx$, -p $N$H$-$O!"OMB8$7$+5a$a$J$$!#

require "./data.pl";
require "./control.pl";
require "./sync.pl";
require "./comm.pl";
require "./select.pl";
require "./call.pl";
require "./expandnode.pl";
require "./prepost.pl"; --contract-based programming with Ada 2012
require "./slicing.pl";

$Filename = undef;

$ALL = 1;
while ($_ = $ARGV[$[]) {
    s/^-// || last;

    $CONTROL = 1 if ($_ eq "c");
    $DATA = 1    if ($_ eq "d");
    $SYNC = 1    if ($_ eq "s");
    $COMM = 1    if ($_ eq "m");
    $SELECT  = 1 if ($_ eq "sl");
    $CALL = 1    if ($_ eq "p");
    $ALL = 1     if ($_ eq "a");

    shift @ARGV;
}

($CONTROL, $DATA, $SYNC, $COMM, $SELECT, $CALL, $ALL) =
    (1, 0, 0, 0 ,0 ,0 ,0) if ($0 =~ /-c$/);
($CONTROL, $DATA, $SYNC, $COMM, $SELECT, $CALL, $ALL) =
    (0, 1, 0, 0 ,0 ,0 ,0) if ($0 =~ /-d$/);
($CONTROL, $DATA, $SYNC, $COMM, $SELECT, $CALL, $ALL) =
    (0, 0, 1, 0 ,0 ,0 ,0) if ($0 =~ /-s$/);
($CONTROL, $DATA, $SYNC, $COMM, $SELECT, $CALL, $ALL) =
    (0, 0, 0, 1 ,0 ,0 ,0) if ($0 =~ /-m$/);
($CONTROL, $DATA, $SYNC, $COMM, $SELECT, $CALL, $ALL) =
    (0, 0, 0, 0 ,1 ,0 ,0) if ($0 =~ /-sl$/);
($CONTROL, $DATA, $SYNC, $COMM, $SELECT, $CALL, $ALL) =
    (0, 0, 0, 0 ,0 ,1 ,0) if ($0 =~ /-p$/);

if ($CONTROL) {
    mkNodeList();
```

```
    mkControlDependence();
    mkSelectDependence();
    &expandDUN();  #
    mkArgnodeControldependence();
    printControlDependence();

} elsif ($DATA){
    mkNodeList();
    &expandDUN();  #
    mkDataDependence();
    printDataDependence();

} elsif ($SELECT){
    mkNodeList();
    &expandDUN();  #
    mkArgnodeControldependence();
    mkControlDependence();
    mkSelectDependence();
    printSelectDependence();

} elsif ($SYNC){
    mkNodeList();
    mkControlDependence();
    mkSyncDependence();
    &expandDUN();  #
    printSyncDependence();

} elsif ($COMM){
    mkNodeList();
    &expandDUN();  #
    mkDataDependence();
    mkCommDependence();
    printCommDependence();

} elsif ($CALL){
    mkNodeList();
    mkDataDependence();
    mkCallArc();
    mkParamArc();
    mkSummuryArc();
    printCallandParamArc();

} elsif ($ALL) {
#print "begin :",‘date‘;
    mkNodeList();
#print "Control begin :",‘date‘;
    mkControlDependence();
#print "Sync begin :",‘date‘;
    mkSyncDependence();
#print "Select begin :",‘date‘;
    mkSelectDependence();
#print "begin :",‘date‘;
    &expandDUN();  #
#print "Control begin :",‘date‘;
    mkArgnodeControldependence();
#print "Data begin :",‘date‘;
    mkDataDependence();
#print "Comm begin :",‘date‘;
    mkCommDependence();
#print "Call begin :",‘date‘;
    mkCallArc();
    mkParamArc();
    mkSummuryArc();
#print "Pre begin :",‘date‘;
mkPreDependence();
#print "Post begin :",‘date‘;
mkPostDependence();
#print "end :",‘date‘;
    printAllDependence();
    mkslice();
}


sub mkNodeList()
{
    # DUN $NF~NO7A<O
    # @nodelist $OOJ2<$N%a%s%P!<$r;}$D
    #   member  type          contents
    #-----------------------------------------------------
    #   lin     int          9THV9f
    #   con     int array    @\B3%N!<%IHV9f%j%9%H
    #   def     int array    Dj5AJQ?tL>%j%9%H
    #   use     int array    ;HMQJQ?tL>%j%9%H
    #
    #   fork                 @\B3%N!<%IHV9f%j%9%H
    #   join                 @\B3%N!<%IHV9f%j%9%H
    #   send                 %A%c%s%M%kL>
    #   recv                 %A%c%s%M%kL>
    #   return                4X?tL>
    #   call                 @\B3%N!<%IHV9f%j%9%H
    #   a-in                 <BOz?tMQ!";HMQJQ?tL>%j%9%H$N%j%9%H
    #   a-out                <BOz?tMQ!"Dj5AJQ?tL>%j%9%H
    #   f-in                 2>Oz?tMQ!"Dj5AJQ?tL>%j%9%H
    #   f-out                2>Oz?tMQ!";HMQJQ?tL>%j%9%H

    #   nor     flag         def $O normalized form $G$"$k
    #   upt     flag         $3$N%N!<%I$O%]%$%s%?$r;HMQ$7$F$$$k
    #   dpt     flag         $3$N%N!<%I$O%]%$%s%?$rDj5A$7$F$$$k
    #   dom     int          8eB3;YG[;R$N%N!<%IHV9f

    #   arg                  procedure dependence net MQ

    # @nodelist, @varlist $O global
    #
    my $node, $line, $con, $scon, $def, $use,
        $fork, $join, $send, $receive, $param,
        $call, $ain, $aout, $fin, $fout, $pre, $post,
        $normalized, $usepointer,$defpointer;
    my @conlist, @sconlist, @deflist, @uselist,
```

```
        @forklist, @joinlist,@sendlist,@receivelist,@paramlist,
        @calllist, @ainlist, @aoutlist, @finlist, @foutlist;

    #$DEBUG = 1;
$Filename = $ARGV[0];
$Filename =~ s/.dun//;
    while (<>){
($node, $line, $con, $scon, $def, $use,
          $fork, $join, $send, $receive, $return,
          $call, $ain, $aout, $fin, $fout, $pre, $post) = ();
chomp;
next if($_ == ());
s/(\d+): +<line> +([a-zA-Z_,.:0-9> -]+) *// || die "Format error at line $.\n";
($node, $line) = ($1, $2);

($con) = ($1) if (s/<connect> +([0-9 ]+)//);
($scon)= ($1) if (s/ *<s-connect> +([0-9 ]+)//);
($pre) = ($1) if (s/<Pre-connect> +([0-9 ]+)//);
($post) = ($1) if (s/<Post-connect> +([0-9 ]+)//);
$con .= $scon;
($def) = ($1) if (s/ *<def> +([a-zA-Z_,.:0-9> \-\(\)]+)//);
($use) = ($1) if (s/ *<use> +([a-zA-Z_,.:0-9> \-\(\)]+)//);
        ($fork)= ($1) if (s/ *<fork> +([0-9 ]+)//);
        ($join)= ($1) if (s/ *<join> +([0-9 ]+)//);
        ($send)= ($1) if (s/ *<send> +([a-zA-Z!_,.:0-9> \-]+)//);
($receive)=($1) if (s/ *<receive> +([a-zA-Z!_,.:0-9> \-]+)//);
        ($return) =($1) if (s/ *<return> +([a-zA-Z_,.:0-9> "\-\(\)\*]+)//);

@calllist = @ainlist = @aoutlist =();

while(/ *<call> +([0-9 ]+)/)
{
    $call = $ain = $aout ='';

    if (s/ *<call> +([0-9 ]+)//){
              ($call)= ($1);
              push(@entrys,$call);
            }
    ($ain) = ($1) if (s/ *<a-in> +([a-zA-Z_,.\|:0-9> \-\(\)]+)//);
    if (s/ *<a-out> +([a-zA-Z_,.\|:0-9> \-\(\)]+)//){
($aout)= ($1) ;
    }
    push(@calllist,$call);
    push(@ainlist,$ain);
    push(@aoutlist,$aout);
}

($fin) = ($1) if (s/ *<f-in> +([a-zA-Z_,.|:0-9> \-\(\)]+)//);
($fout)= ($1) if (s/ *<f-out> +([a-zA-Z_,.|:0-9> \-\(\)]+)//);
@prelist = split(/ +/, $pre);
@postlist = split(/ +/, $post);
@conlist = split(/ +/, $con);
@sconlist = split(/ +/, $scon);
@deflist = split(/ +/, $def);
@uselist = split(/ +/, $use);
        @forklist  = split(/ +/, $fork);
        @joinlist  = split(/ +/, $join);
        @sendlist  = split(/ +/, $send);
        @receivelist = split(/ +/, $receive);

        $normalized = $usepointer = $defpointer = 0;

foreach(@deflist){
    $normalized = 1 if (/Normal/);
    $defpointer = 1 if (/^ptr/);
    s/^(.*),([^,]+)$/$2/;
            push(@declare,$node) if ($1 =~ /^declare/);
}
foreach(@uselist){
    $usepointer = 1 if (/^ptr/);
    s/^(.*),([^,]+)$/$2/;
            push(@protect,$_) if ($1 =~ /^protected/);
#            print @protect,"\n";
}

# @varlist = &orSet(@varlist, @deflist, @uselist);

$nodelist[$node] = {
    "lin" => $line,
    "con" => [@conlist],
             "scon" =>[@sconlist],
    "def" => [@deflist],
    "use" => [@uselist],
             "fork" => [@forklist],
    "join" => [@joinlist],
    "send" => [@sendlist],
    "receive" => [@receivelist],
             "return" => $return,
             "call" => [@calllist],
             "a-in" => [@ainlist],
             "a-out" => [@aoutlist],
             "pre" => [@prelist],
     "post" => [@postlist],
             "f-in" =>  $fin,
             "f-out" => $fout,
    "nor" => $normalized,
    "upt" => $usepointer,
    "dpt" => $defpointer,
};
    }
#‘    $DEBUG = 1;
}

    &printstatus;
sub mkArgnodeControldependence
#();
{
    foreach $n (0..$#nodelist){
next if(!defined $nodelist[$n]);
```

```
$cdependence[$n] = [$nodelist[$n]->{'arg'}] if(defined $nodelist[$n]->{'arg'})
    }
}

sub printstatus
{
    # DEBUG %3!<%I
#    $DEBUG = 1;
    if ($DEBUG) {
for ($i = 1; $i <= $#nodelist; $i++){
    if(defined ($nodelist[$i])){
    print "$i: <line> $nodelist[$i]->{'lin'} ";
    if(@{$nodelist[$i]->{'con'}}){ #5nY\defined
print "<connect> @{$nodelist[$i]->{'con'}} ";
    }
    if(@{$nodelist[$i]->{'scon'}}){ #5nY\defined
print "<s-connect> @{$nodelist[$i]->{'scon'}} ";
    }
    if(@{$nodelist[$i]->{'def'}}){ #5nY\defined
print "<def> @{$nodelist[$i]->{'def'}} ";
    }
    if(@{$nodelist[$i]->{'use'}}){ #5nY\defined
print "<use> @{$nodelist[$i]->{'use'}}";
    }

    if(@{$nodelist[$i]->{'fork'}}){ #5nY\defined
print "  <fork> @{$nodelist[$i]->{'fork'}} ";
    }
    if(@{$nodelist[$i]->{'join'}}){ #5nY\defined
print "<join> @{$nodelist[$i]->{'join'}} ";
    }
    if(@{$nodelist[$i]->{'send'}}){ #5nY\defined
print "<send> @{$nodelist[$i]->{'send'}} ";
    }
    if(@{$nodelist[$i]->{'receive'}}){ #5nY\defined
print "<receive> @{$nodelist[$i]->{'receive'}}";
    }
    if(defined $nodelist[$i]->{'return'}){
print "<return> $nodelist[$i]->{'return'} ";
    }
    if(@{$nodelist[$i]->{'param'}}){ #5nY\defined
print "<param> @{$nodelist[$i]->{'param'}}";
    }

    if(@{$nodelist[$i]->{'call'}}){ #5nY\defined
print "  <call> @{$nodelist[$i]->{'call'}} ";
    }
    if(@{$nodelist[$i]->{'a-in'}}){ #5nY\defined
print "<a-in> @{$nodelist[$i]->{'a-in'}} ";
    }
    if(@{$nodelist[$i]->{'a-out'}}){ #5nY\defined
print "<a-out> @{$nodelist[$i]->{'a-out'}} ";
    }
    if(@{$nodelist[$i]->{'recall'}}){ #5nY\defined
print "  <recall> @{$nodelist[$i]->{'recall'}} ";
    }

    if(defined $nodelist[$i]->{'f-in'}){
print "  <f-in> $nodelist[$i]->{'f-in'} ";
    }
    if(defined $nodelist[$i]->{'f-out'}){
print "<f-out> $nodelist[$i]->{'f-out'}";
    }

    if(defined $nodelist[$i]->{'arg'}){
print "  <arg> $nodelist[$i]->{'arg'}";
    }
    print "\n";
}
}
    }
}

################## --- 6&DL%k!<%A%s ---
sub printAllDependence # (void)
{
open Tarfile, " >$Filename.sdn ";
    # 40@.$7$? @ddependence, @cdependence $rI=<(

    for ($n = 1; $n <= $#nodelist; $n++) {

# perl $,:n$kM>J,$J%N!<%I$O%9%-%C%W
next if (!defined $nodelist[$n]);
print "$n: <line> $nodelist[$n]->{'lin'}";
print Tarfile "$n: <line> $nodelist[$n]->{'lin'}";

if (@{$cdependence[$n]}) {
    print " <control> @{$cdependence[$n]}";
    print Tarfile " <control> @{$cdependence[$n]}";
}
if (@{$ddependence[$n]}) { #5nY\defined
    print " <data> @{$ddependence[$n]}";
    print Tarfile " <data> @{$ddependence[$n]}";
}
if (@{$sldependence[$n]}) { #5nY\defined
    print " <sele> @{$sldependence[$n]}";
    print Tarfile " <sele> @{$sldependence[$n]}";
}
if (@{$sdependence[$n]}) { #5nY\defined
    print " <sync> @{$sdependence[$n]}";
    print Tarfile " <sync> @{$sdependence[$n]}";
}
if (@{$mdependence[$n]}) { #5nY\defined
    print " <comm> @{$mdependence[$n]}";
    print Tarfile " <comm> @{$mdependence[$n]}";
}
if (@{$callarc[$n]}) { #5nY\defined
    print " <call> @{$callarc[$n]}";
```

```
    print Tarfile " <call> @{$callarc[$n]}";
}
if (@{$paramin[$n]}) { #5nY\defined
    print " <param-in> @{$paramin[$n]}";
    print Tarfile " <param-in> @{$paramin[$n]}";
}
if (@{$paramout[$n]}) { #5nY\defined
    print " <param-out> @{$paramout[$n]}";
    print Tarfile " <param-out> @{$paramout[$n]}";
}
if (@{$summary[$n]}) { #5nY\defined
    print " <summary> @{$summary[$n]}";
    print Tarfile " <summary> @{$summary[$n]}";
}
if (@{$predependence[$n]}) {
print " <pre> @{$predependence[$n]}";
print Tarfile " <pre> @{$predependence[$n]}";
}
if (@{$postdependence[$n]}) {
print " <post> @{$postdependence[$n]}";
print Tarfile " <post> @{$postdependence[$n]}"
}
print "\n";
print Tarfile "\n";
    }
    print "SDN generated.\n";
close Tarfile;
}

sub andSet # (\@list, \@list2)
{
    # =89g$N and $r<h$k

    my ($a,$b) = @_;
    local(%mark);
    grep($mark{$_}++,@$a);
    return (grep($mark{$_},@$b));
}

sub orSet # (@ist1, @list2)
{
    # @list1 $H @list2 $N OR =89g$rJV$9

    local(%keylist);
    foreach $key (@_){
$keylist{$key} = 1;
    }
    return keys(%keylist);
}

sub dprint
{
    print @_ if $DEBUG;
}
```

B.1.2 Control Package

```
#!/usr/local/bin/perl5


sub printControlDependence # (void)
{

    # mkControlDependence
    for ($n = 1; $n <= $#nodelist; $n++) {
# perl
next if (!defined $nodelist[$n]);

print "$n: <line> $nodelist[$n]->{'lin'}";

if (@{$cdependence[$n]}) {
    print " <control> @{$cdependence[$n]}";
}
print "\n";
    }
}

sub mkControlDependence # (void)
{
    my @cFather, @nFather;
    my $c, $n, %MARK;
#    $DEBUG = 1;
    # $nodelist[]{'dom'}   mkPostDominator();

    for ($n = 1; $n <= $#nodelist; $n++) {
dprint "control computing: $n ...";
# perl
next if (!defined $nodelist[$n]);

#
if (defined  $nodelist[$n]->{'con'}[1]) {

    #   @nFather = getDomParents($n);

    #
    foreach $c (@{$nodelist[$n]->{'con'}}) {

# @cFather = getDomParents($c);

#
#
# cFather
# {c} + cFather - nFather
if (grep($n == $_, @cFather) ||
    andSet(\@cFather, \@nFather)){

    # {c} + cFather - nFather
```

```
    grep($MARK{$_}++, @nFather);
    @result = orSet(@result, grep(!$MARK{$_}, ($c, @cFather)));#@result = grep(!$MARK{$_}, ($c, @cFather))
    undef %MARK;

    dprint("@result depends on $n\n") if (@result);

    # cdependence
    foreach (@result) {
push(@{$cdependence[$_]}, $n);
    }
}
}
dprint "... done.\n";
    }
}



sub mkPostDominator #(void)
{
    my @existdom,@route;
    my $node,$startnode, $current;
    my $postdomtree;

    # $nodelist[]{'dom'}
    my $n, $h;

    for ($n = 1; $n <= $#nodelist; $n++) {
# perl
dprint "make dominator: $n ...";
next if (!defined $nodelist[$n]);

if (defined  $nodelist[$n]->{'con'}[1]) {
    #
$h = getCrossRoad($n);
$nodelist[$n]->{'dom'} = @{$h}[0];

#     unshift (@undecidenodeQ,$n);
} else {

    #
    $nodelist[$n]->{'dom'} =  $nodelist[$n]->{'con'}[0] || $n;
    dprint("$n  dominator is $nodelist[$n]->{'dom'}\n");
        }
    }
    dprint "...done.\n";
}

sub hetare {

#  {while begin}
#    $DEBUG =1;
    while (@undecidenodeQ){
#
dprint @undecidenodeQ,"\n";
# <STDIN>;
$node = shift @undecidenodeQ;
@routeQ = @{$nodelist[$node]->{'con'}};
while(@routeQ){

    $start = shift @routeQ;
    $route[$start] = Checkroute($start);

    my $status = pop(@{$route[$start]});
    my @dummy = @{$route[$start]};
    $status = "loop." if ($node == pop(@dummy) );

    $stat{"$start"} = $status;
    dprint "node: $node candidate: $start Path: ",@{$route[$start]}," ",$stat{"$start"},"\n";
}
@routeQ = @{$nodelist[$node]->{'con'}};
push(@undecidenodeQ,$node) if (setdominator($node,@routeQ) eq "next.");
    }

}

sub Reachable
# ($n1);
{
    my $startnode = shift;

    my @history;
    my @candidate = @{$nodelist[$startnode]->{'con'}};
    push(@history,$startnode);
    while(@candidate){
dprint @history,"\n";
# <STDIN>;
$startnode = shift @candidate;
if (grep($_ == $startnode,@history)){
    #
}else{
    push(@history,$startnode);
    push(@candidate,@{$nodelist[$startnode]->{'con'}});
}
    }
    [@history];
}


sub setdominator
{
    my $setnode = shift;
    my @starts =@_;
    my $n1,$n2;
    my $msg = "null";
    $n1 = shift @starts;
    $n2 = shift @starts;
    if (($stat{$n1} eq "terminate.") && ($stat{$n2} eq "terminate.")){
my @dummy = andSet(\@{$route[$n1]},\@{$route[$n2]});
```

```
$nodelist[$setnode]->{'dom'} = $dummy[0] || $setnode;
dprint "node : ",$setnode," dom : ",$nodelist[$setnode]->{'dom'},"\n";
dprint "two terminate.\n";
    }elsif(($stat{$n1} eq "terminate.") && ($stat{$n2} eq "undefined.")){
# $n2
$h = Reachable($n2);
@dummy = andSet(\@{$route[$n1]},\@{$h});
dprint "OK? ",@dummy,"\n";
if (@dummy ==() ){
    $msg = "next.";
    print "termnate-undef.\n";
}else{
    $nodelist[$setnode]->{'dom'} = $dummy[0];
    dprint "node : ",$setnode," dom : ",$nodelist[$setnode]->{'dom'},"\n";
}
    }elsif(($stat{$n1} eq "terminate.") && ($stat{$n2} eq "loop.")){
$nodelist[$setnode]->{'dom'} = @{$route[$n1]}[0];
dprint "node : ",$setnode," dom : ",$nodelist[$setnode]->{'dom'},"\n";
dprint "termnate-loop.\n";
    }elsif(($stat{$n1} eq "undefined.") && ($stat{$n2} eq "terminate.")){
# $n1
$h = Reachable($n1);
@dummy = andSet(\@{$route[$n2]},\@{$h});
dprint "OK? ",@dummy,"\n";
if (@dummy ==() ){
    $msg = "next.";
    dprint "undef-terminate.\n";
}else{
    $nodelist[$setnode]->{'dom'} = $dummy[0];
    dprint "node : ",$setnode," dom : ",$nodelist[$setnode]->{'dom'},"\n";
}
    }elsif(($stat{$n1} eq "undefined.") && ($stat{$n2} eq "undefined.")){
my @dummy = andSet(\@{$route[$n1]},\@{$route[$n2]});
if (@dummy == ()){
    $msg = "next.";
}else{
    $nodelist[$setnode]->{'dom'} = $dummy[0];
    dprint "node : ",$setnode," dom : ",$nodelist[$setnode]->{'dom'},"\n";
}
dprint "two undef.\n";
    }elsif(($stat{$n1} eq "undefined.") && ($stat{$n2} eq "loop.")){
$nodelist[$setnode]->{'dom'} = @{$route[$n1]}[0];
dprint "node : ",$setnode," dom : ",$nodelist[$setnode]->{'dom'},"\n";
dprint "undef-loop.\n";
    }elsif(($stat{$n1} eq "loop.") && ($stat{$n2} eq "terminate.")){
$nodelist[$setnode]->{'dom'} = @{$route[$n2]}[0];
dprint "node : ",$setnode," dom : ",$nodelist[$setnode]->{'dom'},"\n";
dprint "loop-terminate.\n";
    }elsif(($stat{$n1} eq "loop.") && ($stat{$n2} eq "undefined.")){
$nodelist[$setnode]->{'dom'} = @{$route[$n2]}[0];
dprint "node : ",$setnode," dom : ",$nodelist[$setnode]->{'dom'},"\n";
dprint "loop-undefined.\n";
    }elsif(($stat{$n1} eq "loop.") && ($stat{$n2} eq "loop.")){
my @dummy = andSet(\@{$route[$n1]},\@{$route[$n2]});
$nodelist[$setnode]->{'dom'} = $dummy[0];
dprint "node : ",$setnode," dom : ",$nodelist[$setnode]->{'dom'},"\n";
dprint "loop-loop.\n";
    }else{
dprint "something else.";
exit(1);
    }
    $msg;
}

sub Checkroute
{
    my $startnode = shift;
    my $move = $startnode;
    my @localroute;


    if ($startnode == $nodelist[$startnode]->{'dom'} ){
push(@localroute,$startnode);
push(@localroute,"terminate.");
    }else{
while (defined $nodelist[$move]->{'dom'}){
#     print $move,"\n";
#     <STDIN>;
    push(@localroute,$move);
    $move = $nodelist[$move]->{'dom'};
    if ($move == $nodelist[$move]->{'dom'} ){
# push(@localroute,$move);
last;
    }
}
push(@localroute,$move);
if ($move == $nodelist[$move]->{'dom'} ){
    push(@localroute,"terminate.");
}else{
    push(@localroute,"undefined.");
}
    }
    [@localroute];
}

sub getDomParents #(nodenumber $n)
{

    # $nodelist[]{'dom'}
    #

    my $n = shift;
    my @ret;


    while (defined $nodelist[$n]->{'dom'}
   && $nodelist[$n]->{'dom'} != $n){
$n = $nodelist[$n]->{'dom'};
push(@ret, $n);
```

```
    }
    return @ret;
}


sub getCrossRoad
{

    getCrossRoadWorkhorse(shift, undef);
}

sub getCrossRoadWorkhorse #(startnode, path_history)
{



    my $start = shift;

    my $h = shift;
    my @history = @{$h};
    my @ret = (1 .. $#nodelist);
    dprint("getCrossRoadWorkhorse $start\n");
    dprint("history = @history\n");


    return undef if (! defined  $nodelist[$start]->{'con'}[0]);

    #  foreach $n (@{$nodelist[$start]->{'con'}}) {
if (grep($_ eq "$start,$n", @history)){
    #   @ret = (1..$#nodelist);
} else {
    #  push(@history, "$start,$n");
    @ret = andSet(\@ret,
  [($n, @{getCrossRoadWorkhorse($n, \@history)})]);
    pop(@history);
}
    }
    dprint("ret = @ret\n");
    return [@ret];
}
```

B.1.3 Data Package

```
#!/usr/local/bin/perl5

################### --- data dependence ---
# $Id: data.pl,v 1.3 2015/8/23 22:15 wangbo Exp $

sub printDataDependence # (void)
{
open Tarfile, " >$Filename.sdn ";
    # 完成した @ddependence を表示

    for ($n = 1; $n <= $#nodelist; $n++) {

# perl が作る余分なノードはスキップ
next if (!defined $nodelist[$n]);

print "$n: <line> $nodelist[$n]->{'lin'}";
print Tarfile "$n: <line> $nodelist[$n]->{'lin'}";
if (@{$ddependence[$n]}) { #去掉 defined
    print " <data> @{$ddependence[$n]}";
    print Tarfile " <data> @{$ddependence[$n]}"
}
print "\n";
print Tarfile "\n";
    }
}

sub mkDataDependence # (void)
{
    # ノードリスト @nodelist から データ依存グラフ @ddependence を作る

    my $change, $n, $var, $con, $items, @F;
    my $use, $tonode;
    my @loop =( 1.. $#nodelist );
    my $loopcheck = 0;

    #到達性を検査
    while(@loop)
    {
     $loopcheck++;
undef @history;
undef @varlist;
undef @candidate;
my $current = $loop[0];
my @history,@varlist;
my @candidate = @{$nodelist[$current]->{'con'}};
push(@history, $current);
@varlist = &orSet(@varlist,
  @{$nodelist[$current]->{'def'}},
  @{$nodelist[$current]->{'use'}});
     while(@candidate)
     {
$current = shift @candidate;
if (grep ($_ == $current ,@history))
{
     #
}else
{
     push(@history, $current);
     @varlist = &orSet(@varlist,
       @{$nodelist[$current]->{'def'}},
       @{$nodelist[$current]->{'use'}});
     push(@candidate, @{$nodelist[$current]->{'con'}});
}
```

```
    }
local(%mark);
grep($mark{$_}++,@history);
@loop = grep(!$mark{$_},@loop);
$checkfff = 0;

# まず、変数定義の到達ノードリスト F を作る
do {
$checkfff++ ;
    $change = 0;
    # 全ノードについて
    foreach $n (@history){
next if (!defined $nodelist[$n]);

# 各変数について
foreach $var (@varlist){

    # n から con に枝が伸びている
    foreach $con (@{$nodelist[$n]->{'con'}}){

# n で var が定義されていれば、
if (grep(/^$var$/, @{$nodelist[$n]->{'def'}})){
    $items = $#{$F[$con]->{$var}};
    # n の先の con に n から var の定義が伝わるとして、
    # 集合 F(con)(var) に n を加える

    @{$F[$con]->{$var}} =
&orSet(@{$F[$con]->{$var}}, ($n));
    print"$loopcheck, $checkfff, $var, $con, $n \n";
    # items が増えていれば F に変化があったとマーク+
    $change = 1 if ($items < $#{$F[$con]->{$var}});

} else {
    $items = $#{$F[$con]->{$var}};
    # そうでなければ、F(con)(var) には n での内容がその
    # まま伝わる
    @{$F[$con]->{$var}} = &orSet(@{$F[$con]->{$var}},
 @{$F[$n]->{$var}});
    $change = 1 if ($items < $#{$F[$con]->{$var}});
}
    }
}
    }
} while ($change == 1);

# できた F をもとにして データ依存 @ddependence を作る
foreach $n (@history){

    # perl が作る余分なノードはスキップ
    next if (!defined $nodelist[$n]);

    foreach $use (@{$nodelist[$n]->{'use'}}){
foreach $tonode (@{$F[$n]->{$use}}){
    push(@{$ddependence[$n]}, $tonode);
    print"line $nodelist[$n]->{'lin'} ($n) to ";
    print"$nodelist[$tonode]->{'lin'} ($tonode) about $use\n";
}
    }
}
    }
}

1;
```

B.1.4 Sync Package

```
#!/usr/local/bin/perl5

#
################## ---Synchronization Dependence---
#    mkSyncDependence();
#    printSyncDependence();
#    connectsync(nodenumber $v1,$v2);
########

#$DEBUG = 1;

sub mkSyncDependence #(void)
{
# 並行実行枝ならば、同期従属弧を引く。

    for ($v = 1; $v <= $#nodelist; $v++) {
if(defined $nodelist[$v]->{'fork'}){
    foreach $u(@{$nodelist[$v]->{'fork'}}){
&connectsync($u,$v);
    }
}
if(defined $nodelist[$v]->{'join'}){
    foreach $u(@{$nodelist[$v]->{'join'}}){
&connectsync($u,$v);
    }
}
    }

#その他の同期依存弧を引く。

    for ($v = 1; $v <= $#nodelist; $v++) {
if ( @{$nodelist[$v]->{'send'}}){ #去掉 defined
    dprint "@{$nodelist[$v]->{'send'}} <-\n";
    for($u = 1; $u <= $#nodelist; $u++){
next if (!@{$nodelist[$u]->{'receive'}}); #去掉 defined

#ここで集合 S(v) と R(u) の比較を行なっている。
#本来なら、関数にして正確にすべき、潜在的なバグがここにある。
# print "<sd> ",$sd++,'date';
if ("@{$nodelist[$v]->{'send'}}"
```

```
    eq "@{$nodelist[$u]->{'receive'}}"){
    &connectsync($u,$v);
}
    }
}
    }
}

sub connectsync #(nodenumber $v1,$v2)
{
    my $v;

    $v1 = shift;
    $v2 = shift;

# $v1 -> $v2 に 弧を引く。

    push(@{$sdependence[$v1]},$v2);

    $v = $v1;
    while (scalar(@{$nodelist[$v]->{'con'}}) == 1){
$v = @{$nodelist[$v]->{'con'}}[0];
last if ( (@{$nodelist[$v]->{'send'}} ) #去掉 defined
 || ( @{$nodelist[$v]->{'receive'}}) ); #去掉 defined
push(@{$sdependence[$v]},$v2);
    }
}

sub printSyncDependence # (void)
{
    # 完成した @sdependence を表示

    for ($n = 1; $n <= $#nodelist; $n++) {

# perl が作る余分なノードはスキップ
next if (!defined $nodelist[$n]);

print "$n: <line> $nodelist[$n]->{'lin'}";
if (@{$sdependence[$n]}) { #去掉 defined
    print " <sync> @{$sdependence[$n]}";
}
print "\n";
    }
}

1;
```

B.1.5 Select Package

```
#!/usr/local/bin/perl5

#
########################## --- selection dependence
#    mkSelectDependence();
#    printSelectDependence();
##########################

sub mkSelectDependence
{
    my @nondeterministicnodes,@dummy,@startnodes;

        for ($n = 1; $n <= $#nodelist; $n++) {
    # perl が作る余分なノードはスキップ
    next if (!defined $nodelist[$n]);

    if (@{$nodelist[$n]->{'scon'}}){ #去掉 defined
#非決定的選択点を求める。
push (@nondeterministicnodes,$n);
    }
}

    #これらの点につながっている制御従属枝は、選択従属である。

    foreach $i(@nondeterministicnodes){
undef @startnodes;
for ($n = 1; $n <= $#nodelist; $n++) {
    # perl が作る余分なノードはスキップ
    next if (!defined $nodelist[$n]);

    #この点$i に制御従属している節点を求める。
    @dummy = @{$cdependence[$n]};
    push(@startnodes,$n) if (grep($_ == $i, @dummy));
}

foreach $s (@startnodes){
    #選択従属枝を引く。
    push (@{$sldependence[$s]},$i);

    #制御従属枝をはずす。
    @{$cdependence[$s]} = &cutsameelement(@{$cdependence[$s]});
    splice(@{$cdependence[$s]},
   &searchforArray(\@{$cdependence[$s]},\$i),1)
    }
    }
}



sub printSelectDependence
{
    # 完成した @sldependence を表示

    for ($n = 1; $n <= $#nodelist; $n++) {

# perl が作る余分なノードはスキップ
next if (!defined $nodelist[$n]);
```

```
print "$n: <line> $nodelist[$n]->{'lin'}";
if (@{$sldependence[$n]}) { #去掉 defined
    print " <sele> @{$sldependence[$n]}";
}
print "\n";
    }
}

sub searchforArray
{
    my @array;
    my $element,$index;
    @array = shift;
    $element =shift;

    @array = &cutsameelement(@array);
    for(0 .. $#array){
$index = $_,last if(@array[$_] == $element);
    }
    $index;
}

sub cutsameelement #(@multi)
{
    my @dummy;

#仕方のない処理 -> 配列中に 同じ値の要素は複数ないようにする。

    foreach $item(@_){
push (@dummy,$item) if (!grep($_ == $item,@dummy));
    }
    @dummy;
}

1;
```

B.1.6 Comm Package

```
#!/usr/local/bin/perl5
################### ---Communication Dependence  ---
#    mkCommDependence();
#    connectcomm();
#    connectcomm();
#    printCommDependence();
##########

sub printCommDependence # (void)
{
    # 完成した @mdependence を表示

    for ($n = 1; $n <= $#nodelist; $n++) {

# perl が作る余分なノードはスキップ
next if (!defined $nodelist[$n]);

print "$n: <line> $nodelist[$n]->{'lin'}";
if ( @{$mdependence[$n]}) { #去掉 defined
    print " <comm> @{$mdependence[$n]}";
}
print "\n";
    }
}

sub mkCommDependence # (void)
{
    my $v,$u;

    # ノードリスト @nodelist から 通信依存グラフ @mdependence を作る
    # send - receive に関わる通信依存弧を引く。
    for($v = 1;$v<= $#nodelist; $v++){
if (@{$nodelist[$v]->{'send'}}){ #去掉 defined
    for($u = 1; $u <= $#nodelist; $u++){
next if (!@{$nodelist[$u]->{'receive'}}); #去掉 defined
if ("@{$nodelist[$v]->{'send'}}"
    eq "@{$nodelist[$u]->{'receive'}}"){
    dprint "@{$nodelist[$v]->{'send'}}:$v -> @{$nodelist[$u]->{'receive'}}:$u\n";
    &connectcomm($u,$v);

}
    }
}
    }
#    print "comm dep.end ",`date`;

    #大域変数に対して通信依存弧を引く。
#jh    for($v = 1;$v <= $#nodelist; $v++){
#jh if (defined @{$nodelist[$v]->{'use'}}){
#jh     for($u = 1; $u <= $#nodelist; $u++){
#jh next if (!defined @{$nodelist[$u]->{'def'}});
#jh dprint "@{$nodelist[$v]->{'use'}}:$v -> @{$nodelist[$u]->{'def'}}:$u\n";
#jh if (andSet(\@{$nodelist[$v]->{'use'}},
#jh    \@{$nodelist[$u]->{'def'}})){
#jh
#     &connectcommG($v,$u);
#jh }
#jh     }
#jh }

    complementComm();
}

sub complementComm()
{
    #protected 型の変数を求めておく。
    #この変数が使用される頂点のうち、
    #receive ノードから初めて到達する頂点が、
```

```
    #ひとまずの候補である。

    #ノードリストから、receive ノードを求める。
    foreach(1..$#nodelist){
push(@receivenodes,$_) if (@{$nodelist[$_]->{'receive'}}); #去掉 defined
    }

    foreach(@receivenodes){
#print @receivenodes,":r-nodes.\n";
$n = getNodeUsingprotectedvalue($_);
#print $n,"\n";
push(@protectnodes,$n);
    }
    @protectnodes = orSet(@protectnodes);
    #この protected 型の変数が宣言されているところへ通信従属である
    foreach(@protectnodes){
foreach $l (@declare){
    push(@{$mdependence[$_]},$l) if (andSet(\@{$nodelist[$_]->{'use'}},\@{$nodelist[$l]->{'def'}}));
}
    }

    foreach(@protectnodes){
#到達可能な頂点上からの send を行っているチャンネルを求める。
my @channels = getchannelfromsend($_);
# print @channels,":channel $_\n";
foreach $channel (orSet(@channels)){
    my @candidate = getreceivenodes($channel);
#     print @candidate,"receive candidate.\n";
    foreach $n (@candidate){
#もし、$n から$_に到達すれば、ここでは何も行わない。
#そうでなければ、最後に protect 型の変数を定義したノードを返す。
#そこへ通信従属枝を引く。
$m = getNodeDifiningprotectvariable($n,$_);
# print $m,":tonode\n";
push(@{$mdependence[$_]},$m) if $m;
undef $m;
    }
}
    }
}
sub getNodeDifiningprotectvariable
#($n,$_)
{
    my @candidate = (shift);
    my $usingnode = shift;
    #   print $usingnode,":using node.\n";
    my @history;
    undef $last;
    while(@candidate){
my $n = shift @candidate;
$last = $n if (andSet(\@protect,\@{$nodelist[$n]->{'def'}}));
unless (grep($_ == $n,@history)){
    @candidate = (@candidate,@{$nodelist[$n]->{'con'}});
    push (@history,$n);
}

    }
#    print @history,":history.\n";
#    print grep($_ == $usingnode,@history),"and.\n";
    if (grep($_ == $usingnode,@history)){
# print "self\n";
return undef;
    }else{
return $last;
    }
}
sub getreceivenodes
#($channel);
{
    my @ret;
    $channel = shift;
    foreach $i (@receivenodes){
push(@ret,$i) if(grep($_ eq $channel,@{$nodelist[$i]->{'receive'}}));
    }
    @ret;
}

sub getchannelfromsend
#($_);
{
    my @candidate = (shift);
    my @history,@ret;
    while(@candidate){
my $n = shift @candidate;
@ret = (@ret,@{$nodelist[$n]->{'send'}}) if(@{$nodelist[$n]->{'send'}}); #去掉 defined
unless (andSet(\@history,[($n)])){
    @candidate = (@candidate,@{$nodelist[$n]->{'con'}});
    push (@history,$n);
}

    }
    return @ret;
}

sub getNodeUsingprotectedvalue
#($_);
{
    my @candidate = (shift);
    my @history;
undef @deflist;
    while(@candidate){
my $n = shift @candidate;
if (andSet(\@protect,\@{$nodelist[$n]->{'use'}})){
    if (@deflist){
return(undef) if(andSet(\@deflist,\@protect));
    }else{
return($n);
    }
}
```

```
unless (andSet(\@history,[($n)])){
    @candidate = (@candidate,@{$nodelist[$n]->{'con'}});
    @deflist = (@deflist,@{$nodelist[$n]->{'def'}});
    push (@history,$n);
}

    }
    return undef;
}
sub connectcomm #(nodenumber $v1,$v2)
{
    my $v1,$v2;
    my @dummy,@startnodes,@endnodes;

    $v1 = shift;
    $v2 = shift;

    undef @startnodes;
    for ($i = 1; $i < $#nodelist ; $i++){
        @dummy = @{$ddependence[$i]};
        if (grep($_ == $v1, @dummy)){
dprint "@dummy:@{$ddependence[$i]};$i\n";
push(@startnodes,$i);
}
    }

    @endnodes   = @{$ddependence[$v2]};

    foreach $s(@startnodes){
foreach $e(@endnodes){
    dprint "$s -> $e \n";
    push(@{$mdependence[$s]},$e);
}
    }
}

sub connectcommG #(nodenumber $usev,$defv)
{
    my $usev,$defv;
    my @dummy,@usevalues,@defvalues;

    $usev = shift;
    $defv = shift;
    @usevalues = @{$nodelist[$usev]->{'use'}};
    @defvalues = @{$nodelist[$defv]->{'def'}};

    undef @dummy;
    foreach $s(@usevalues){
foreach $e(@defvalues){
    if ("$s" eq "$e"){
@dummy = @{$ddependence[$usev]};
# dprint "$s:$usev -> $e:$defv =>@dummy\n";
push(@{$mdependence[$usev]},$defv) unless(grep($_ == $defv,@dummy));
    }
}
    }
}

1;
```

B.1.7 Call Package

```
#!/usr/local/bin/perl5


# $Id: call.pl,v 2.0 2016/08/19 18:50 wangbo Exp $
################### --- call relation arc ---
#    mkCallArc();
#    mkParamArc();
#    printCallandParamArc();
########

#$DEBUG = 1;

sub mkCallArc #(void)
{

    for ($v = 1; $v <= $#nodelist; $v++) {
if(defined $nodelist[$v]->{'recall'}){
    foreach $u(@{$nodelist[$v]->{'recall'}}){
push(@{$callarc[$u]},$v);
# &connectarc($u,$v);
    }
}
    }
}
sub mkParamArc #(void)
{

    for ($v = 1; $v <= $#nodelist; $v++) {
if(defined $nodelist[$v]->{'param-in'}){
    foreach $u(@{$nodelist[$v]->{'param-in'}}){
push(@{$paramin[$u]},$v);
    }
}
if(defined $nodelist[$v]->{'param-out'}){
    foreach $u(@{$nodelist[$v]->{'param-out'}}){
push(@{$paramout[$u]},$v);
    }
}
    }
    for ($v = 1; $v <= $#nodelist; $v++) {
@{$paramin[$v]} = orSet(@{$paramin[$v]})  if(defined $paramin[$v]);
@{$paramout[$v]}= orSet(@{$paramout[$v]}) if(defined $paramout[$v]);
    }
```

```
}

sub mkSummuryArc
{
    foreach(0..$#callnodes){


next if(!defined $callnodes[$_]->{'a-out'});

my @ain = @{$callnodes[$_]->{'a-in'}};
my @aout = @{$callnodes[$_]->{'a-out'}};
my @aout2= @{$callnodes[$_]->{'a-out'}};

foreach $s (@aout2){
    my @route = &Checksummaryroute($s);
    @{$summary[$s]} = andSet(\@route,\@ain);
}


foreach $s (@aout){
    @ain = @{$summary[$s]};
    foreach $i (0..$#ain){
@datadep = @{$ddependence[$ain[$i]]};
@nextnodes = @{$nodelist[$ain[$i]]->{"con"}}; #ここから辿って
@deflist = @{$nodelist[$s]->{"def"}};
while(@nextnodes){
    $start = shift @nextnodes;
    next if (andSet(\@markd,[$start]));
    push(@markd,$start);
    next if (andSet(\@{$nodelist[$start]->{"def"}},\@deflist));
    push(@{$ddependence[$start]},$s) if (andSet(\@{$nodelist[$start]->{"use"}},\@deflist));
    foreach(@{$nodelist[$start]->{"con"}}){
push(@nextnodes,$_);
    }
}
    }
}
    }

    for ($n = 1; $n <= $#nodelist; $n++) {
next if (!@{$paramout[$n]}); #去掉 defined
next if (@{$summary[$n]}); #去掉 defined
&appendsummaryarc($n);
    }
}

sub appendsummaryarc
{
    my $startnode = shift;
    my $move;
    my @route,@candidate;

    @candidate =($startnode);
    while(@candidate){
$move = shift @candidate;
# print "---->$nodelist[$move]->{'arg'}\n";
push(@{$summary[$startnode]},$move) if ($nodelist[$move]->{'arg'} == $startnode);
unless (grep($_ == $move, @route)){
    push(@route,$move);
    @candidate=(@candidate,@{$paramout[$move]},@{$paramin[$move]},@{$ddependence[$move]});
}
    }
}


sub Checksummaryroute
{
    my $startnode = shift;
    my $move;
    my @route,@candidate;

    @candidate =($startnode);
    while(@candidate){
$move = shift @candidate;
unless (grep($_ == $move, @route)){
    push(@route,$move);
#     print "move : $move param-out : @{$paramout[$move]}";<STDIN>;
    @candidate=(@candidate,@{$paramout[$move]},@{$paramin[$move]},@{$ddependence[$move]});
#     print "candidate :@candidate";<STDIN>;
}
    }
    @route;
}
sub printCallandParamArc # (void)
{
    # 完成した @callarc @paramarc を表示

    for ($n = 1; $n <= $#nodelist; $n++) {

# perl が作る余分なノードはスキップ
next if (!defined $nodelist[$n]);

print "$n: <line> $nodelist[$n]->{'lin'}";
if (@{$callarc[$n]}) { #去掉 defined
    print " <call> @{$callarc[$n]}";
}
if (@{$paramin[$n]}) { #去掉 defined
    print " <param-in> @{$paramin[$n]}";
}
if (@{$paramout[$n]}) { #去掉 defined
    print " <param-out> @{$paramout[$n]}";
}
if (@{$summary[$n]}) { #去掉 defined
    print " <summary> @{$summary[$n]}";
}
print "\n";
    }
}
```

```
#need summary arc.

1;
```

B.1.8 Expandnode Package

```
#!/usr/local/bin/perl

#$Id: expandnode.pl,v 1.8 2016/03/23 21:42 wangbo Exp $
sub expandDUN()
{
    foreach(orSet(@entrys)){
next if (!defined $nodelist[$_]);
dprint "-------------------->",$_,"\n";
&Expandentrynode($_); #%functionname
#    &printfunctionname;
#    <STDIN>;
    }


    foreach(0..$#nodelist){
next if (!defined $nodelist[$_]);
next if (!@{$nodelist[$_]->{'call'}}); #去掉 defined
dprint "-->>>$_\n";
&CreateNesttree($_);
&Expandcallnode($_,$_,0);
    }

#    $DEBUG = 1;
#    &printexpandstatus();
    &setparameterarc();
    &setcallarc();
#    &printexpandstatus();
#    &printstatus();

}

sub Expandentrynode
#(関数の始まるノード番号)
{
    my $startnode = shift;

    #<f-in>について
    #  新しいノードを割り当て
    #  割り当てたノードはもともとどのノードに属するものか?
    #  開始ノードの制御の流れを新しいノードに割り当てる
    #  <f-in>の定義集合を新しいノードに割り当てる
    ##########################################################
    my @finlist = split(/ /,$nodelist[$startnode]->{'f-in'});

    foreach $item(@finlist){
$newnode = &Allocatenewnode();
push(@{$expandnodelist[$startnode]->{'f-in'}}, $newnode);
$nodelist[$newnode]->{"lin"} = $nodelist[$startnode]->{"lin"} ;
$nodelist[$newnode]->{"arg"} = $startnode;
$nodelist[$newnode]->{"con"} = [@{$nodelist[$startnode]->{'con'}}];
@deflist = split(/ /,$item);
        $normalized = $defpointer = 0;
foreach(@deflist){
    $normalized = 1 if (/Normal/);
    $defpointer = 1 if (/^ptr/);
    s/^.*,([^,]+)$/$1/;
}
$nodelist[$newnode]->{"def"} = [@deflist];
    }
    #<f-out>について
    #  新しいノードを割り当て
    #  割り当てたノードはもともとどのノードに属するか
    #  終了ノードは引数ノードに制御枝を引く。
    #  <f-out>の使用集合を新しいノードに割り当てる
    ##########################################################
#    print "--->>>",$startnode,"\n";
    my @endnode = &Getendnode([$startnode]);  #%functionname
    if (@endnode > 1) {
print "error!@endnode\n" ;
exit;
    }
    my @foutlist = split(/ /,$nodelist[$startnode]->{'f-out'});
    foreach $item(@foutlist){
$newnode = &Allocatenewnode();
push(@{$expandnodelist[$startnode]->{'f-out'}}, $newnode);
$nodelist[$newnode]->{"lin"} = $nodelist[$startnode]->{"lin"} ;
$nodelist[$newnode]->{"arg"} = $startnode;
@uselist = split(/ /,$item);
        $usepointer = 0;
foreach(@uselist){
    $usepointer = 1 if (/^ptr/);
    s/^.*,([^,]+)$/$1/;
}
$nodelist[$newnode]->{"use"} = [@uselist];
        push(@{$nodelist[$endnode[0]]->{'con'}},$newnode);
    }

    #開始ノードは<connect>を破棄してどのノードを引数として持つか割り当てる。
    ##########################################################
    if(@finlist){
$nodelist[$startnode]->{'con'} = [@{$expandnodelist[$startnode]->{'f-in'}}];
dprint "set: ",@{$nodelist[$startnode]->{'con'}},"\n";
    }
}

sub Allocatenewnode
#()
{
    return($#nodelist + 1);
}
```

```
sub printfunctionname
{
    foreach(0..$#expandnodelist){
next if(!@{$expandnodelist[$_]->{'return'}}); #去掉 defined
print $_,":",@{$expandnodelist[$_]->{'return'}},"\n";
print $_,":",$functionname{$_},"\n";
    }
}

sub Getendnode
#($candiate , $route ,$termnate)
{
    my ($candidate ,$route , $termnate)= @_;
    my $startnode;
    #@candidate:未走査ノード
    #@route:既走査ノード
    #@termnate:終端ノード

    print "  --->",@$route,"\n";

    #候補から一つ取りだし、チェック
    #次の接続ノードがあれば候補に入れて呼出
    #  ただし、既に調べてあるものは入れず。
    #次の接続ノードがなければ、終端に入れて呼出
    #候補がなくなるまでやる。
    ##############################################
    if(@$candidate){
$startnode = shift(@$candidate);
unless (grep($_ == $startnode,@$route)){
    #既に通った経路を記憶
    push(@$route,$startnode);
    #経路中に return があれば、%functionname を設定。
    if (defined $nodelist[$startnode]->{'return'}){
$functionname{"$$route[0]"} = $nodelist[$startnode]->{'return'};
push(@{$expandnodelist[$$route[0]]->{'return'}},$startnode);
print $$route[0],":",@{$expandnodelist["$$route[0]"]->{'return'}},"\n";
print $$route[0],":",$functionname{"$$route[0]"},"\n";
    }

    if (@{$nodelist[$startnode]->{'con'}}){ #去掉 defined
@$candidate =&orSet(@$candidate,@{$nodelist[$startnode]->{'con'}});
    }else{
push(@$termnate,$startnode);
    }
}
@$termnate = &Getendnode($candidate,$route,$termnate);
    }
    @$termnate;
}

sub CreateNesttree
#callnode.
{
    my $startnode = shift;
    my $target = 1;
    my @nestdom,@stack,@argument;
    my @ain = @{$nodelist[$startnode]->{'a-in'}};
    @stack =();
#    print " ---->>>",@ain,"\n";
    #入れ子支配木を形成。
    foreach $item(@ain){
#初期値設定
if (@stack == ()){
    $nestdom[$target] = 0;
}else{
    $nestdom[$target] = shift(@stack);
}
# print "  ==>>$target:$nestdom[$target]\n";
push(@argument, split(/[ \|]/,$item));
my $n = &istherefunction(@argument);
for ($i=0;$i<$n;$i++) {
    push(@stack,$target);
}
$target++;
    }
    #準備終
    #入れ子木初期化と生成。
    undef @nest;
    for($i=1;$i<=$#nestdom;$i++){
push(@{$nest[$nestdom[$i]]}, $i);
# print "  ==>>>",$nestdom[$i],":",$i,"\n";
    }
}

sub Expandcallnode
#(<call>のあるノード番号)
{

    my $startnode = shift;
    my $parentnode= shift;
    my $target = shift;
    my @con;
    my @candidate;
    #
    ##############################################
#    print "=======",$startnode,":",$parentnode,":",$target,"\n";
#    <STDIN>;
    if ($target == 0) {
foreach (@{$nest[0]}){
    &Expandcallnode($startnode,$startnode,$_);
}
    }else{
my @ain  = split(/ \| /,@{$nodelist[$startnode]->{'a-in'}}[$target-1]);
my @aout = split(/ \| /,@{$nodelist[$startnode]->{'a-out'}}[$target-1]);
my $newexpandnode = $#callnodes +1;
$callnodes[$newexpandnode]->{'node'} = $parentnode;
$callnodes[$newexpandnode]->{'call'} = @{$nodelist[$startnode]->{'call'}}[$target-1];

@con = @{$nodelist[$parentnode]->{"con"}};
```

```
$nodelist[$parentnode]->{"con"} = [];
$nodelist[$parentnode]->{"con"} = [@con] if (@ain == 0);
  foreach $item(@ain){
    $newnode = &Allocatenewnode();
    push(@{$nodelist[$parentnode]->{"con"}},$newnode);
    push(@{$callnodes[$newexpandnode]->{'a-in'}},$newnode);
    $nodelist[$newnode]->{"lin"} = $nodelist[$parentnode]->{"lin"} ;
    $nodelist[$newnode]->{"arg"} = $parentnode;
    $nodelist[$newnode]->{"con"} = [@con];
    @uselist = split(/ /,$item);
    $usepointer = 0;
    foreach(@uselist){
$usepointer = 1 if (/^ptr/);
s/^.*,([^,]+)$/$1/;
    }
    $nodelist[$newnode]->{"use"} = [@uselist];
    my $n = &istherefunction(@use);
    for($i=0; $i<$n; $i++){
push(@candidate,$newnode);
    }
}
foreach $item(@aout){
    $newnode = &Allocatenewnode();
    push(@{$callnodes[$newexpandnode]->{'a-out'}},$newnode);
    $nodelist[$newnode]->{"lin"} = $nodelist[$parentnode]->{"lin"} ;
    $nodelist[$newnode]->{"arg"} = $parentnode;
    @deflist = split(/ /,$item);
    $normalized = $defpointer = 0;
    foreach(@deflist){
$normalized = 1 if (/Normal/);
$defpointer = 1 if (/^ptr/);
s/^.*,([^,]+)$/$1/;
    }
    $nodelist[$newnode]->{"def"} = [@deflist];
}
foreach(@candidate){
    &Expandcallnode($startnode,$_,shift(@{$nest[$target]}));
}
    }
}

sub istherefunction
{
   my @names = values %functionname;
 #  print @names,"\n";
   return (&andSet(\@_,\@names));
}

sub setcallarc
#();
{
    my $callsrc,$calldest;
    foreach(0..$#callnodes){
$callsrc  = $callnodes[$_]->{'node'};
$calldest = $callnodes[$_]->{'call'};

push(@{$nodelist[$callsrc]->{'recall'}},$calldest);
    }
}
sub setparameterarc
#();
{
    my $calldest;
    foreach(0..$#callnodes){
my $calldest = $callnodes[$_]->{'call'};
my @ain = @{$callnodes[$_]->{'a-in'}};
my @aout= @{$callnodes[$_]->{'a-out'}};
# print $calldest,"\n";
foreach $u (@{$expandnodelist[$calldest]->{'f-in'}}){
    $v = shift @ain;
    push(@{$nodelist[$v]->{'param-in'}},$u);
}
foreach $u (@{$expandnodelist[$calldest]->{'f-out'}}){
    $v = shift @aout;
    push(@{$nodelist[$u]->{'param-out'}},$v);
}
    }
    &CreateReturnarc();
}

sub CreateReturnarc
#
{
    my $callsrc,$calldest;

    #@callnodes をもとに関数ごとに call の場所を求めておく
    foreach(0..$#callnodes){

$callsrc = $callnodes[$_]->{'node'};
$calldest= $callnodes[$_]->{'call'};
# Getendnode([$calldest]);  #%functionname
dprint $callsrc,$calldest,"\n";
push(@{$expandnodelist[$calldest]->{'src'}},$callsrc);
dprint @{$expandnodelist[$calldest]->{'src'}},"\n";
    }

    #return の存在するノードを関数ごとにまとめる。Getendnode() でやってます。


    #return->src に枝を引く。
    foreach(0..$#expandnodelist){
next if(!defined($expandnodelist[$_]));
foreach $returnsrc(@{$expandnodelist[$_]->{'return'}}){
    foreach $returndest(@{$expandnodelist[$_]->{'src'}}){
push(@{$nodelist[$returnsrc]->{'param-out'}},$returndest);
    }
}
    }
}
```

```
############################################################
#debug
sub printexpandstatus
#
{
    foreach(0..$#callnodes){
next if (!defined $callnodes[$_]);
print "$_:$callnodes[$_]->{'node'}<call>$callnodes[$_]->{'call'}";
print "<a-in>",@{$callnodes[$_]->{'a-in'}},"<a-out>",@{$callnodes[$_]->{'a-out'}},"\n";
    }

    foreach(0..$#expandnodelist){
next if (!defined $expandnodelist[$_]);
print "$_:<f-in>",@{$expandnodelist[$_]->{'f-in'}},"<f-out>",@{$expandnodelist[$_]->{'f-out'}},"<src>",@{$expandnodelist[$_]->{'src'}},"\n";
    }
}

1;
```

B.1.9 Prepost Package

```
#!/usr/local/bin/perl
# $Id: call.pl,v 1.0 2016/02/17 19:09 wangbo Exp $
################## ---precondition and postcondition dependence ---
#    #    printPostDependence();
#    #    printPreDependence();
#    #    mkPreDependence();
#    #    mkPostDependence();
########

sub printPreDependence # (void)
{
    for ($n = 1; $n <= $#nodelist; $n++) {
next if (!defined $nodelist[$n]);

print "$n: <line> $nodelist[$n]->{'lin'}";

if (@{$predependence[$n]}) {
    print " <pre> @{$prependence[$n]}";
}
print "\n";
    }
}

sub printPostDependence # (void)
{
    for ($n = 1; $n <= $#nodelist; $n++) {
next if (!defined $nodelist[$n]);

print "$n: <line> $nodelist[$n]->{'lin'}";

if (@{$postdependence[$n]}) {
    print " <post> @{$postdependence[$n]}";
}
print "\n";
    }
}

sub mkPreDependence # (void)
{
sub mkPreDependenceрecusive #($preconditionnode,$currentprocessnode)
{
my $prenode, $currentnode, $preprocessnode;
$prenode = $_[0];
$currentnode = $_[1];
print "$prenode ,$currentnode\n";
if (defined currentnode->{'return'}[0]) {
push(@{$predependence[$prenode]}, $currentnode);
}
else {
foreach $preprocessnode(@{$nodelist[$currentnode]->{'con'}}) {
push(@{$predependence[$preprocessnode]}, $prenode);
mkPreDependenceрecusive($prenode, $preprocessnode);
}
foreach $preprocessnode(@{$nodelist[$currentnode]->{'pre'}}) {
push(@{$predependence[$preprocessnode]}, $prenode);
mkPreDependenceрecusive($prenode, $preprocessnode);
}
}
}
    my $n;

    for ($n = 1; $n <= $#nodelist; $n++) {
next if (!defined $nodelist[$n]);
if (defined $nodelist[$n]->{'pre'}[0]) {
print "$nodelist[$n]->{'lin'}\n";
mkPreDependenceрecusive($n,$n);
     }
}
}
sub mkPostDependence #(void)
{
my $n, $postprocessnode;
for ($n = 1; $n <= $#nodelist; $n++) {
next if (!defined $nodelist[$n]);
if (defined $nodelist[$n]->{'post'}[0]) {
foreach $postprocessnode(@{$nodelist[$n]->{'post'}}) {
push(@{$postdependence[$postprocessnode]}, $n);
}
     }
}
}

1;
```

Appendix C

A Tasking Deadlock Detector for Ada 2012 Programs

C.1 Source Transformation Tool

C.1.1 Asis_Utils Package

```
with Asis;

package Asis_Utils is

   type Mode_Kinds_Array is array(Integer range <>) of Asis.Mode_Kinds;

   --  gain ASIS.Expression
   function Get_Called_Name(Elem : Asis.Statement) return Asis.Expression;
   function Corresponding_Called_Function(Expression : in Asis.Expression)
  return Asis.Declaration;
   function Is_Variable(Element : Asis.Expression) return Boolean;
   function Variable_Name(Element : Asis.Expression) return String;
   function Get_Identifier(Element : Asis.Expression) return Asis.Expression;
   function Get_Mode(Element : Asis.Association) return Asis.Mode_Kinds;

   function Get_Mode_Array(Element : Asis.Declaration)
   return Mode_Kinds_Array;
   function Var_Type_Name(Element : Asis.Expression) return String;
   function Exp_Name(Element : Asis.Expression) return String;
   function Unique_Identifier return String;

   function Au_Corresponding_Name_Definition(Reference : in Asis.Expression)
     return Asis.Defining_Name;

   function Au_Corresponding_Name_Declaration(Reference : in Asis.Expression)
     return Asis.Defining_Name;
   function Declaration_Full_Name(Decl : Asis.Declaration) return String;

private

   Identlast : Integer := 0;

end Asis_Utils;


with Asis;
with Asis.Definitions;
with Asis.Declarations;
with Asis.Statements;
with Asis.Elements;
with Asis.Expressions;
with Asis.Exceptions;
with Asis.Text;
with Ada.Text_IO; use Ada.Text_IO;
with String_Handler; use String_Handler;
with Ada.Characters.Handling; use Ada.Characters.Handling;
with V_Strings; use V_Strings;

package body Asis_Utils is

   use Asis;
   -- gain ASIS.Expression
   function Get_Called_Name(Elem : Statement) return Expression is
   begin
      case Asis.Elements.Element_Kind(Elem) is
 when An_Expression =>
    if Asis.Elements.Expression_Kind(Elem) = A_Function_Call then
       return Asis.Expressions.Prefix(Elem);
    else
       Put_Line("not handled by get_called_name...");
       raise Program_Error;
       return Nil_Element;
    end if;
 when A_Statement =>
    case Asis.Elements.Statement_Kind(Elem) is
       when An_Entry_Call_Statement |
 A_Procedure_Call_Statement =>
```

```
   return Asis.Statements.Called_Name(Elem);
        when A_Timed_Entry_Call_Statement =>
   declare
      Plist : Asis.Path_List
        := Asis.Statements.Statement_Paths(Elem);
      Slist : Asis.Statement_List
        := Asis.Statements.Sequence_Of_Statements(Plist(1));
   begin

      --       Put_Line(To_String(Asis.Elements.Debug_Image(First_Element(Slist))));
      return Asis.Statements.Called_Name(Slist(1));
   end ;
        when An_Accept_Statement =>
   return Asis.Statements.Accept_Entry_Direct_Name(Elem);
        when An_Abort_Statement =>

   return Asis.Statements.Aborted_Tasks(Elem)(1);
        when A_Conditional_Entry_Call_Statement =>
   return Asis.Statements.Called_Name
     (Asis.Statements.Sequence_Of_Statements
      (Asis.Statements.Statement_Paths(Elem)(1))(1));
--         when A_Selective_Accept_Statement =>
--    return Nil_Element;
        when others =>
   Put_Line("not handled by get_called_name...");
   raise Program_Error;
   return Nil_Element;
     end case;
 when others =>
     Put_Line("not handled by get_called_name...");
     raise Program_Error;
     return Nil_Element;
       end case;
   end Get_Called_Name;


    function Corresponding_Called_Function(Expression : in Asis.Expression)
   return Asis.Declaration is
       Function_Name : Asis.Expression := Asis.Expressions.Prefix(Expression);
    begin
       case Asis.Elements.Expression_Kind(Function_Name) is
 when A_Selected_Component =>
     return Asis.Elements.Enclosing_Element(Asis.Expressions.Corresponding_Name_Definition(Asis.Expressions.Selector(Function_Name)));
 when others =>
     null;
       end case;
       return Asis.Elements.Enclosing_Element(Asis.Expressions.Corresponding_Name_Definition(Function_Name));
    end Corresponding_Called_Function;

    function Get_Called_Entity(Elem : Element) return Declaration is
    begin
       case Asis.Elements.Element_Kind(Elem) is
 when An_Expression =>
     case Asis.Elements.Expression_Kind(Elem) is
        when A_Function_Call =>
   return Corresponding_Called_Function(Elem);
        when others =>
   Put_Line("Get_Called_Entity : inappropriate element.");
   raise Program_Error;
     end case;
 when A_Statement =>
     case Asis.Elements.Statement_Kind(Elem) is
        when A_Procedure_Call_Statement =>
   return Asis.Statements.Corresponding_Called_Entity(Elem);
        when others =>
   Put_Line("Get_Called_Entity : inappropriate element.");
   raise Program_Error;
     end case;
 when others =>
     Put_Line("Get_Called_Entity : inappropriate element.");
     raise Program_Error;
       end case;
       return Nil_Element;
    end Get_Called_Entity;


    function Is_Variable(Element : Asis.Expression) return Boolean is
       Decl : Asis.Declaration := Au_Corresponding_Name_Declaration(Element);

    begin
       if not Asis.Elements.Is_Nil(Decl) and then
Asis.Elements.Declaration_Kind(Decl) = A_Variable_Declaration then
 return True;
       else
 return False;
       end if;
    end Is_Variable;


    function Variable_Name(Element : Asis.Expression) return String is
       Tmp_Element : Asis.Element := Element;
       Next_Element : Asis.Element;
    begin
       loop
 Next_Element := Asis.Elements.Enclosing_Element(Tmp_Element);
 if Asis.Elements.Is_Nil(Next_Element) or else
   Asis.Elements.Element_Kind(Next_Element) /= An_Expression or else
   (Asis.Elements.Expression_Kind(Next_Element) /= A_Selected_Component
    and Asis.Elements.Expression_Kind(Next_Element) /= An_Identifier)
 then
    return Eliminate_Space(To_String(Asis.Text.Element_Image(Tmp_Element)));
 else
    Tmp_Element := Next_Element;
 end if;
       end loop;
    end Variable_Name;


    function Get_Identifier(Element : Asis.Expression) return Asis.Expression is
```

```
      Now : Asis.Expression := Element;
   begin
      loop
 case Asis.Elements.Expression_Kind(Now) is
    when An_Identifier =>
       return Now;
    when A_Selected_Component =>
       Now := Asis.Expressions.Selector(Now);
    when An_Indexed_Component =>
       Now := Asis.Expressions.Prefix(Now);
    when others =>
       return Nil_Element;
 end case;
      end loop;
   end Get_Identifier;


   function Get_Mode(Element : Asis.Association) return Asis.Mode_Kinds is
      F_Param : Asis.Element := Asis.Expressions.Formal_Parameter(Element);
      P_Spec : Asis.Parameter_Specification;
   begin
      if Asis.Expressions.Is_Normalized(Element) then
 P_Spec := Asis.Elements.Enclosing_Element(F_Param);
 return Asis.Elements.Mode_Kind(P_Spec);
      else
 Put_Line("unnormalized association.");
 raise Program_Error;
      end if;
      return Not_A_Mode;
   end Get_Mode;



   function Get_Mode_Array(Element : Asis.Declaration)
   return Mode_Kinds_Array is
      Count : Integer := 0;
      Now : Integer := 1;
   begin
      if not Asis.Elements.Is_Nil(Element) then
 declare
    Params : Asis.Parameter_Specification_List
      := Asis.Declarations.Parameter_Profile(Element);
 begin


    for I in Params'RANGE loop
       declare
  N_Tmp : Defining_Name_List := Asis.Declarations.Names(Params(I));
       begin
  Count := Count + N_Tmp'LENGTH ;
       end;
    end loop;

    declare
       M_Kinds : Mode_Kinds_Array(1..Count);
    begin
       for I in Params'RANGE loop
 declare
    N_Tmp : Defining_Name_List := Asis.Declarations.Names(Params(I));
 begin
    for J in N_Tmp'RANGE loop
M_Kinds(Now) := Asis.Elements.Mode_Kind(Params(I));
Now := Now + 1;
     end loop;
  end;
       end loop;
       return M_Kinds;
    end;
 end;
      else
 declare
    M_Kinds : Mode_Kinds_Array(1..0);
 begin
    return M_Kinds;
 end;
      end if;

   end Get_Mode_Array;


   function Var_Type_Name(Element : Asis.Expression) return String is
      Decl : Asis.Declaration
:= Au_Corresponding_Name_Declaration(Element);
   begin
      if not Asis.Elements.Is_Nil(Decl) then
 return Eliminate_Space
   (To_String
    (Asis.Text.Element_Image
     (Asis.Definitions.Subtype_Mark
      (Asis.Declarations.Object_Declaration_View(Decl)))));
      end if;
      return To_S(Null_Str);
   end Var_Type_Name;


   function Exp_Name(Element : Asis.Expression) return String is
      Result : V_String := Null_Str;
      Iter : Asis.Expression := Element;
   begin
      loop
 case Asis.Elements.Expression_Kind(Iter) is
   when An_Identifier =>
      Result := To_String(Asis.Expressions.Name_Image(Iter))
& Result;
      return To_S(Result);
    when A_Selected_Component =>
       Result := Result &
 To_String(Asis.Expressions.Name_Image
   (Asis.Expressions.Selector(Iter)));
```

```
      Iter := Asis.Expressions.Prefix(Iter);
    when others =>
       return To_S(Null_Str);
 end case;
      end loop;
      return To_S(Null_Str);
   end Exp_Name;



   function Unique_Identifier return String is
      package Int_IO is new Ada.Text_IO.Integer_IO(Integer);
      use Int_IO;
      Result : String(1..5);
   begin
      Put(Result, IdentLast);
      IdentLast := IdentLast + 1;
      for I in Result'Range loop
         if Result(I) = ' ' then
            Result(I) := '0';
         end if;
      end loop;
      return "DECL_" & Result;
   end Unique_Identifier;


   function Au_Corresponding_Name_Definition(Reference : in Asis.Expression)
     return Asis.Defining_Name is
   begin
      return Asis.Expressions.Corresponding_Name_Definition(Reference);
   exception
      when Asis.Exceptions.Asis_Failed =>
 return Nil_Element;
   end Au_Corresponding_Name_Definition;


   function Au_Corresponding_Name_Declaration(Reference : in Asis.Expression)
     return Asis.Defining_Name is
   begin
      return Asis.Expressions.Corresponding_Name_Declaration(Reference);
   exception
      when Asis.Exceptions.Asis_Failed =>
 return Nil_Element;
   end Au_Corresponding_Name_Declaration;

   function Declaration_Full_Name(Decl : Asis.Declaration) return String is
      Result : V_String := Null_Str;
      Elem : Asis.Element := Decl;
   begin
      Result := To_V(Asis.Declarations.Defining_Name_Image
     (Asis.Declarations.Names(Decl)(1)));
      loop
 Elem := Asis.Elements.Enclosing_Element(Elem);
 exit when Asis.Elements.Is_Nil(Elem);
 if Asis.Elements.Element_Kind(Elem) = A_Declaration then
    Result := To_V(Asis.Declarations.Defining_Name_Image
     (Asis.Declarations.Names(Elem)(1)) & "." ) & Result;
 end if;
      end loop;
      return To_S(Result);
   end Declaration_Full_Name;

end Asis_Utils;
```

C.1.2 Call_Analyzer Package

```
with Id_List;
with Asis;
with V_Strings;
with Name_Handler; use Name_Handler;

package Call_Analyzer is

   type Info_Source is new Boolean;

--   procedure Find_Calls(Elem : Asis.Element; Current_Block : V_String);
   procedure Find_Calls(Elem : Asis.Element);

   package Bool_Id_List is new Id_List(Boolean, False);

   Is_In_Package : Bool_Id_List.A_Node := null;

--private

--   Local_Current_Unit : Unit_Item_Link;

end Call_Analyzer;


with Ada.Characters.Handling;
use Ada.Characters.Handling;
with Asis;
with Asis.Declarations;
with Asis.Definitions;
with Asis.Elements;
with Asis.Expressions;
with Asis.Statements;
with Gela_Ids;
with Asis.Iterator;
with Asis.Text;
with Name_Handler;
use Name_Handler;
with V_Strings; use V_Strings;
with Global_Types; use Global_Types;
with String_Handler; use String_Handler;
with Ada.Text_IO;
```

```
use Ada.Text_IO;
with Asis_Utils; use Asis_Utils;

package body Call_Analyzer is

   use ASIS;
   Top_Element : Asis.Element;

   function Is_Here (Element : Asis.Element) return Boolean is
   begin
      return (Asis.Elements.Element_Kind (Element) /= Not_An_Element);
   end Is_Here;

   function First_Element (List : Asis.Element_List) return Asis.Element is
   begin
      return (List (List'First));
   end First_Element;


   function Is_Protected(Element : Asis.Declaration) return Boolean is
      Enc_Element : Asis.Element := Asis.Elements.Enclosing_Element(Element);
   begin
      if Is_Here(Enc_Element) then
 Enc_Element := Asis.Elements.Enclosing_Element(Enc_Element);
      else
 return False;
      end if;
--      Put_Line(To_String(Asis.Elements.Debug_Image(Enc_Element)));
      if Is_Here(Enc_Element) and then Asis.Elements.Element_Kind(Enc_Element)
--  = A_Definition and then
--  Asis.Elements.Definition_Kind(Element)
--  = A_Protected_Definition then
= A_Declaration and then
(Asis.Elements.Declaration_Kind(Enc_Element)
 = A_Protected_Type_Declaration or
 Asis.Elements.Declaration_Kind(Enc_Element)
 = A_Single_Protected_Declaration) then
 return True;
      end if;
      return False;
   end Is_Protected;


   function Is_Child(Parent : Gela_Ids.Id; Child : Asis.Declaration) return Boolean is
      Enc_Element : Asis.Element := Asis.Elements.Enclosing_Element(Child);
   begin
      for I in 1..2 loop
 if Is_Here(Enc_Element) then
    -- ???
    if Gela_Ids.Is_Equal(Gela_Ids.Create_ID(Enc_Element), Parent) then
       return True;
    end if;
    -- ???
    Enc_Element := Asis.Elements.Enclosing_Element(Enc_Element);
 else
    return False;
 end if;
      end loop;
      if Is_Here(Enc_Element) and then
Gela_Ids.Is_Equal(Gela_Ids.Create_ID(Enc_Element), Parent) then
 return True;
      end if;
      return False;
   end Is_Child;


   function Is_Task_Type(Child : Asis.Declaration) return Boolean is
      Enc_Element : Asis.Element := Asis.Elements.Enclosing_Element(Child);
   begin
      if Is_Here(Enc_Element) then
 Enc_Element := Asis.Elements.Enclosing_Element(Enc_Element);
      else
 return False;
      end if;
      if Is_Here(Enc_Element) and then
Asis.Elements.Element_Kind(Enc_Element) = A_Declaration and then
Asis.Elements.Declaration_Kind(Enc_Element) = A_Task_Type_Declaration
then
   return True;
      else
 return False;
      end if;
   end Is_Task_Type;

   function Is_Access(Decl : in Asis.Declaration) return Boolean is

      Name_Def : Asis.Defining_Name
:= Asis.Expressions.Corresponding_Name_Definition(
  Asis.Definitions.Subtype_Mark(
    Asis.Declarations.Object_Declaration_View(Decl)));
      Type_Def : Asis.Definition;
      use Asis;
   begin
--       if Name_Def = Nil_Element then
      if Asis.Elements.Is_Part_Of_Implicit(Name_Def) then
 return False;
      end if;
      Type_Def := Asis.Elements.Enclosing_Element(Name_Def);
--      Put_Line(Asis.Text.Element_Image(Type_Def));
--      Put_Line(Asis.Elements.Debug_Image(Asis.Declarations.Type_Declaration_View(Type_Def)));
      if Asis.Elements.Type_Kind(Asis.Declarations.Type_Declaration_View(Type_Def)) = An_Access_Type_Definition then
-- Put_Line("access!");
 return True;
      end if;
      return False;
   end Is_Access;

   function Is_Dynamic_Binding(Exp : Asis.Expression) return Boolean is
      Now_Elem : Asis.Element;
```

```
   begin
      if Asis.Elements.Expression_Kind(Exp) = An_Indexed_Component then
 return True;
      end if;
      Now_Elem := Asis.Expressions.Corresponding_Name_Declaration(Exp);
      if Asis.Elements.Is_Nil(Now_Elem) then
 return False;
      end if;
      return False;
--      return Is_Access(Now_Elem);
   end Is_Dynamic_Binding;


   procedure Pre_Source
(Element : in     Asis.Element;
 Control : in out Asis.Traverse_Control;
 State   : in out Info_Source) is
   begin

      case Asis.Elements.Element_Kind(Element) is
 when A_Declaration =>
    case Asis.Elements.Declaration_Kind(Element) is
       when A_Task_Body_Declaration =>
  Enter_Unit_Body
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Task_Body_Declaration, TASKS);
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);

       when A_Single_Task_Declaration =>
  Enter_Unit_Spec
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Single_Task_Declaration, TASKS);
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when A_Task_Type_Declaration =>
  Enter_Unit_Spec
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Task_Type_Declaration, TASKS);
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when A_Protected_Body_Declaration =>
  Enter_Unit_Body
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Protected_Body_Declaration, PROTECTED_OBJECTS);
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when A_Protected_Type_Declaration =>
  Enter_Unit_Spec
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Protected_Type_Declaration,
     PROTECTED_OBJECTS);
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when A_Single_Protected_Declaration =>
  Enter_Unit_Spec
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Single_Protected_Declaration, PROTECTED_OBJECTS);
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when A_Procedure_Declaration =>
  declare
     This_Unit_Class : Unit_Class;
  begin
     if Is_Protected(Element) then
This_Unit_Class := PROTECTED_PROCEDURES;
     else
This_Unit_Class := PROCEDURES;
     end if;
     Enter_Unit_Spec
       (To_V(Asis.Declarations.Defining_Name_Image
     (First_Element (Asis.Declarations.Names (Element)))),
Element,
A_Procedure_Body_Declaration, This_Unit_Class);
  end;
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when A_Procedure_Body_Declaration =>
  declare
     This_Unit_Class : Unit_Class;
  begin
     if Current_Unit.This_Unit_Class = MAIN_TASK then
This_Unit_Class := MAIN_PROCEDURE;
     elsif Is_Protected(Element) then
This_Unit_Class := PROTECTED_PROCEDURES;
     else
This_Unit_Class := PROCEDURES;
     end if;
     Enter_Unit_Body
       (To_V(Asis.Declarations.Defining_Name_Image
     (First_Element (Asis.Declarations.Names (Element)))),
Element,
A_Procedure_Body_Declaration, This_Unit_Class);
  end;
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when A_Function_Declaration =>
```

```
  declare
     This_Unit_Class : Unit_Class;
  begin
     if Is_Protected(Element) then
This_Unit_Class := PROTECTED_FUNCTIONS;
     else
This_Unit_Class := FUNCTIONS;
     end if;
  Enter_Unit_Spec
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Procedure_Body_Declaration, This_Unit_Class);
  end;
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when A_Function_Body_Declaration =>
  declare
     This_Unit_Class : Unit_Class;
  begin
     if Is_Protected(Element) then
This_Unit_Class := PROTECTED_FUNCTIONS;
     else
This_Unit_Class := FUNCTIONS;
     end if;
  Enter_Unit_Body
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Procedure_Body_Declaration, This_Unit_Class);
  end;
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when An_Entry_Declaration =>
  Enter_Unit_Spec
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     An_Entry_Body_Declaration, PROTECTED_ENTRY_BODIES);
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);
       when An_Entry_Body_Declaration =>
  Enter_Unit_Body
    (To_V(Asis.Declarations.Defining_Name_Image
  (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     An_Entry_Body_Declaration, PROTECTED_ENTRY_BODIES);
  Bool_Id_List.Set_Node(Is_In_Package,
Gela_Ids.Create_ID(Element), True);

       when A_Package_Declaration | A_Package_Body_Declaration =>
  if not Asis.Elements.Is_Equal(Element, Top_Element) then
     Control := Abandon_Children;
  end if;
       when others =>
  null;
    end case;


 when A_Statement =>
    case Asis.Elements.Statement_Kind(Element) is
       when A_Block_Statement =>
  if Is_Here (Asis.Statements.Statement_Identifier (Element))
  then
     Enter_Unit_Spec(To_V(Asis.Declarations.Defining_Name_Image
  (Asis.Statements.Statement_Identifier (Element))),
     Element,
     Not_A_Declaration, BLOCKS);
  else
     Enter_Unit_Spec(To_V(Unique_Identifier),
     Element,
     Not_A_Declaration, BLOCKS);
  end if;
       when A_Procedure_Call_Statement =>
  declare
     Dest : Asis.Declaration
       := Asis.Statements.Corresponding_Called_Entity(Element);
  begin
     if not Asis.Elements.Is_Nil(Dest) then
Call_State_List.Set_Node
  (Current_Unit.Is_Calling_List,
   Gela_Ids.Create_ID(Dest),
   (UNCHECKED, Null_Str, False, False, Gela_Ids.Nil_Id));
     end if;
  end;
       when An_Entry_Call_Statement =>
  declare

     Dest : Asis.Declaration
       := Asis.Statements.Corresponding_Called_Entity(Element);

     Dest_For_Tt : Asis.Declaration;
     A_Call_State : Call_State;
     Task_Name : V_String := Null_Str;
     Call_Name : Asis.Expression;
     Has_CDT : Boolean := True;
     Has_CDT_For_TT : Boolean := False;
     Var_Decl : Gela_Ids.Id := Gela_Ids.Nil_Id;
  begin
     if not Asis.Elements.Is_Nil(Dest)
       and then not Is_Protected(Dest) then
if Is_Child(Current_Unit.Element_ID, Dest) then
   A_Call_State.State := DECLARED;
else
   A_Call_State.State := CHECKED;
end if;

Call_Name := Asis.Statements.Called_Name(Element);
if Asis.Elements.Expression_Kind(Call_Name)
```

```
  = An_Indexed_Component then
   Call_Name := Asis.Expressions.Prefix(Call_Name);
end if;
if Asis.Elements.Expression_Kind(Call_Name)
  = A_Selected_Component then
   Task_Name := To_V
     (Eliminate_Space
      (To_String
       (Asis.Text.Element_Image
(Asis.Expressions.Prefix
 (Call_Name)))));

   Dest_For_Tt := Asis.Expressions.Corresponding_Name_Declaration(Asis_Utils.Get_Identifier(Asis.Expressions.Prefix(Call_Name)));
--     Put_Line("== ca ==");
--      Put_Line(To_String(Asis.Text.Element_Image(Dest_For_Tt)));
--      Put_Line("current_unit : " & To_S(Current_Unit.Name));
   if not Asis.Elements.Is_Nil(Dest_For_Tt) and then
     Is_Child(Current_Unit.Element_ID, Dest_For_Tt)
   then
--      Put_Line("declared!");
      A_Call_State.State := DECLARED;
   end if;

   if Is_Dynamic_Binding
     (Asis.Expressions.Prefix(Call_Name)) then
      Has_CDT := False;
      Has_CDT_For_TT := True;
   elsif Is_Task_Type(Dest) then
      Var_Decl := Gela_Ids.Create_ID(Asis.Expressions.Corresponding_Name_Declaration(Get_Identifier(Asis.Expressions.Prefix(Call_Name))));
   end if;
--else
end if;
--Put_Line(To_S(Task_Name) & " is called in " & To_S(Current_Unit.Name));
Call_State_List.Set_Node
  (Current_Unit.Is_Calling_List,
   Gela_Ids.Create_ID(Dest), (A_Call_State.State, Task_Name, Has_CDT, Has_CDT_For_TT, Var_Decl));
     end if;
  end;

       when others =>
  null;
    end case;
 when others =>
    null;
      end case;
   exception
      when others =>
 Put_Line("exception raised at " & To_String(Asis.Text.Element_Image(Element)));
 raise;
   end Pre_Source;



   procedure Post_Source
     (Element : in     Asis.Element;
      Control : in out Asis.Traverse_Control;
      State   : in out Info_Source) is
   begin
      case Asis.Elements.Element_Kind(Element) is
 when A_Declaration =>
    case Asis.Elements.Declaration_Kind(Element) is
       when A_Task_Body_Declaration |
 A_Single_Task_Declaration |
 A_Task_Type_Declaration |
 A_Protected_Body_Declaration |
 A_Protected_Type_Declaration |
 A_Single_Protected_Declaration |
 A_Procedure_Declaration |
 A_Procedure_Body_Declaration |
 A_Function_Declaration |
 A_Function_Body_Declaration |
 An_Entry_Body_Declaration |
 An_Entry_Declaration =>
  Exit_Unit;
       when others =>
  null;
    end case;
 when A_Statement =>
    case Asis.Elements.Statement_Kind(Element) is
       when A_Block_Statement =>
  Exit_Unit;
       when others =>
  null;
    end case;
 when others =>
    null;
      end case;
   end Post_Source;

   procedure Traverse_Call is new Asis.Iterator.Traverse_Element
     (Info_Source, Pre_Source, Post_Source);

   procedure Revise_Call is


      function Indirect_Calls(Iter : Unit_Item_Link)
      return Call_State_List.A_Node is
 A_Call : Call_State_List.A_Node := Iter.Is_Calling_List;
 Tmp_State_List, A_State_List : Call_State_List.A_Node := null;
 use Call_State_List;
      begin
 if Iter = null then
    return null;
 end if;
 while A_Call /= null loop
    if A_Call.Object.State = UNCHECKED and then
      (Call_State_List.Search_Node(Tmp_State_List, A_Call.Index)
      = Null_Call_State or else
      Call_State_List.Search_Node(Tmp_State_List, A_Call.Index).State
```

```
      /= CHECKED) then

       if Bool_Id_List.Search_Node(Is_In_Package, A_Call.Index) then
  -- ???
  A_Call.Object.State := Checked ;
  -- ???
  A_State_List := Indirect_Calls
    (Unit_Item_List.Search_Node
     (Unit_Items,
      A_Call.Index));
  while A_State_List /= null loop
     Call_State_List.Set_Node(Tmp_State_List, A_State_List.Index,
      A_State_List.Object);
     A_State_List := A_State_List.Next;
  end loop;
       else

  A_Call.Object := (OUTOFUNIT, Null_Str, False, False, Gela_Ids.Nil_Id);
       end if;


    end if;
    A_Call := A_Call.Next;
 end loop;

 while Tmp_State_List /= null loop
    Call_State_List.Set_Node(Iter.Is_Calling_List,
     Tmp_State_List.Index,
     Tmp_State_List.Object);
    Tmp_State_List := Tmp_State_List.Next;
 end loop;

 return Iter.Is_Calling_List;
      end Indirect_Calls;

      procedure Unit_Iter(Iter : Unit_Item_Link) is
-- A_Call : Call_State_List.A_Node := Iter.Is_Calling_List;
 Dummy : Call_State_List.A_Node := null;
 Dummy2 : Unit_Item_Link;
      begin
 if Iter /= null then

--   if Iter.Name /= Null_Str then
--      Put_Line(To_S(Iter.Name));
--   else
--      Put_Line("null!");
--   end if;


    Dummy := Indirect_Calls(Iter);
    Dummy2 := Iter.Brother;

    Unit_Iter(Iter.Brother);

    Unit_Iter(Iter.Children);
 end if;
      end Unit_Iter;

   begin
      Unit_Iter(Current_Unit);
   end Revise_Call;


--   procedure Find_Calls(Elem : Asis.Element; Current_Block : V_String) is
   procedure Find_Calls(Elem : Asis.Element) is
      The_Control : Asis.Traverse_Control := Asis.Continue ;
      The_Spec_Information : Info_Source := True;
   begin
      Top_Element := Elem;
      Is_In_Package := null;
      Traverse_Call(Elem, The_Control, The_Spec_Information);
      Revise_Call;
   end Find_Calls;

end Call_Analyzer;
```

C.1.3 Comp_Measure Package

```
with Ada;
with Ada.Command_Line;
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Characters.Handling; use Ada.Characters.Handling;
with Ada.Exceptions;


with Source_Trav; use Source_Trav;
with Spec_Reader; --use Spec_Reader;

with Asis;
with Asis.Iterator;
with Asis.Elements ;
with Asis.Exceptions ;
with Asis.Compilation_Units ;
with Asis.Ada_Environments ;
with Asis.Implementation ;
with V_Strings; use V_Strings;


procedure Comp_Measure is

   procedure Traverse_Source is new Asis.Iterator.Traverse_Element
     (Info_Source, Pre_Source, Post_Source) ;

   -- Some global variables.
   --   The_DS_Context : Asis.Context;
   The_Unit : Asis.Compilation_Unit;
```

```
   The_Spec : Asis.Compilation_Unit;
   The_Declaration : Asis.Declaration ;
   The_Control : Asis.Traverse_Control := Asis.Continue ;
   The_Source_Information : Info_Source ;
   The_Spec_Information : Spec_Reader.Info_Source ;

   -- Measurement Compositter Main
begin

   if Ada.Command_Line.Argument_Count not in 1..2 then
      Put_Line("Usage : " & Ada.Command_Line.Command_Name & " Unit [Spec]");
      return;
   end if;

   Spec_Reader.Has_Spec := (Ada.Command_Line.Argument_Count = 2);

   -- Initialization of Asis environment.
   Asis.Implementation.Initialize;
   Asis.Ada_Environments.Associate
     (The_Context => The_DS_Context,
      Name        => "The_DS_Context",
      Parameters  => "-FS");
   Asis.Ada_Environments.Open ( The_DS_Context );
   declare
      Unite : String := Ada.Command_Line.Argument(1);
   begin
      if Spec_Reader.Has_Spec then
         declare
            Spec : String := Ada.Command_Line.Argument(2);
         begin
            The_Spec := Asis.Compilation_Units.Library_Unit_Declaration
              (To_Wide_String(Spec),
               The_DS_Context);

            if ( Asis.Compilation_Units.Is_Nil ( The_Spec ) ) then
               Put_Line ( "Unit " & Spec & " is Nil...");
               raise Asis.Exceptions.Asis_Inappropriate_Compilation_Unit ;
            end if ;

            The_Declaration := Asis.Elements.Unit_Declaration(The_Spec) ;

            Spec_Reader.Traverse_Spec(The_Declaration, The_Control, The_Spec_Information);

            Spec_Name := To_V(Spec);


            if Spec_Reader.Spec_Name = Null_Str then
               Put_Line("The spec is not a package.");
               raise Program_Error;
            end if;
         end;
      end if;
      The_Unit := Asis.Compilation_Units.Compilation_Unit_Body(To_Wide_String(Unite),
                                                               The_DS_Context);
      -- If it's null, continuing makes no sense ...
      if ( Asis.Compilation_Units.Is_Nil ( The_Unit ) ) then
         Put_Line ( "Unit " & Unite & " is Nil...");
         --raise Asis.Exceptions.Asis_Inappropriate_Compilation_Unit ;
      end if ;

      --      The_Declaration := Asis.Elements.Unit_Declaration(The_unit) ;
      --     Put_Line(To_String(Asis.Elements.Debug_Image(The_Declaration)));
      declare
         Clause_List : Asis.Context_Clause_List :=
           Asis.Elements.Context_Clause_Elements ( The_Unit, true ) ;
      begin
         for Each_Clause in Clause_List'Range loop
            Traverse_Source(Clause_List(Each_Clause),
                            The_Control,
                            The_Source_Information);
         end loop ;
      end ;
      Initiate_Source( The_Unit, Unite, The_Control, The_Source_Information ) ;
   end ;
   The_Declaration := Asis.Elements.Unit_Declaration(The_Unit) ;
   Make_CDT_Of_Top(The_Declaration);

   Is_Making_Clause := True;
   Traverse_Source(The_Declaration,
                   The_Control,
                   The_Source_Information);
   Terminate_Source( The_Control, The_Source_Information );

   -- Closing Asis ....
   Asis.Ada_Environments.Close ( The_DS_Context );
   Asis.Ada_Environments.Dissociate ( The_DS_Context );
   Asis.Implementation.Finalize ("");

exception

   when Asis.Exceptions.Asis_Inappropriate_Compilation_Unit =>
      Put_Line ( "The file " & Ada.Command_Line.Argument (1) &
                 " does not contain any Ada Unit.");
      New_Line ;
      Put_Line("USAGE: " & Ada.Command_Line.Command_Name & " [-n|-s] Unit[.ads|.adb]");
      Put_Line("     : " & Ada.Command_Line.Command_Name & " -h") ;
      raise ;

   when Asis.Exceptions.Asis_Failed |
        Asis.Exceptions.Asis_Inappropriate_Element |
        Asis.Exceptions.Asis_Inappropriate_Context =>
      Put_Line (Ada.Characters.Handling.To_String
                (Asis.Implementation.Diagnosis));   --  ???
      raise ;

   when The_Error : others =>
      Put_Line ( "The exception received : " & Ada.Exceptions.Exception_Name ( The_Error ) ) ;
      Put_Line (Ada.Characters.Handling.To_String
                (Asis.Implementation.Diagnosis));
```

```
      raise ;

end Comp_Measure;
```

C.1.4 Designations Package

```
with Gnat.Regexp, V_Strings;
use Gnat.Regexp, V_Strings;


package Designations is

   type V_String_Link is access V_String;
   type Regexp_Link is access Regexp;



   type Designation is
      record

 Name : V_String_Link := null;

 Pattern : Regexp_Link := null;

 IsMatchCondition : Boolean;
      end record;

   Null_Designation : constant Designation := (null, null, True);

   procedure Set_Designation(Rec : in out Designation; P_Str : in String);
   function Match_Designation(S_Pattern : Designation; Str : V_String)
      return Boolean;

end Designations;


with Gnat.Regexp, V_Strings;
use Gnat.Regexp, V_Strings;

package body Designations is

   procedure Set_Designation(Rec : in out Designation; P_Str : in String)
   is
   begin
      if P_Str'LENGTH > 0 and then
P_Str(P_Str'FIRST) = '#' then
 if P_Str'LENGTH = 1 then
    Rec.Name := new V_String;
    Rec.Name.all := Null_Str;
 else
    Rec.Pattern := new Regexp;
    Rec.Pattern.all := Compile(P_Str(P_Str'FIRST+1 .. P_Str'LAST),
       False,
       False);
    Rec.IsMatchCondition := True;
 end if;
      elsif P_Str'LENGTH > 1 and then
P_Str(P_Str'FIRST) = '~' and then
P_Str(P_Str'FIRST + 1) = '#' then
 if P_Str'LENGTH = 2 then
    Rec.Name := new V_String;
    Rec.Name.all := Null_Str;
 else
    Rec.Pattern := new Regexp;
    Rec.Pattern.all := Compile(P_Str(P_Str'FIRST+2 .. P_Str'LAST),
       False,
       False);
    Rec.IsMatchCondition := False;
 end if;
      else
 Rec.Name := new V_String;
 Rec.Name.all := To_V(P_Str);
      end if;
   end Set_Designation;

   function Match_Designation(S_Pattern : Designation; Str : V_String)
      return Boolean is
   begin
      if S_Pattern.Name /= null then
-- return Equal_Insensitive(S_Pattern.Name.all, Str);
 return Match_Wildcard(S_Pattern.Name.all, Str);
      elsif S_Pattern.Pattern /= null then
 declare
    Tmp_Str : String := To_S(Str);
 begin
    if S_Pattern.IsMatchCondition then
       return Match(Tmp_Str, S_Pattern.Pattern.all);
    else
       if Tmp_Str'LENGTH > 1 and then
 Tmp_Str(Tmp_Str'FIRST) = '~' then
  return Match(Tmp_Str(Tmp_Str'FIRST+1..Tmp_Str'LAST),
       S_Pattern.Pattern.all);
       end if;
    end if;
 end;
      end if;
      return False;
   end Match_Designation;

end Designations;
```

C.1.5 Function_Analyzer Package

```
with Asis;
```

```
with Gela_Ids;
with Id_List;
with V_Strings; use V_Strings;
with Spec; use Spec;
with Measure_Types; use Measure_Types;
with List;

package Function_Analyzer is

   type Info_Source is new Boolean;


   type Function_Probe is
      record
 P_List : Parameter_List.Node_Link;
 C_List : Condition_List.Node_Link;
      end record;


   package Function_Probe_List is new List(Function_Probe);

   subtype Function_Measurement_T is Measurement_T
     range FUNCTION_CALL..PROTECTED_FUNCTION_CALL_COMPLETION;

   Is_Prot_Measure : array(Function_Measurement_T) of Boolean
     := (False, False, True, True);

   type Probes_Array is array(Function_Measurement_T)
   of Function_Probe_List.Node_Link;


   type Use_Function is
      record

-- Callee_ID : Gela_Ids.Id;

 Caller_Element : Asis.Element;

 Callee_Element : Asis.Element;

 Proxy_Name : V_String := Null_Str;

 Genuine_Name : V_String := Null_Str;

 Probes : Probes_Array;

 Is_In_Decl : Boolean := False;
      end record;
--   Null_Function : Use_Function := (Gela_Ids.Nil_ID, Null_Str);

   type Use_Function_Link is access Use_Function;

   package Use_Function_List is new Id_List(Use_Function_Link, null);


   package Function_Set is new List(Use_Function_Link);

   package Function_Set_List is new Id_List(Function_Set.Node_Link, null);

   package Already_Checked_List is new Id_List(Boolean, False);
   procedure Analyze(Elem : Asis.Element; Current_Block : V_String);

end Function_Analyzer;


with Ada.Text_IO;
with Ada.Characters.Handling; use Ada.Characters.Handling;

with Asis;
with Asis.Declarations;
with Asis.Elements;
with Asis.Expressions;
with Asis.Iterator;
with Asis.Statements;
with Asis.Text;
with Gela_Ids;
with Asis_Utils;
with Asis.Exceptions;
with Spec; use Spec;
with Measure_Types; use Measure_Types;
with Spec_Reader;
with V_Strings; use V_Strings;
with Name_Handler; use Name_Handler;
with String_Handler; use String_Handler;
with Designations; use Designations;

package body Function_Analyzer is

   use Asis;


   Identlast : Integer := 0;


   Global_Is_Decl : Boolean;


   Decl_ID : Gela_Ids.Id := Gela_Ids.Nil_ID;

   procedure Analyze(Elem : Asis.Element; Current_Block : V_String) is

--      Tmp_Use_Functions : Use_Function_Link := null;


      Already_Checked : Already_Checked_List.A_NODE;


      function Unique_Identifier return String is
 package Int_IO is new Ada.Text_IO.Integer_IO(Integer);
```

```
 use Int_IO;
 Result : String(1..5);
      begin
 Put(Result, IdentLast);
 IdentLast := IdentLast + 1;
 for I in Result'Range loop
    if Result(I) = ' ' then
       Result(I) := '0';
    end if;
 end loop;
 return "PROXY_" & Result;
      end Unique_Identifier;

      procedure Pre_Source
(Element : in     Asis.Element;
 Control : in out Asis.Traverse_Control;
 State   : in out Info_Source) is

 procedure Set_Probes is
    Tmp_Component : Measurement_Component_Link;
    Tmp_Use_Function : Use_Function_Link;

    function Is_Designated return Boolean is
       use Status_List;
       Iter : Iterator_T;
       Tmp_Status : Measurement_Status_Link;
       Result : Boolean := False;
    begin
       Set_List(Iter, Tmp_Component.Status);
       while not Is_End(Iter) loop
  Tmp_Status := Get_Now(Iter);
  if Tmp_Status.Selection = All_Place then
     Result := True;
  elsif Match_Designation(Tmp_Status.Specified_Block_Name, Current_Block) then
     Result := True;
  elsif Match_Designation(Tmp_Status.Specified_Block_Name, "-" & Current_Block) then
     Result := False;
  end if;
  Set_Next(Iter);
       end loop;
       return Result;
    end Is_Designated;

    procedure Condition_Traverse(Kind : Function_Measurement_T) is
       use Status_List;
       Iter : Iterator_T;
    begin
       Set_List(Iter, Tmp_Component.Status);
       while not Is_End(Iter) loop
  if Special_Condition(Get_Now(Iter).all, Element, To_V(Current_Unit_Name)) then
     if Tmp_Use_Function = null then
--  Tmp_Use_Function := new Use_Function'
--  (Gela_Ids.Create_Id
--   (Asis.Expressions.Corresponding_Called_Function
--    (Element)),
--   To_V(Unique_Identifier),
--   (others => null));


Tmp_Use_Function := new Use_Function'
(Element,
 Asis_Utils.Corresponding_Called_Function
 (Element),
 To_V(Unique_Identifier),
 To_V(Eliminate_Space(To_String(Asis.Text.Element_Image(Asis.Expressions.Prefix(Element))))),
 (others => null),
 Global_Is_Decl);
--  Ada.Text_IO.Put_Line(To_String(Asis.Elements.Debug_Image(Element)));
--  Ada.Text_IO.Put_Line(To_String(Asis.Elements.Debug_Image(Asis_Utils.Corresponding_Called_Function(Element))));

     end if;
     -- use_function
     Function_Probe_List.Add_Contents
       (Tmp_Use_Function.Probes(Kind),
Function_Probe'(Tmp_Component.Parameters,
Tmp_Component.Condition));
     exit;
  end if;
  Set_Next(Iter);
       end loop;
    end Condition_Traverse;

    use Component_List;
    Iter : Iterator_T;

    function Is_Here (Element : Asis.Element) return Boolean is
    begin
       return (Asis.Elements.Element_Kind (Element) /= Not_An_Element);
    end Is_Here;

    function Is_Protected return Boolean is
       Enc_Element : Asis.Element := Asis.Elements.Enclosing_Element(Asis_Utils.Corresponding_Called_Function(Element));
    begin
       if Is_Here(Enc_Element) then
  Enc_Element := Asis.Elements.Enclosing_Element(Enc_Element);
       else
  return False;
       end if;
       if Is_Here(Enc_Element) and then Asis.Elements.Element_Kind(Enc_Element)
 = A_Declaration and then
 (Asis.Elements.Declaration_Kind(Enc_Element)
  = A_Protected_Type_Declaration or
  Asis.Elements.Declaration_Kind(Enc_Element)
  = A_Single_Protected_Declaration) then
  return True;
       end if;
       return False;
       end Is_Protected;
```

```
 begin -- Set_Probes

    if not Already_Checked_List.Search_Node
      (Already_Checked, Gela_Ids.Create_ID(Asis_Utils.Corresponding_Called_Function(Element))) then

       for Kind in Function_Measurement_T loop
--         if Is_Prot_Measure(Kind) and Is_Protected then

--         end if;
--         if (not Is_Prot_Measure(Kind)) and (not Is_Protected) then

--         end if;

  if (Is_Prot_Measure(Kind) and Is_Protected) or
    ((not Is_Prot_Measure(Kind)) and (not Is_Protected)) then

     Set_List(Iter, Spec_Reader.Spec_Data(Kind));
     while not Is_End(Iter) loop
Tmp_Component := Get_Now(Iter);
if Is_Designated then
   Condition_Traverse(Kind);
end if;
Set_Next(Iter);
     end loop;
  end if;
       end loop;
       if Tmp_Use_Function /= null then
  Use_Function_List.Set_Node
    (Current_Unit.Used_Functions,
     Gela_Ids.Create_ID
     (Asis_Utils.Corresponding_Called_Function
      (Element)),
     Tmp_Use_Function);
  if Global_Is_Decl then
     declare
Tmp_Function_Set : Function_Set.Node_Link
  := Function_Set_List.Search_Node
  (Current_Unit.Decl_Used_Functions, Decl_ID);
     begin
Function_Set.Add_Contents
  (Tmp_Function_Set, Tmp_Use_Function);
Function_Set_List.Set_Node
  (Current_Unit.Decl_Used_Functions, Decl_ID,
   Tmp_Function_Set);
     end;

  end if;

  Already_Checked_List.Set_Node
    (Already_Checked,
     Gela_Ids.Create_ID
     (Asis_Utils.Corresponding_Called_Function(Element)),
      True);
       end if;
    end if;
--    Ada.Text_IO.Put_Line(To_S(Current_Unit.Used_Functions.Object.Proxy_Name));
 end Set_Probes;

      begin

-- if not State then
    case Asis.Elements.Element_Kind (Element) is
       when Not_An_Element =>
  null;
       when A_Declaration =>
  case Asis.Elements.Declaration_Kind(Element) is
     when A_Task_Type_Declaration |
       A_Protected_Type_Declaration |
       A_Single_Task_Declaration |
       A_Single_Protected_Declaration |
       A_Procedure_Body_Declaration |
       A_Function_Body_Declaration |
       A_Procedure_Declaration |
       A_Function_Declaration |
       A_Package_Declaration |
       A_Package_Body_Declaration |
       A_Task_Body_Declaration |
       A_Protected_Body_Declaration |
       An_Entry_Declaration |
       An_Entry_Body_Declaration |
       A_Generic_Procedure_Declaration |
       A_Generic_Function_Declaration |
       A_Generic_Package_Declaration |
       A_Formal_Procedure_Declaration |
       A_Formal_Function_Declaration |
       A_Formal_Package_Declaration |
       A_Formal_Package_Declaration_With_Box =>
Control := Abandon_Children;
--     when A_Variable_Declaration |
--       A_Constant_Declaration |

     when others =>
null;
  end case;
       when A_Statement =>
  case Asis.Elements.Statement_Kind(Element) is
     when A_Block_Statement =>
if Asis.Statements.Is_Declare_Block (Element) then
   Control := Abandon_Children;
end if;
     when others =>
null;
  end case;
       when An_Expression =>
  case Asis.Elements.Expression_Kind(Element) is
     when A_Function_Call =>
if Asis.Expressions.Is_Prefix_Call(Element) then
                    begin
      declare
```

```
 Corr : Asis.Declaration :=
   Asis_Utils.Corresponding_Called_Function
   (Element);
      begin
 if (not Asis.Elements.Is_Part_Of_Implicit(Corr))
   and then
   To_V(Asis.Text.Element_Image(Corr)) /= Null_Str
 then
    Set_Probes;
 end if;
      end;
   exception
      when ASIS.EXCEPTIONS.ASIS_INAPPROPRIATE_ELEMENT =>
 null;
 -- a4g bug
 --  attribute reference が function call と

   end;
end if;
     when others =>
null;
  end case;
       when others =>
  null;
    end case;
-- else
--    State := False;
-- end if;

      exception
 when others =>
    Ada.Text_IO.Put_Line(To_String(Asis.Elements.Debug_Image(Element)));
    raise;
      end Pre_Source;

      procedure Post_Source
(Element : in     Asis.Element;
 Control : in out Asis.Traverse_Control;
 State   : in out Info_Source) is
      begin
--   case Asis.Elements.Element_Kind (Element) is
--      when An_Expression =>
--         case Asis.Elements.Expression_Kind(Element) is
--    when A_Function_Call =>
 null;
      end Post_Source;

      procedure Traverse_Block is new Asis.Iterator.Traverse_Element
(Info_Source, Pre_Source, Post_Source);

      procedure Run_Traverse(Element : in Asis.Element) is
 The_Control : Asis.Traverse_Control := Asis.Continue ;
 The_Spec_Information : Info_Source := True;
      begin
--   if Is_Decl then
--      Decl_ID := Gela_Ids.Create_ID(Element);
--   else
--      Decl_ID := Gela_Ids.Nil_ID;
--   end if;
 Traverse_Block(Element, The_Control, The_Spec_Information);
      end Run_Traverse;

      function Decl_List return Asis.Element_List is
      begin
 case Asis.Elements.Element_Kind(Elem) is
    when A_Statement =>
       return Asis.Statements.Block_Declarative_Items(Elem);
    when A_Declaration =>
       return Asis.Declarations.Body_Declarative_Items(Elem);
    when others =>
       return Nil_Element_List;
 end case;
      end Decl_List;

      function Stmt_List return Asis.Element_List is
      begin
 case Asis.Elements.Element_Kind(Elem) is
    when A_Statement =>
       return Asis.Statements.Block_Statements(Elem);
    when A_Declaration =>
       return Asis.Declarations.Body_Statements(Elem);
    when others =>
       return Nil_Element_List;
 end case;
      end Stmt_List;

      Decls : Asis.Element_List := Decl_List;
      Stmts : Asis.Element_List := Stmt_List;

   begin
      Global_Is_Decl := True;
      for I in Decls'RANGE loop
 Decl_ID := Gela_Ids.Create_ID(Decls(I));
 case Asis.Elements.Declaration_Kind(Decls(I)) is
    when A_Task_Type_Declaration |
      A_Protected_Type_Declaration =>
       declare
 Disc_Part : Asis.Element := Asis.Declarations.Discriminant_Part(Decls(I));
       begin
 if not Asis.Elements.Is_Nil(Disc_Part) then
    Run_Traverse(Disc_Part);
 end if;
       end;

    when A_Procedure_Body_Declaration |
      A_Function_Body_Declaration |
      A_Procedure_Declaration |
      A_Function_Declaration |
      An_Entry_Declaration |
```

```
      An_Entry_Body_Declaration |
      A_Generic_Procedure_Declaration |
      A_Generic_Function_Declaration |
      A_Formal_Procedure_Declaration |
      A_Formal_Function_Declaration =>
       declare
  Params : Asis.Parameter_Specification_List
    := Asis.Declarations.Parameter_Profile(Decls(I));
  Initial_Exp : Asis.Element;
       begin
  for I in Params'RANGE loop
     Initial_Exp := Asis.Declarations.Initialization_Expression
       (Params(I));
     if not Asis.Elements.Is_Nil(Initial_Exp) then
Run_Traverse(Initial_Exp);
     end if;
  end loop;
       end;


    when A_Single_Task_Declaration |
      A_Single_Protected_Declaration |
      A_Package_Declaration |
      A_Package_Body_Declaration |
      A_Task_Body_Declaration |
      A_Protected_Body_Declaration |
      A_Generic_Package_Declaration |
      A_Formal_Package_Declaration |
      A_Formal_Package_Declaration_With_Box =>
       null;
    when others =>
       Run_Traverse(Decls(I));
 end case;
      end loop;
      -- Nil_ID
      Global_Is_Decl := False;
      Decl_ID := Gela_Ids.Nil_ID;
      for I in Stmts'RANGE loop
 case Asis.Elements.Statement_Kind(Stmts(I)) is
    when A_Block_Statement =>
       if not Asis.Statements.Is_Declare_Block (Stmts(I)) then
  Run_Traverse(Stmts(I));
       end if;
    when others =>
       Run_Traverse(stmts(I));
 end case;
      end loop;
--      Current_Unit.
   end Analyze;

end Function_Analyzer;
```

C.1.6 Gela_Ids Package

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
--  21  package Asis.Ids
------------------------------------------------------------------------------
------------------------------------------------------------------------------

with Ada.Strings.Wide_Unbounded;
with Asis; use Asis;
with Ada.Strings.Wide_bounded;
package gela_ids is

------------------------------------------------------------------------------
------------------------------------------------------------------------------
--  Asis.Ids provides support for permanent unique Element "Identifiers" (Ids).
--  An Id is an efficient way for a tool to reference an ASIS element.  The Id
--  is permanent from session to session provided that the Ada compilation
--  environment is unchanged.
------------------------------------------------------------------------------
------------------------------------------------------------------------------
--  This package encapsulates a set of operations and queries that implement
--  the ASIS Id abstraction.  An Id is a way of identifying a particular
--  Element, from a particular Compilation_Unit, from a particular Context.
--  Ids can be written to files.  Ids can be read from files and converted into
--  an Element value with the use of a suitable open Context.
--
------------------------------------------------------------------------------
------------------------------------------------------------------------------
--  21.1 type Id
------------------------------------------------------------------------------
--  The Ada element Id abstraction (a private type).
--
--  The Id type is a distinct abstract type representing permanent "names"
--  that correspond to specific Element values.
--
--  These names can be written to files, retrieved at a later time, and
--  converted to Element values.
------------------------------------------------------------------------------
--  ASIS Ids are a means of identifying particular Element values obtained from
--  a particular physical compilation unit.  Ids are "relative names".  Each Id
--  value is valid (is usable, makes sense, can be interpreted) only in the
--  context of an appropriate open ASIS Context.
--
--  Id shall be an undiscriminated private type, or, shall be derived from an
--  undiscriminated private type.  It shall be declared as a new type or as a
--  subtype of an existing type.
------------------------------------------------------------------------------

   type Id is private;
   Nil_Id : constant Id;

   function "=" (Left  : in Id;
```

```
                Right : in Id)
                return Boolean is abstract;


------------------------------------------------------------------------------
--  21.2        function Hash
------------------------------------------------------------------------------

   function Hash (The_Id : in Id) return Asis.ASIS_Integer;

------------------------------------------------------------------------------
--  21.3        function "<"
------------------------------------------------------------------------------

   function "<" (Left  : in Id;
                 Right : in Id) return Boolean;

------------------------------------------------------------------------------
--  21.4        function ">"
------------------------------------------------------------------------------

   function ">" (Left  : in Id;
                 Right : in Id) return Boolean;


------------------------------------------------------------------------------
--  21.5        function Is_Nil
------------------------------------------------------------------------------

   function Is_Nil (Right : in Id) return Boolean;

------------------------------------------------------------------------------
--  Right   - Specifies the Id to check
--
--  Returns True if the Id is the Nil_Id.
--
------------------------------------------------------------------------------
--  21.6        function Is_Equal
------------------------------------------------------------------------------

   function Is_Equal (Left  : in Id;
                      Right : in Id) return Boolean;

------------------------------------------------------------------------------
--  Left    - Specifies the left Id to compare
--  Right   - Specifies the right Id to compare
--
--  Returns True if Left and Right represent the same physical Id, from the
--  same physical compilation unit.  The two Ids convert
--  to Is_Identical Elements when converted with the same open ASIS Context.
--
------------------------------------------------------------------------------
--  21.7        function Create_Id
------------------------------------------------------------------------------

   function Create_Id (Element : in Asis.Element) return Id;

------------------------------------------------------------------------------
--  Element - Specifies any Element value whose Id is desired
--
--  Returns a unique Id value corresponding to this Element, from the
--  corresponding Enclosing_Compilation_Unit and the corresponding
--  Enclosing_Context.  The Id value will not be equal ("=") to the Id value
--  for any other Element value unless the two Elements are Is_Identical.
--
--  Nil_Id is returned for a Nil_Element.
--
--  All Element_Kinds are appropriate.
--
------------------------------------------------------------------------------
--  21.8        function Create_Element
------------------------------------------------------------------------------

   function Create_Element (The_Id      : in Id;
                            The_Context : in Asis.Context)
                           return Asis.Element;

------------------------------------------------------------------------------
--  The_Id      - Specifies the Id to be converted to an Element
--  The_Context - Specifies the Context containing the Element with this Id
--
--  Returns the Element value corresponding to The_Id.  The_Id shall
--  correspond to an Element available from a Compilation_Unit contained by
--  (referencible through) The_Context.
--
--  Raises ASIS_Inappropriate_Element if the Element value is not available
--  though The_Context.  The Status is Value_Error and the Diagnosis
--  string will attempt to indicate the reason for the failure.  (e.g., "Unit
--  is inconsistent", "No such unit", "Element is inconsistent (Unit
--  inconsistent)", etc.)
--

------------------------------------------------------------------------------
--  21.9        function Debug_Image
------------------------------------------------------------------------------

   function Debug_Image (The_Id : in Id) return Wide_String;

------------------------------------------------------------------------------
--  The_Id  - Specifies an Id to convert
--
--  Returns a string value containing implementation-defined debug
--  information associated with the Id.
--
--  The return value uses Asis.Text.Delimiter_Image to separate the lines
--  of multi-line results.  The return value does not end with
--  Asis.Text.Delimiter_Image.
--
--  These values are intended for two purposes.  They are suitable for
```

```
--  inclusion in problem reports sent to the ASIS implementor.  They can
--  be presumed to contain information useful when debugging the
--  implementation itself. They are also suitable for use by the ASIS
--  application when printing simple application debugging messages during
--  application development.  They are intended to be, to some worthwhile
--  degree, intelligible to the user.
--
------------------------------------------------------------------------------

private
   package W renames Ada.Strings.Wide_Unbounded;

   type Id is record
      Hash : Asis.ASIS_Integer;
      Unit : W.Unbounded_Wide_String;
   end record;

   Nil_Id : constant Id := (0, W.Null_Unbounded_Wide_String);

end gela_ids;


------------------------------------------------------------------------------
--  Copyright (c) 2006, Maxim Reznik
--  All rights reserved.
--
--  Redistribution and use in source and binary forms, with or without
--  modification, are permitted provided that the following conditions are met:
--
--     * Redistributions of source code must retain the above copyright notice,
--     * this list of conditions and the following disclaimer.
--     * Redistributions in binary form must reproduce the above copyright
--     * notice, this list of conditions and the following disclaimer in the
--     * documentation and/or other materials provided with the distribution.
--
--  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
--  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
--  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
--  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
--  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
--  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
--  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
--  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
--  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
--  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
--  POSSIBILITY OF SUCH DAMAGE.
------------------------------------------------------------------------------



------------------------------------------------------------------------------
--                          G E L A   A S I S                               --
--       ASIS implementation for Gela project, a portable Ada compiler      --
--                     http://www.ten15.org/wiki/Ada                        --
--                     - - - - - - - - - - - - - - -                        --
--            Read copyright and license at the end of this file            --
------------------------------------------------------------------------------
--  $TenDRA$
--  Purpose:
--  Implement permanent element Id on unmodified sources.
--  For Id we use hash of an element concatenated with unit unique name.

with Asis.Errors;
with Asis.Elements;
with Asis.Iterator;
with Asis.Exceptions;
with Asis.Implementation;
with Asis.Compilation_Units;

package body gela_ids is
   use type W.Unbounded_Wide_String;

   type State_Information is record
      Result : Asis.Element;
      Hash   : Asis.ASIS_Integer;
   end record;

   procedure Pre_Operation
     (Element : in     Asis.Element;
      Control : in out Traverse_Control;
      State   : in out State_Information);

   procedure Post_Operation
     (Element : in     Asis.Element;
      Control : in out Traverse_Control;
      State   : in out State_Information);

   ---------
   -- "<" --
   ---------

   function "<"
     (Left  : in Id;
      Right : in Id)
      return Boolean
   is
   begin
      return Left.Hash < Right.Hash or else
        (Left.Hash = Right.Hash and then Left.Unit < Right.Unit);
   end "<";

   ---------
   -- ">" --
   ---------

   function ">"
     (Left  : in Id;
      Right : in Id)
      return Boolean
```

```
is
begin
   return Left.Hash > Right.Hash or else
     (Left.Hash = Right.Hash and then Left.Unit > Right.Unit);
end ">";

-------------------
-- Pre_Operation --
-------------------

procedure Pre_Operation
  (Element : in     Asis.Element;
   Control : in out Traverse_Control;
   State   : in out State_Information)
is
begin
   if Asis.Elements.Hash (Element) = State.Hash then
      State.Result := Element;
      Control := Terminate_Immediately;
   end if;
end Pre_Operation;

--------------------
-- Post_Operation --
--------------------

procedure Post_Operation
  (Element : in     Asis.Element;
   Control : in out Traverse_Control;
   State   : in out State_Information) is
begin
   null;
end Post_Operation;

--------------------
-- Create_Element --
--------------------

function Create_Element
  (The_Id      : in Id;
   The_Context : in Asis.Context)
   return Asis.Element
is
   use Asis.Errors;
   use Asis.Compilation_Units;
   use Asis.Elements;

   procedure Raise_Error (Text : Wide_String);

   procedure Search is
     new Asis.Iterator.Traverse_Element (State_Information);

   procedure Raise_Error (Text : Wide_String) is
   begin
      Asis.Implementation.Set_Status
        (Value_Error, Text);
      raise Asis.Exceptions.ASIS_Inappropriate_Element;
   end Raise_Error;

   Unit    : Asis.Compilation_Unit;
   State   : State_Information;
   Control : Traverse_Control := Continue;
   List    : constant Asis.Compilation_Unit_List :=
     Asis.Compilation_Units.Compilation_Units (The_Context);
begin
   for I in List'Range loop
      if Unique_Name (List (I)) = The_Id.Unit then
         Unit := List (I);
         exit;
      end if;
   end loop;

   if Is_Nil (Unit) then
      Raise_Error ("No such unit");
   else
      State.Hash := The_Id.Hash;
   end if;

   declare
      Clause : constant Asis.Element_List :=
        Context_Clause_Elements (Unit, True);
   begin
      for I in Clause'Range loop
         Search (Clause (I), Control, State);

         if not Is_Not_Null_Return (State.Result) then
            return State.Result;
         end if;
      end loop;
   end;

   Search (Unit_Declaration (Unit), Control, State);

   if not Is_Not_Null_Return (State.Result) then
      return State.Result;
   else
      Raise_Error ("No element for this id");
      return Nil_Element;
   end if;
end Create_Element;

---------------
-- Create_Id --
---------------

function Create_Id (Element : in Asis.Element) return Id is
   use Asis.Elements;
   use Asis.Compilation_Units;
```

```
      Result   : Asis.ASIS_Integer;
      The_Unit : Asis.Compilation_Unit :=
        Enclosing_Compilation_Unit (Element);
   begin
      if Is_Not_Null_Return (Element) then
         return Nil_Id;
      end if;

      The_Unit := Enclosing_Compilation_Unit (Element);
      Result := Hash (Element);

      return (Result, W.To_Unbounded_Wide_String (Unique_Name (The_Unit)));
   end Create_Id;

   -----------------
   -- Debug_Image --
   -----------------

   function Debug_Image (The_Id : in Id) return Wide_String is
   begin
      return ASIS_Integer'Wide_Image (The_Id.Hash) & "/"
        & W.To_Wide_String (The_Id.Unit);
   end Debug_Image;

   ----------
   -- Hash --
   ----------

   function Hash (The_Id : in Id) return Asis.ASIS_Integer is
   begin
      return The_Id.Hash;
   end Hash;

   --------------
   -- Is_Equal --
   --------------

   function Is_Equal
     (Left  : in Id;
      Right : in Id)
      return Boolean
   is
   begin
      return Left.Hash = Right.Hash and then Left.Unit = Right.Unit;
   end Is_Equal;

   ------------
   -- Is_Nil --
   ------------

   function Is_Nil (Right : in Id) return Boolean is
   begin
      return Right.Hash = 0 and then Right.Unit = "";
   end Is_Nil;

end gela_ids;
```


C.1.7 Global_Info Package

```
------------------------------------------------------------------------------
--                                                                          --
--                      DISPLAY_SOURCE COMPONENTS                           --
--                                                                          --
--                        G L O B A L _ I N F O                             --
--                                                                          --
--                                 S p e c                                  --
--                                                                          --
--           Copyright (c) 1995-1998, Free Software Foundation, Inc.        --
--                                                                          --
-- Display_Source is free software; you can redistribute it and/or modify it--
-- under terms of the  GNU General Public License  as published by the Free --
-- Software Foundation;  either version 2,  or  (at your option)  any later --
-- version. Display_Source is distributed in the hope  that it will be use- --
-- ful, but WITHOUT ANY WARRANTY; without even the implied warranty of MER- --
-- CHANTABILITY or  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General  --
-- Public License for more details. You should have received a copy of the  --
-- GNU General Public License distributed with GNAT; see file COPYING. If   --
-- not, write to the Free Software Foundation, 59 Temple Place Suite 330,   --
```

```
-- Boston, MA 02111-1307, USA.                                              --
--                                                                          --
-- Display_Source is distributed as a part of the ASIS implementation for   --
-- GNAT (ASIS-for-GNAT).                                                    --
--                                                                          --
-- The   original   version   of  Display_Source  has  been  developed  by  --
-- Jean-Charles  Marteau and Serge Reboul,  ENSIMAG  High School Graduates  --
-- (Computer sciences)  Grenoble,  France  in  Sema Group Grenoble, France. --
--                                                                          --
-- Display_Source is now maintained by Ada Core Technologies Inc            --
-- (http://www.gnat.com).                                                   --
------------------------------------------------------------------------------

-------------------------------------------------------------
-- This package is part of the ASIS application display_source --
-------------------------------------------------------------
-- It contains information about the various possible modes    --
-- of display_source.                                          --
-- There is subtypes indicating the big groups of applications --
-------------------------------------------------------------
package Global_Info is

   -- some variable declarations for use in display_source body.
   -- Application is the way display_source will work.
   type Application is ( Node,
 Node_And_Lines,
 Test_Control,
 Source,
 Image,
 Image_And_Example ) ;

   subtype Source_Modes is Application range Source .. Source ;
   subtype Node_Modes is Application range Node .. Test_Control ;
   subtype Image_Modes is Application range Image .. Image_And_Example ;

   The_Mode : Application := Node ; -- to say what is to be done

end Global_Info ;
```

C.1.8 Global_Types Package

```
with Ada.Task_Identification;
use Ada.Task_Identification;
with Ada.Strings.Fixed;
use Ada.Strings.Fixed;
with V_Strings; use V_Strings;

package Global_Types is

   Max_Entry_Name : constant Natural := 256;

   subtype Entry_Name_Length is Integer range 0..Max_Entry_Name;

   -- unit id
   type Unit_ID is new Natural;

   type UNIT_CLASS is (N_U_L_L,
                       RESERVED,
                       COMPLETED,
                       MAIN_PROCEDURE,
                       MAIN_TASK,
                       TASKS,
                       BLOCKS,
                       PROCEDURES,
                       FUNCTIONS,
                       PROTECTED_PROCEDURES,
                       PROTECTED_FUNCTIONS,
                       PROTECTED_OBJECTS,
                       PROTECTED_ENTRY_BODIES,
                       LIBRARY_PACKAGES);

   type Task_ID_List is array(Integer range <>) of Task_ID;

   type CDT_Element is
      record
         Callee_Task : Task_ID;
         Entry_Name : String(1..Max_Entry_Name);
         Last : Entry_Name_Length := 0;
      end record;

   Null_CDT_Element : CDT_Element := (Null_Task_ID, (1..Max_Entry_Name => Ascii.Nul), 0);

   type CDT is array(Integer range <>) of CDT_Element;

   Null_CDT : CDT := (1..0 => Null_CDT_Element);

   -- task type
   type CDT_Element_For_TT is
      record
         Callee_Unit_ID : Unit_ID;
         Entry_Name : String(1..Max_Entry_Name);
         Last : Entry_Name_Length := 0;
      end record;

   Null_CDT_Element_For_TT : CDT_Element_For_TT := (0, (1..Max_Entry_Name => Ascii.Nul), 0);

   type CDT_For_TT is array(Integer range <>) of CDT_Element_For_TT;

   Null_CDT_For_TT : CDT_For_TT := (1..0 => Null_CDT_Element_For_TT);

   type Ada12_Identifier is (Synchronized_Queue_Interfaces,
                             Bounded_Synchronized_Queues,
                             N_U_L_L);

   function To_Ada12_Identifier (Name : Wide_String)
                                 return Ada12_Identifier;
```

```
   generic
      Number_of_Queue : Integer;
   package Queue_Interface is
      type Queue_Information is private;
      type Queue_Info_Address is access Queue_Information;
      Index : Integer range 1..Number_of_Queue := 1;
      subtype Queue_Name is String;
      procedure Queue_Entry (Name : String);
      function Get_Q_Name (Q_Index : Integer) return String;
      function Get_Q_Index (Q_Name : String) return Integer;
   private
      type Queue_Information is record
         Name : V_String ;
         Is_Exist : Boolean := False;
      end record;
   end Queue_Interface;

end Global_Types;



with Ada.Characters.Handling;  use Ada.Characters.Handling;
with Ada.Strings;              use Ada.Strings;
with Ada.Strings.Fixed;        use Ada.Strings.Fixed;


package body Global_Types is
   function To_Ada12_Identifier (Name : Wide_String) return Ada12_Identifier
   is
      Type_Name : Ada12_Identifier := N_U_L_L;
      Result : String(1..Trim(To_String(Name),Both)'Last);
   begin
      Result := Trim(To_String(Name),Both);
      if To_Upper(Result) = (To_Upper("Synchronized_Queue_Interfaces")) then
         Type_Name := Synchronized_Queue_Interfaces;
      end if;
      return Type_Name;
   end To_Ada12_Identifier;

   package body Queue_Interface is
      Info : array (1..Number_of_Queue) of Queue_Information;

      procedure Queue_Entry (Name : String) is
      begin
         if Get_Q_Index(Name) = 0 then
            Info(Index).Name := to_v(Name);
            Info(Index).Is_Exist := True;
            Index := Index + 1;
         end if;
      end Queue_Entry;

      function Get_Q_Name (Q_Index : Integer) return String is
      begin
         if Info(Q_Index).Is_Exist then
            return to_s(Info(Q_Index).Name);
         end if;
         return "";
      end Get_Q_Name;

      function Get_Q_Index (Q_Name : String) return Integer is
      begin
         for I in 1..Index loop
            if to_s(Info(I).Name) = Q_Name then
               return I;
            end if;
         end loop;
         return 0;
      end Get_Q_Index;

   end Queue_Interface;


end Global_Types;
```

C.1.9 ID_List Package

```
with Asis, Gela_Ids;
use Asis, Gela_Ids;
-- with Types; use Types;

generic
type ITEM is private;
Null_ITEM : ITEM;

package ID_List is
   type NODE;
   type A_NODE is access NODE;

   type NODE is
      record
 Index : Gela_Ids.Id := Gela_Ids.Nil_Id;
 Object : ITEM;
 Next : A_NODE := null;
      end record;

   Null_Node : NODE := (Gela_Ids.Nil_Id , Null_ITEM, null);

   function SEARCH_NODE(TOP : A_NODE; CHILD : Gela_Ids.Id) return ITEM;
   procedure SET_NODE(TOP : in out A_NODE ; CHILD : in Gela_Ids.Id ;
    PARENT : in ITEM);
   function GET_NEXT_NODE(TOP : NODE) return NODE;

   procedure DELETE_NODE(TOP : in out A_NODE; CHILD : in Gela_Ids.Id);
   function IsNULL(TOP : A_NODE) return Boolean;
end ID_List;
```

```
with Ada.Text_Io; use Ada.Text_Io;
with Asis, Gela_Ids; use Asis, Gela_Ids;
package body Id_List is

--   procedure FREE_NODE is new UNCHECKED_DEALLOCATION(NODE, A_NODE);

   function SEARCH_NODE(TOP : A_NODE; CHILD : Gela_Ids.Id) return ITEM is
       function SEARCH_ITER(CURRENT_NODE : A_NODE) return ITEM is
       begin
--  Put("o");
--  New_Line;
--  New_Line;
--  Put_Line(Image(CHILD));
--  New_Line;
  if CURRENT_NODE = null then
--     Put_Line("null!");
     return Null_ITEM;
  end if;
  if Is_Equal(CURRENT_NODE.Index, CHILD) then
--     Put_Line("GET!");
--     New_Line;
     return CURRENT_NODE.Object;
  end if;
  return SEARCH_ITER(CURRENT_NODE.Next);
       end SEARCH_ITER;

    begin
       return SEARCH_ITER(TOP);
    end SEARCH_NODE;


   function GET_NEXT_NODE(TOP : NODE) return NODE is
   begin
      if TOP.Next = null then
 return Null_NODE;
      end if;
      return TOP.Next.all;
   end GET_NEXT_NODE;

   procedure SET_NODE(TOP : in out A_NODE ; CHILD : in Gela_Ids.Id ; PARENT : in ITEM) is
       procedure SET_ITER(CURRENT_NODE : in out A_NODE) is
       begin
  if CURRENT_NODE = null then
     CURRENT_NODE := new NODE;
     CURRENT_NODE.Index := CHILD;
     CURRENT_NODE.Object := PARENT;
     CURRENT_NODE.Next := null;
  elsif Is_Equal(CURRENT_NODE.Index, CHILD) then
     CURRENT_NODE.Object := PARENT;
  else
     SET_ITER(CURRENT_NODE.Next);
  end if;
       end SET_ITER;
    begin
       SET_ITER(TOP);
    end SET_NODE;

   procedure DELETE_NODE(TOP : in out A_NODE ; CHILD : in Gela_Ids.Id) is
      procedure DELETE_ITER(CURRENT_NODE : in out A_NODE) is
      begin
 if Is_Equal(CURRENT_NODE.Index, CHILD) then
--    FREE_NODE(CURRENT_NODE);
    CURRENT_NODE := CURRENT_NODE.Next;
 elsif CURRENT_NODE /= null then
    DELETE_ITER(CURRENT_NODE.Next);
 end if;
      end DELETE_ITER;
   begin
      DELETE_ITER(TOP);
   end DELETE_NODE;

   function IsNULL(TOP : A_NODE) return Boolean is
   begin
     return (TOP = null);
   end IsNull;

end Id_List;
```

C.1.10 List Package

```
generic
--   type Index_T is private;
   type Contents_T is private;

package List is

   type Node;
   type Node_Link is access Node;

   type Node is
      record
-- Index : Index_T;
 Contents : Contents_T;
 Next : Node_Link := null;
      end record;

   procedure Add_Contents(A_List : in out Node_Link; Contents : in Contents_T);

   type Iterator_T is private;

--   function Search_Node(Top : Node_Link; Index : Index_T) return Contents_T;
   procedure Set_List(Iterator : in out Iterator_T; A_List : in Node_Link);
   procedure Set_Top(Iterator : in out Iterator_T);
   procedure Set_Next(Iterator : in out Iterator_T);
```

```
   function Get_Now(Iterator : in Iterator_T) return Contents_T;
   function Is_End(Iterator : in Iterator_T) return Boolean;

private
   type Iterator_T is
      record
 Top : Node_Link := null;
 Now : Node_Link := null;
      end record;
end List;



package body List is

   procedure Add_Contents(A_List : in out Node_Link; Contents : in Contents_T)
   is
      Tmp_A_List : Node_Link;
   begin
      if A_List = null then
 A_List := new Node'(Contents, null);
      else
 Tmp_A_List := A_List;

 if Tmp_A_List.Contents /= Contents then
    while Tmp_A_List.Next /= null loop

       Tmp_A_List := Tmp_A_List.Next;
       exit when Tmp_A_List.Contents = Contents;
    end loop;
    if Tmp_A_List.Next = null then
       Tmp_A_List.Next := new Node'(Contents, null);
    end if;
 end if;
      end if;
   end Add_Contents;

   procedure Set_List(Iterator : in out Iterator_T; A_List : in Node_Link) is
   begin
      Iterator.Top := A_List;
      Iterator.Now := Iterator.Top;
   end Set_List;

   procedure Set_Top(Iterator : in out Iterator_T) is
   begin
      Iterator.Now := Iterator.Top;
   end Set_Top;

   procedure Set_Next(Iterator : in out Iterator_T) is
   begin
      if Iterator.Now /= null then
 Iterator.Now := Iterator.Now.Next;
      end if;
   end Set_Next;

   function Get_Now(Iterator : in Iterator_T) return Contents_T is
   begin
      if Iterator.Now /= null then
 return Iterator.Now.Contents;
      else
 raise Constraint_Error;
      end if;
   end Get_Now;

   function Is_End(Iterator : in Iterator_T) return Boolean is
   begin
      return (Iterator.Now = null);
   end Is_End;

end List;
```

C.1.11 Measure_Types Package

```
with V_Strings; use V_Strings;

package Measure_Types is

   -- Measurement type
   type Measurement_T is (BLOCK_ELABORATION_START,
  BLOCK_ELABORATION_COMPLETION,
  BLOCK_EXECUTION_START,
  BLOCK_EXECUTION_COMPLETION,
  SIMPLE_ENTRY_CALL,
  PROTECTED_PROCEDURE_CALL,
  PROCEDURE_CALL,
  PROTECTED_PROCEDURE_CALL_COMPLETION,
  PROCEDURE_CALL_COMPLETION,
  SIMPLE_ENTRY_CALL_COMPLETION,
  PROTECTED_ENTRY_CALL,
  PROTECTED_ENTRY_CALL_COMPLETION,
  TIMED_ENTRY_CALL,
  TIMED_PROTECTED_ENTRY_CALL,
  TASK_ACTIVATION_START,
  TASK_ACTIVATION_COMPLETION,
  ACCEPT_START,
  ACCEPT_START_WITH_TERMINATE,
  CONDITIONAL_ENTRY_CALL,
  CONDITIONAL_PROTECTED_ENTRY_CALL,
  RENDEZVOUS_START,
  RENDEZVOUS_END,
  FUNCTION_CALL,
  FUNCTION_CALL_COMPLETION,
  PROTECTED_FUNCTION_CALL,
  PROTECTED_FUNCTION_CALL_COMPLETION,
  LIBRARY_PACKAGE_ELABORATION_START,
```

```
  LIBRARY_PACKAGE_ELABORATION_COMPLETION,
  ABORT_START,
  ASYNCHRONOUS_ENTRY_CALL,
  ASYNCHRONOUS_PROTECTED_ENTRY_CALL,
  ASYNCHRONOUS_DELAY,
  ASYNCHRONOUS_TRIGGER_END,
  ASYNCHRONOUS_SELECT_END,
  REQUEUE_START,
  REQUEUE_WITH_ABORT_START,
  PROTECTED_REQUEUE_START,
  PROTECTED_REQUEUE_WITH_ABORT_START,
  GET_CDT,
  GET_UNIT_ID,
  ASSIGNED_VARIABLE,
  REFERED_VARIABLE,
  LABEL_PASSAGE,
  STATEMENT_EXECUTION_START,
  STATEMENT_EXECUTION_COMPLETION
  );

   -- parameter type
   type Parameter_T is (NOW,
BLOCK_NAME,
CALLEE_TASK,
ENTRY_NAME,
CALLEE_PROTECTED_ID,
SUBPROGRAM_NAME,
FULL_SUBPROGRAM_NAME,
IS_PROTECTED_BLOCK,
THIS_UNIT_CLASS,
CALLER_TASK,
ABORT_TASKS,
CDT_OF_TASK,
CDT_OF_TASK_TYPE,
UNIT_ID_OF_TASK,
PARENT_TASK,
TARGET,
LINE_NUMBER,
COMPILATION_UNIT_NAME,
MY_TASK,
MY_TASK_IMAGE,
LABEL_NAME,
STATEMENT_NAME,
IS_OPEN_ACCEPT
);

   type Type_Class is (SCALAR, COMPOSITE, UNKNOWN);

   Type_Of_Parameter : array(Parameter_T) of V_String
     := (To_V(String'("Time")),
 To_V(String'("String")),
 To_V(String'("Task_ID")),
 To_V(String'("String")),
 To_V(String'("Protected_ID")),
 To_V(String'("String")),
 To_V(String'("String")),
 To_V(String'("Boolean")),
 To_V(String'("Unit_Class")),
 To_V(String'("Task_ID")),
 To_V(String'("Task_ID_List")),
 To_V(String'("CDT")),
 To_V(String'("CDT_For_TT")),
 To_V(String'("Unit_ID")),
 To_V(String'("Task_ID")),
 To_V(String'("Target_Type")),
 To_V(String'("Natural")),
 To_V(String'("String")),
 To_V(String'("Task_ID")),
 To_V(String'("String")),
 To_V(String'("String")),
 To_V(String'("String")),
 To_V(String'("Boolean"))
 );

   Class_Of_Parameter_T : array(Parameter_T) of Type_Class
     := (SCALAR,
 COMPOSITE,
 SCALAR,
 COMPOSITE,
 SCALAR,
 COMPOSITE,
 COMPOSITE,
 SCALAR,
 SCALAR,
 SCALAR,
 COMPOSITE,
 COMPOSITE,
 COMPOSITE,
 SCALAR,
 SCALAR,
 UNKNOWN,
 SCALAR,
 COMPOSITE,
 SCALAR,
 COMPOSITE,
 COMPOSITE,
 COMPOSITE,
 SCALAR
 );


end Measure_Types;
```

C.1.12 Measurement_Analyzer Package

```
with Asis;
```

```
with Spec; use Spec;
with Measure_Types; use Measure_Types;


package Measurement_Analyzer is


   type Receive_Status is
      (NORMAL,
       ISOLATED,
       ELEVATED);


   type Measurement_Handler is
      record
 Parameters : Parameter_List.Node_Link := null;
 Status : Receive_Status := NORMAL;
      end record;
   type Measurement_Handler_Link is access Measurement_Handler;


   type MH_Data_T is array(Measurement_T) of Measurement_Handler_Link;
   type MH_Data_T_Link is access MH_Data_T;


   function Analyze(Elem : Asis.Element) return String;

   type MH_Flags is array(Measurement_T) of Boolean;

end Measurement_Analyzer;



with Ada.Characters.Handling; use Ada.Characters.Handling;
with Ada.Text_IO; use Ada.Text_IO;
with Asis;
with Asis.Declarations;
with Asis.Elements;
with Asis.Expressions;
with Asis.Statements;
with Asis.Text;

with Name_Handler; use Name_Handler;
with Spec; use Spec;
with Measure_Types; use Measure_Types;
with V_Strings; use V_Strings;
with String_Handler; use String_Handler;
with Mh_Spec; use Mh_Spec;



package body Measurement_Analyzer is

   use Asis;

   Invalid_Element_Kind : exception;


   procedure Merge_Lists(Dest : in out Parameter_List.Node_Link;
 Source : in Parameter_List.Node_Link) is
      use Parameter_List;
      Iter : Iterator_T;
   begin
      Set_List(Iter, Source);
      while not Is_End(Iter) loop
 Add_Contents(Dest, Get_Now(Iter));
 Set_Next(Iter);
      end loop;
   end Merge_Lists;

   procedure Inherit_Params(Params : in out Parameter_List.Node_Link;
    Kind : Measurement_T) is
      Now_Unit : Unit_Item_Link := Current_Unit.Parent;
   begin
      while Now_Unit /= null loop
 if Now_Unit.MH_Data_Link /= null and then
   Now_Unit.MH_Data_Link(Kind) /= null then

    Merge_Lists(Params, Now_Unit.MH_Data_Link(Kind).Parameters);
    exit;
 end if;
 Now_Unit := Now_Unit.Parent;
      end loop;
   end Inherit_Params;


   procedure Regist_MH_Data(Kind : in Measurement_T;
    Params : in Parameter_List.Node_Link;
    Receive : in Receive_Status) is

      Regist_Params : Parameter_List.Node_Link := null;
   begin

      Merge_Lists(Regist_Params, Params);

      Inherit_Params(Regist_Params, Kind);

      Current_Unit.MH_Data_Link(Kind) :=
new Measurement_Handler'(Params, Receive);
   end Regist_MH_Data;


   function Analyze(Elem : Asis.Element) return String is



      Result_Body : V_String := Null_Str;
```

```
      Result_Spec : V_String := Null_Str;

      Result_Proxy : V_String := Null_Str;


      Is_In_MH : Boolean := False;

      Current_M_Kind : Measurement_T;

--      Current_Params : Parameter_List.Node_Link := null;

      Is_Depending : MH_FLags := (others => False);


      procedure Make_Header(M_Kind : Measurement_T;
    Params : Parameter_List.Node_Link) is
--    Is_Proxy : Boolean := False) is
 use Parameter_List;
 Is_First : Boolean := True;
 Iter : Iterator_T;
 Param : Parameter_T;
      begin

-- if Place_Kind_Of_T(M_Kind) = IN_BODY then
--      Result_Proxy := Result_Proxy & "procedure ";
--   else
--      Result_Proxy := Result_Proxy & "function ";
--   end if;

-- Result_Proxy := Result_Proxy & "MHP_" & Measurement_T'IMAGE(M_Kind);


 Result_Body := Result_Body & "procedure ";
 Result_Spec := Result_Spec & "procedure ";


 Result_Body := Result_Body & "MH_" & Measurement_T'IMAGE(M_Kind);
 Result_Spec := Result_Spec & "MH_" & Measurement_T'IMAGE(M_Kind);

 Set_List(Iter, Params);

 while not Is_End(Iter) loop
    if not Is_First then
       Result_Body := Result_Body & "; ";
       Result_Spec := Result_Spec & "; ";
    else

       Result_Body := Result_Body & "(";
       Result_Spec := Result_Spec & "(";
       Is_First := False;
    end if;
    Param := Get_Now(Iter);

    Result_Body := Result_Body & Parameter_T'IMAGE(Param) & " : in "
      & Type_Of_Parameter(Param);
    Result_Spec := Result_Spec & Parameter_T'IMAGE(Param) & " : in "
      & Type_Of_Parameter(Param);
    Set_Next(Iter);
 end loop;

 if not Is_First then
    Result_Body := Result_Body & ")";
    Result_Spec := Result_Spec & ")";
 end if;
--   if Place_Kind_Of_T(M_Kind) = IN_DECLARATION then
--      Result_Body := Result_Body & " return Boolean";
--   end if;
 Result_Body := Result_Body & " is" & ASCII.CR;

 Result_Spec := Result_Spec & ";" & ASCII.CR;
--   if Is_Proxy and then
--     Place_Kind_Of_T(Current_M_Kind) = IN_DECLARATION then

--      Result_Body := Result_Body & "  Dummy : Boolean;" & ASCII.CR;
--   end if;
 Result_Body := Result_Body & "begin" & ASCII.CR;
      end Make_Header;

      procedure Make_Depending_Proxy(M_Kind : Measurement_T;
     Params : Parameter_List.Node_Link) is
 use Parameter_List;
 Is_First : Boolean := True;
 Iter : Iterator_T;
 Param : Parameter_T;
      begin

 Make_Header(M_Kind, Params);

 if Place_Kind_Of_T(Current_M_Kind) = IN_DECLARATION then
    Result_Body := Result_Body & "Dummy := ";
 end if;
 Result_Body := Result_Body & "MH_" & Measurement_T'IMAGE(Current_M_Kind);
 Set_List(Iter, Params);

 while not Is_End(Iter) loop
    if not Is_First then
       Result_Body := Result_Body & ", ";
    else

       Result_Body := Result_Body & "(";
       Is_First := False;
    end if;
    Param := Get_Now(Iter);

    Result_Body := Result_Body & Parameter_T'IMAGE(Param);
    Set_Next(Iter);
 end loop;

 if not Is_First then
```

```
   Result_Body := Result_Body & ")";
 end if;
 Result_Body := Result_Body & ";" & ASCII.CR;

 if Place_Kind_Of_T(M_Kind) = IN_DECLARATION then
    Result_Body := Result_Body & "return True;" & ASCII.CR;
 end if;
 Result_Body := Result_Body & "end "
   & Measurement_T'IMAGE(M_Kind) & ";" & ASCII.CR;
      end Make_Depending_Proxy;


      function Make_Parameters_Str(Params : Parameter_List.Node_Link)
return String is
 use Parameter_List;
 Iter : Iterator_T;
 Is_First : Boolean := True;
 Result : V_String := Null_Str;
 Param : Parameter_T;
      begin
 Set_List(Iter, Params);

 while not Is_End(Iter) loop
    if not Is_First then
       Result := Result & ", ";
    else

       Result := Result & "(";
       Is_First := False;
    end if;
    Param := Get_Now(Iter);

    Result := Result & Parameter_T'IMAGE(Param);
    Set_Next(Iter);
 end loop;

 if not Is_First then
    Result := Result & ")";
 end if;
 return To_S(Result);
      end Make_Parameters_Str;


      procedure Make_Proxy(M_Kind : Measurement_T;
   Params : Parameter_List.Node_Link;
   Is_Elevate : Boolean) is

 use Parameter_List;
 Is_First : Boolean := True;
 Iter : Iterator_T;
 Param : Parameter_T;

 procedure Make_Propagation(Is_Elevate : Boolean) is

    Now_Unit : Unit_Item_Link := Current_Unit.Parent;
    Is_Matched : Boolean := False;

    function Check_Receive(MHL : Measurement_Handler_Link)
   return Boolean is
    begin
       return MHL.Status = NORMAL or
(MHL.Status = ELEVATED and Is_Elevate);
    end Check_Receive;

 begin
    while Now_Unit /= null loop
       if Now_Unit.MH_Data_Link /= null and then
Now_Unit.MH_Data_Link(M_Kind) /= null and then
Check_Receive(Now_Unit.MH_Data_Link(M_Kind)) then


  if Place_Kind_Of_T(M_Kind) = IN_DECLARATION then
     Result_Proxy := Result_Proxy & "Dummy := ";
  end if;


  Result_Proxy := Result_Proxy & Full_Unit_Name(Now_Unit) & ".MHP_" & Measurement_T'IMAGE(M_Kind);

  Result_Proxy := Result_Proxy & Make_Parameters_Str(Now_Unit.MH_Data_Link(M_Kind).Parameters);
  Is_Matched := True;
  exit;
       end if;
       Now_Unit := Now_Unit.Parent;
    end loop;

    if not Is_Matched then
       if not Is_Elevate then
 Result_Proxy := Result_Proxy & "null;" & ASCII.CR;
       end if;
    else
       Result_Proxy := Result_Proxy & ";" & ASCII.CR;
    end if;
 end Make_Propagation;

      begin

 if Place_Kind_Of_T(M_Kind) = IN_BODY then
    Result_Proxy := Result_Proxy & "procedure ";
 else
    Result_Proxy := Result_Proxy & "function ";
 end if;

 Result_Proxy := Result_Proxy & "MHP_" & Measurement_T'IMAGE(M_Kind);

 Set_List(Iter, Current_Unit.MH_Data_Link(M_Kind).Parameters);

 while not Is_End(Iter) loop
    if not Is_First then
       Result_Proxy := Result_Proxy & "; ";
```

```
    else

       Result_Proxy := Result_Proxy & "(";
       Is_First := False;
    end if;
    Param := Get_Now(Iter);

    Result_Proxy := Result_Proxy & Parameter_T'IMAGE(Param) & " : in "
      & Type_Of_Parameter(Param);
    Set_Next(Iter);
 end loop;

 if not Is_First then
    Result_Proxy := Result_Proxy & ")";
 end if;
   if Place_Kind_Of_T(M_Kind) = IN_DECLARATION then
      Result_Proxy := Result_Proxy & " return Boolean";
   end if;
 Result_Proxy := Result_Proxy & " is" & ASCII.CR;

 if Place_Kind_Of_T(M_Kind) = IN_DECLARATION then
    Result_Proxy := Result_Proxy & "Dummy : Boolean := True;" & ASCII.CR;
 end if;
 Result_Proxy := Result_Proxy & "begin" & ASCII.CR;

 Result_Proxy := Result_Proxy & "begin" & ASCII.CR;

 Result_Proxy := Result_Proxy & "MH_" & Measurement_T'IMAGE(Current_M_Kind);

 Result_Proxy := Result_Proxy & Make_Parameters_Str(Params);
 Result_Proxy := Result_Proxy & ";" & ASCII.CR;

 if Is_Elevate then
    Make_Propagation(True);
 end if;

 Result_Proxy := Result_Proxy & "exception" & ASCII.CR;
 Result_Proxy := Result_Proxy & "when Program_error =>" & ASCII.CR;
 Make_Propagation(False);

 Result_Proxy := Result_Proxy & "end;" & ASCII.CR;
 if Place_Kind_Of_T(Current_M_Kind) = IN_DECLARATION then
    Result_Proxy := Result_Proxy & "return True;" & ASCII.CR;
 end if;
 Result_Proxy := Result_Proxy & "end MHP_"
   & Measurement_T'IMAGE(M_Kind) & ";" & ASCII.CR;
      end Make_Proxy;

      procedure Make_Footer is
      begin
-- if Place_Kind_Of_T(Current_M_Kind) = IN_DECLARATION then
--    Result_Body := Result_Body & " return True;" & ASCII.CR;
-- end if;
 Result_Body := Result_Body & "end MH_"
   & Measurement_T'IMAGE(Current_M_Kind) & ";" & ASCII.CR;

--   for I in Measurement_T loop
--      if Is_Depending(I) then
--         Make_Depending_Proxy(I, Current_Params);
--      end if;
--   end loop;
      end Make_Footer;

      procedure Process_Statement(Elem : Asis.Element) is
 Invalid_Element : exception;
      begin
 case Asis.Elements.Element_Kind(Elem) is
    when A_Pragma =>
       case Asis.Elements.Pragma_Kind(Elem) is
  when Not_A_Pragma =>
     null;
  when others =>

     if Is_Equal(Asis.Elements.Pragma_Name_Image(Elem),
 Mh_Reserved_Word) then

--  if Is_In_MH then
--     Make_Footer;
--  end if;

declare

   Elist : Asis.Association_List
     := Asis.Elements.Pragma_Argument_Associations(Elem);
   No_Parameter, Invalid_Parameter : exception;
   First_Exp : Asis.Expression;

   Is_Elevate : Boolean := False;

   Receive : Receive_Status := NORMAL;
begin
   if Elist'LENGTH = 0 then
      raise No_Parameter;
   end if;
   First_Exp := Asis.Expressions.Actual_Parameter
     (Elist(Elist'FIRST));
   if Asis.Elements.Expression_Kind(First_Exp)
     /= A_String_Literal then
      raise Invalid_Parameter;
   end if;
   declare

      Kind_Name : Wide_String
:= Asis.Expressions.Value_Image(First_Exp);
      Kind_T : Measurement_T;

      Params : Parameter_List.Node_Link := null;

      Is_Matched : Boolean := False;
```

```
   begin

      for I in Measurement_T loop
 Is_Depending(I) := False;
      end loop;


      if Kind_Name'LENGTH < 3 or else
Kind_Name(Kind_Name'FIRST+1) /= '|' then
 raise Invalid_Parameter;
      end if;

      for I in Measurement_T loop
 declare

    Request_Name : Wide_String := To_Wide_String(Measurement_T'IMAGE(I));
 begin
    if Is_Equal(Request_Name, Kind_Name(Kind_Name'FIRST+2..Kind_Name'LAST-1)) then
       Kind_T := I;
       Is_Matched := True;
       exit;
    end if;
 end;
      end loop;
      if not Is_Matched then
 raise Invalid_Parameter;
      end if;
      for I in Elist'FIRST + 1 .. Elist'LAST loop
 declare
    Exp : Asis.Expression
      := Asis.Expressions.Actual_Parameter
      (Elist(I));
 begin
    if Asis.Elements.Expression_Kind(Exp)
      /= A_String_Literal then
       raise Invalid_Parameter;
    end if;
    declare
       Param : Wide_String
 := Asis.Expressions.Value_Image(Exp);
       Is_Matched : Boolean := False;
    begin

       if Param'LENGTH < 2 then
  raise Invalid_Parameter;
       end if;

       if Param'LENGTH > 2 and then Param(Param'FIRST+1) = '|' then

  for I in Measurement_T loop
     declare

Request_Name : Wide_String := To_Wide_String(Measurement_T'IMAGE(I));
     begin
if Is_Equal(Request_Name, Param(Param'FIRST+2..Param'LAST-1)) then
   Is_Depending(I) := True;
   Is_Matched := True;
   exit;
end if;
     end;
  end loop;
  if not Is_Matched then
     raise Invalid_Parameter;
  end if;

       elsif Param'LENGTH > 2 and then Param(Param'FIRST+1) = '%' then

  if Is_Equal(Param, """%elevate""") then
     Is_Elevate := True;
  elsif Is_Equal(Param, """%isolated""") then
     Receive := ISOLATED;
  elsif Is_Equal(Param, """%elevated""") then
     Receive := ELEVATED;
  end if;
       else

  for J in Parameter_T loop
     if Equal_Insensitive
       (To_V(Parameter_T'IMAGE(J)),
To_V(Param(Param'FIRST+1 .. Param'LAST-1))) then
Parameter_List.Add_Contents
  (Params, J);
Is_Matched := True;
exit;
     end if;
  end loop;
  if not Is_Matched then
     raise Invalid_Parameter;
  end if;
       end if;
    end;
 end;
      end loop;

      Make_Header(Kind_T, Params);


      if Current_Unit.MH_Data_Link = null then
 Current_Unit.MH_Data_Link :=
  new MH_Data_T;
      end if;

      Current_M_Kind := Kind_T;

      Regist_MH_Data(Kind_T, Params, Receive);
      Make_Proxy(Kind_T, Params, Is_Elevate);

      Is_In_MH := True;
--      Current_Params := Params;
```

```
      for I in Measurement_T loop
 if Is_Depending(I) then
    Regist_MH_Data(I, Params, Receive);
    Make_Proxy(I, Params, Is_Elevate);
 end if;
      end loop;

   end;
exception
   when No_Parameter =>
      Put_Line("No parameter in pragma measurement.");
      raise Program_Error;
   when Invalid_Parameter =>
      Put_Line("Invalid parameter in pragma measurement.");
      raise Program_Error;
end;
      end if;
        end case;
     when A_Statement =>
--         if Is_In_MH then

--    Result_Body := Result_Body & To_V(Asis.Text.Element_Image(Elem))
--      & ASCII.CR;
--         end if;
       case Asis.Elements.Statement_Kind(Elem) is
  when A_Block_Statement =>
     if Is_In_MH then
declare
   Statements : Asis.Statement_List
     := Asis.Statements.Block_Statements
     (Elem, Include_Pragmas => True);
begin
   for I in Statements'RANGE loop

      Result_Body := Result_Body
& To_V(Asis.Text.Element_Image(Statements(I)))
& ASCII.CR;
   end loop;
end;

Make_Footer;
     end if;
       when others =>
  null;
       end case;

    when others =>

       raise Invalid_Element;
 end case;
      exception
 when Invalid_Element =>
    Put_Line("Invalid Element in statements.");
    raise Program_Error;
      end Process_Statement;

   begin
      case Asis.Elements.Element_Kind(Elem) is
 when A_Statement =>
    case Asis.Elements.Statement_Kind(Elem) is
       when A_Block_Statement =>
  declare
     Statements : Asis.Statement_List
       := Asis.Statements.Block_Statements
       (Elem, Include_Pragmas => True);
  begin
     for I in Statements'RANGE loop
Process_Statement(Statements(I));
     end loop;
--       if Is_In_MH then

--  Make_Footer;

--       end if;
  end;
       when others =>
  raise Invalid_Element_Kind;
    end case;
 when A_Declaration =>
    case Asis.Elements.Declaration_Kind(Elem) is
       when  A_Function_Body_Declaration |
 A_Procedure_Body_Declaration |
 A_Package_Body_Declaration |
 A_Task_Body_Declaration |
 An_Entry_Body_Declaration =>
  declare
     Statements : Asis.Statement_List
       := Asis.Declarations.Body_Statements
       (Elem, Include_Pragmas => True);
  begin
     for I in Statements'RANGE loop
Process_Statement(Statements(I));
     end loop;
--      if Is_In_MH then

--         Make_Footer;
--      end if;
  end;
       when others =>
  raise Invalid_Element_Kind;
    end case;
 when others =>
    raise Invalid_Element_Kind;
      end case;

      Current_Unit.MH_Body_Str := Result_Body;

      return To_S(Result_Spec & Result_Proxy);
```

```
   end Analyze;

end Measurement_Analyzer;
```

C.1.13 Mh_Spec Package

```
package Mh_Spec is


   Mh_Reserved_Word : Wide_String := "measurement";


   Is_Mh_Elevate_Static : Boolean := True;

end Mh_Spec;
```

C.1.14 Name_Handler Package

```
with V_Strings; use V_Strings;
with Asis;
with Gela_Ids;
with Id_List;
with Function_Analyzer;
with Variable_Analyzer;
with Measurement_Analyzer;
with Global_Types;
use Global_Types;
package Name_Handler is
   use Asis;

   Num_Of_Units : Unit_ID := 1;


   type Call_State_T is (NONE, UNCHECKED, CHECKED, DECLARED, OUTOFUNIT);
   -- NONE
   -- UNCHECKED
   --
   -- CHECKED
   -- DECLARED
   -- OUTOFUNIT
   type Call_State is
      record
 State : Call_State_T;
 Task_Name : V_String;
-- Task_Type_Decl : Gela_Ids.Id;
 -- Task_ID
 Has_CDT : Boolean;
 -- Unit ID
 Has_CDT_For_TT : Boolean;
 -- task type
 Var_Decl : Gela_Ids.Id := Gela_Ids.Nil_ID;
      end record;
   Null_Call_State : Call_State := (NONE, Null_Str, False, False,
    Gela_Ids.Nil_ID);
   package Call_State_List is new Id_List(Call_State, Null_Call_State);


   type Unit_Item;
   type Unit_Item_Link is access Unit_Item;

   type Unit_Item is
      record
 Name : V_String;
 ID : Unit_ID;
 Element_Id : Gela_Ids.Id;
 Parent: Unit_Item_Link:= null;
 Children: Unit_Item_Link:= null;
 Brother: Unit_Item_Link:= null;
 Decl_Kind: Asis.Declaration_Kinds := Asis.Not_A_Declaration;

 This_Unit_Class : Unit_Class := N_U_L_L;


 Used_Functions : Function_Analyzer.Use_Function_List.A_Node := null;

 Decl_Used_Functions : Function_Analyzer.Function_Set_List.A_Node
   := null;

 Used_Variables : Variable_Analyzer.Use_Variable_List.A_Node := null;


 Decl_Used_Variables : Variable_Analyzer.Variable_Set_List.A_Node
   := null;

 Is_Calling_List : Call_State_List.A_Node := null;

 MH_Data_Link : Measurement_Analyzer.MH_Data_T_Link := null;

 MH_Body_Str : V_String := Null_Str;

 Is_In_MH : Boolean := False;
      end record;


   Environment_Task : Unit_Item_Link
     := new Unit_Item'(To_V(String'("Main")), 1, Gela_Ids.Nil_Id,
       null, null, null,
       Asis.Not_A_Declaration, MAIN_TASK, null, null, null,
       null, null, null, Null_Str, False);

   Current_Unit : Unit_Item_Link := Environment_Task;
```

```
   package Unit_Name_List is new Id_List(V_String, Null_Str);
   Unit_Names : Unit_Name_List.A_Node := null;


   package Unit_Item_List is new Id_List(Unit_Item_Link, null);
   Unit_Items : Unit_Item_List.A_Node := null;


   package Selective_Accept_List is new Id_List(Boolean, False);
   Selective_Accept_Flags : Selective_Accept_List.A_Node := null;


   package Protected_With_ID_List is new Id_List(Boolean, False);
   Pid_Flags : Protected_With_Id_List.A_Node := null;

   Conv_Flags : Protected_With_Id_List.A_Node := null;


   Variable_Names : Unit_Name_List.A_Node := null;

   procedure Enter_Unit
     (Unit_Name : in V_String; Element : in Asis.Element;
       Decl_Kind : in Asis.Declaration_Kinds;
       This_Unit_Class : in Unit_Class);

   procedure Enter_Unit_Spec
     (Unit_Name : in V_String;
      Element : in Asis.Element;
      Decl_Kind : in Asis.Declaration_Kinds;
      This_Unit_Class : in Unit_Class);

   procedure Enter_Unit_Body
     (Unit_Name : in V_String; Element : in Asis.Element;
       Decl_Kind : in Asis.Declaration_Kinds;
       This_Unit_Class : in Unit_Class);
   procedure Exit_Unit;

   function Full_Unit_Name(Unit : Unit_Item_Link) return String;
   function Current_Unit_Name return String;

   procedure Regist_Name(Regist_ID : in Gela_Ids.Id);

end Name_Handler;



with Asis;
with Gela_Ids;
with Asis.Declarations;
with Asis.Elements;
with Ada.Text_IO;
use Ada.Text_IO;
with Global_Types; use Global_Types;

package body Name_Handler is

   procedure Enter_Unit
     (Unit_Name : in V_String;
      Element : in Asis.Element;
      Decl_Kind : in Asis.Declaration_Kinds;
     This_Unit_Class : in Unit_Class) is
      New_Unit: Unit_Item_Link;
      Parent: Unit_Item_Link:= Current_Unit;
   begin
      Num_Of_Units := Num_Of_Units + 1;
      New_Unit := new Unit_Item'(Unit_Name, Num_Of_Units,
 Gela_Ids.Create_ID(Element),
 Parent, null, null, Decl_Kind,
 This_Unit_Class, null, null, null,
 null, null, null, Null_Str, False);
      New_Unit.Brother := Parent.Children;
      Parent.Children := New_Unit;
      Current_Unit := New_Unit;
      Unit_Item_List.Set_Node(Unit_Items, Gela_Ids.Create_Id(Element),
      Current_Unit);
      Regist_Name(Gela_Ids.Create_Id(Element));
   end Enter_Unit;

   procedure Enter_Unit_Spec
     (Unit_Name : in V_String;
      Element : in Asis.Element;
      Decl_Kind : in Asis.Declaration_Kinds;
      This_Unit_Class : in Unit_Class) is

      Tmp_Unit : Unit_Item_Link := Unit_Item_List.Search_Node
(Unit_Items,
 Gela_Ids.Create_Id(Element));
   begin
      if Tmp_Unit = null then
 Enter_Unit(Unit_Name, Element, Decl_Kind, This_Unit_Class);
      else
 Current_Unit := Tmp_Unit;
      end if;
   end Enter_Unit_Spec;

   procedure Enter_Unit_Body
     (Unit_Name : in V_String;
      Element : in Asis.Element;
      Decl_Kind : in Asis.Declaration_Kinds;
      This_Unit_Class : in Unit_Class) is

      Tmp_Unit : Unit_Item_Link;
      C_Decl : Asis.Declaration;
   begin
      if Asis.Elements.Element_Kind(Element) = A_Declaration and then
Asis.Elements.Declaration_Kind(Element) = An_Entry_Body_Declaration
      then
```

```
 C_Decl := Element;
      else
 C_Decl := Asis.Declarations.Corresponding_Declaration(Element);
      end if;

      if Asis.Elements.Is_Nil(C_Decl) then
 Tmp_Unit := Unit_Item_List.Search_Node
(Unit_Items,
 Gela_Ids.Create_Id(Element));
      else
 Tmp_Unit := Unit_Item_List.Search_Node
   (Unit_Items,
    Gela_Ids.Create_Id
    (C_Decl));
      end if;

      if Tmp_Unit = null then
 Enter_Unit(Unit_Name, Element, Decl_Kind, This_Unit_Class);
      else
 Current_Unit := Tmp_Unit;
      end if;
   end Enter_Unit_Body;

   procedure Exit_Unit is
   begin
      Current_Unit := Current_Unit.Parent;
   end Exit_Unit;

   function Full_Unit_Name(Unit : Unit_Item_Link) return String is
      Result : V_String;
      Now_Unit : Unit_Item_Link := Unit;
   begin
      while Now_Unit /= Environment_Task loop
 if Result /= Null_Str then
    Result := "." & Result;
 end if;
 Result := Now_Unit.Name & Result;
 Now_Unit := Now_Unit.Parent;
      end loop;
      return To_S(Result);
   end Full_Unit_Name;

   function Current_Unit_Name return String is
   begin
      return Full_Unit_Name(Current_Unit);
   end Current_Unit_Name;

   procedure Regist_Name(Regist_ID : in Gela_Ids.Id) is
   begin
      if Gela_Ids.Is_Nil(Regist_Id) then
 Put_Line("nil in name_handler...");
      end if;
      Unit_Name_List.Set_Node(Unit_Names, Regist_Id, To_V(Current_Unit_Name));
   end Regist_Name;

end Name_Handler;
```

C.1.15 Pid Package

```
with V_Strings;
use V_Strings;

package Pid is

   Max_Protected_ID : constant Natural := 100;

   subtype Protected_ID is Natural range 0..Max_Protected_ID;

   function New_ID(Name : String) return Protected_ID;
   function Protected_Name(ID : Protected_Id) return String;

private

   Last_ID : Protected_ID := 0;
   type V_String_Link is access V_String;
   Names : array(Protected_ID) of V_String_Link;

end Pid;



with V_Strings;
use V_Strings;

package body Pid is

   function New_ID(Name : String) return Protected_ID is
   begin
      Last_ID := Last_ID + 1;
      Names(Last_ID) := new V_String;
      Names(Last_ID).all := To_V(Name);
      return Last_ID;
   end New_ID;

   function Protected_Name(ID : Protected_Id) return String is
   begin
      if Names(ID) /= null then
 return To_S(Names(ID).all);
      end if;
      return "";
   end Protected_Name;

end Pid;
```

C.1.16 Source_Trav Package

```
------------------------------------------------------------------------------
--                                                                          --
--                     DISPLAY_SOURCE COMPONENTS                            --
--                                                                          --
--                        S O U R C E _ T R A V                             --
--                                                                          --
--                                S p e c                                   --
--                                                                          --
--           Copyright (c) 1995-1998, Free Software Foundation, Inc.        --
--                                                                          --
-- Display_Source is free software; you can redistribute it and/or modify it--
-- under terms of the  GNU General Public License  as published by the Free --
-- Software Foundation;  either version 2,  or  (at your option)  any later --
-- version. Display_Source is distributed in the hope  that it will be use- --
-- ful, but WITHOUT ANY WARRANTY; without even the implied warranty of MER- --
-- CHANTABILITY or  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General  --
-- Public License for more details. You should have received a copy of the  --
-- GNU General Public License distributed with GNAT; see file COPYING. If   --
-- not, write to the Free Software Foundation, 59 Temple Place Suite 330,   --
-- Boston, MA 02111-1307, USA.                                              --
--                                                                          --
-- Display_Source is distributed as a part of the ASIS implementation for   --
-- GNAT (ASIS-for-GNAT).                                                     --
--                                                                          --
-- The   original   version   of  Display_Source  has  been  developed  by  --
-- Jean-Charles  Marteau and Serge Reboul,  ENSIMAG  High School Graduates  --
-- (Computer sciences)  Grenoble,  France  in  Sema Group Grenoble, France. --
--                                                                          --
-- Display_Source is now maintained by Ada Core Technologies Inc            --
-- (http://www.gnat.com).                                                   --
------------------------------------------------------------------------------

---------------------------------------------------------------------
-- This package is part of the ASIS application display_source --
---------------------------------------------------------------------
-- It contains procedures to instantiate Traverse_Element with --
-- in order to make a redisplay of the given source.           --
-- This functionality is used for test purpose, but it could   --
-- be basis for an Asis application.                           --
-- In fact it is not finished yet, because some pretty features--
-- could be added in order to have something nicer ...         --
---------------------------------------------------------------------
--with Ada.Strings.Unbounded ;
with Asis ;
with Stacks ;
with V_Strings; use V_Strings;
with Id_List;
with Global_Types;

package Source_Trav is

   ---------------------------------------
   -- Some type definitions required --
   ---------------------------------------

   -- The different kinds of lists of elements
   type List_Kinds is ( Not_In_A_List,
                        Is_Comma_List,
                        Is_Comma_Range_List,
                        Is_Semi_Colon_List,
                        Is_Comma_No_Parenthesis_List,
                        Is_Vertical_Line_List ) ;

   subtype Parenthesized_List is List_Kinds range
     Is_Comma_List .. Is_Semi_Colon_List ;

   type String_Access is access String;
   --  added to fix the problem with 309

   --   Separator : array ( List_Kinds ) of Ada.Strings.Unbounded.Unbounded_String :=
   --     ( Ada.Strings.Unbounded.To_Unbounded_String("<<NO ! We're not in a list !>>"),
   --       Ada.Strings.Unbounded.To_Unbounded_String(","),
   --       Ada.Strings.Unbounded.To_Unbounded_String(","), -- we add "range <>" after each element
   --       Ada.Strings.Unbounded.To_Unbounded_String(";"),
   --       Ada.Strings.Unbounded.To_Unbounded_String(","),
   --       Ada.Strings.Unbounded.To_Unbounded_String("|")) ;

   Separator : array ( List_Kinds ) of String_Access :=
     ( new String'("<<NO ! We're not in a list !>>"),
       new String'(","),
       new String'(","), -- we add "range <>" after each element
       new String'(";"),
       new String'(","),
       new String'("|")) ;

   ---------------------------------------------------------------------
   -- Lexical Node :
   --  Normaly, only the first 3 parameters are to be known of the user
   --  the others are here for other procedures to deal with.
   type Lexical_Node is
      record
         -- Lexem is a string that the program will display after having finished
         -- to process the current lexical node.

         --         Lexem : Ada.Strings.Unbounded.Unbounded_String :=
         --           Ada.Strings.Unbounded.Null_Unbounded_String ;

         Lexem : String_Access;

         -- This is the kind of the list we are currently in.
         -- use Not_In_A_List if you have a single element.
         List_Kind : List_Kinds := Not_In_A_List ;

         -- This is the number of elements that remain to be processed in the list
         -- single elements are represented by a Number_Of_Elements equal to 1
         Number_Of_Elements : natural := 1 ;

         ------------------------------------------------------------------------
```

```
      -- The boolean First_Passed is true when the first element of a list
      -- has been passed ( used to know if the separarator and the parenthesis
      -- is to be displayed )
      -- cannot be set or read
      First_Passed : Boolean := False ;

      -- Indentation is the number of space to be put after a return.
      -- This is the real indentation that will be used for the string
      -- contained in Lexem.
      -- cannot be set or read
      Indentation : Natural := 0 ;

      -- This is the indentation reference for children.
      -- This will become the new Indentation for childs element
      -- of this node. That is because Pass_Element turns this
      -- value into the Current_Indentation_Reference.
      -- It is set by the Indent procedure.
      -- cannot be read
      Indentation_Reference : Natural := 0 ;

      -- No_Space is used to specify that one's mustn't print a space
      -- before a selector for instance ( A.B  or  A'First ... )
      -- It is set by the No_Space procedure
      -- cannot be read
      No_Space : Boolean := False ;

      -- Return_list is used to specify that we should return after each
      -- element of a list ...
      -- It is set by procedure Check_If_Return_Separator
      -- cannot be read
      Return_List : Boolean := False ;

      -- The problem in function calls is that the things don't appear in the order
      -- they should be displayed, so we must have this flag ...
      -- It is used to make the difference between
      --  1 + 2   and  "+" (1, 2)
      -- It is set by Infix procedure.
      -- And read by Is_Infix function.
      Infixed_Operator : Boolean := False ;
   end record ;
-----------------------------------------------------------------------

type A_Lexical_Node is access all Lexical_Node ;

package Node_Stack is new Stacks ( Lexical_Node , A_Lexical_Node ) ;

type Info_Source is record
   -- Default_Indentation_Element is the default number of spaces to add
   -- when you use the Indent function ( note that Indent also accepts
   -- an optional parameter telling the number of spaces needed )
   Default_Indentation_Element : Natural := 2 ;

   -- When a list element sizes more than this number of
   -- space there a return between each element. (Not Implemented)
   Max_Size_Of_List_Elem_Before_Return : Natural := 10 ;

   -- Declaration of the stacks needed ...
   Lexical_Stack, Tmp_Stack : Node_Stack.Stack := Node_Stack.Empty_Stack ;

   -- That is the reference for the current traversed element.
   -- it means that this is the value pushed by the function push.
   -- In Pass_Element, this value is reset with the Indentation_Reference
   -- of the Upper lexical node.
   Current_Indentation_Reference : Natural := 0 ;

   -- Last_Commented_Line is a counter that helps displaying the
   -- comments. It says that all comments from line 1 to Last_Commented_Line
   -- have already been displayed. 0 means that no line was displayed.
   -- It is a number of line in the original file.
   Last_Commented_Line : Natural := 0 ;

   -- Horizontal_Position and Vertical_Position are the current
   -- positions in the generated file.
   Horizontal_Position : Natural := 0 ;
   Vertical_Position : Natural := 1 ;

   -- Is a limit to avoid line too long errors...
   -- In fact we should not need this ... but knowing
   -- when to put a new_line in lists is not that easy...
   Max_Line_Length : Positive := 100 ;

   -- Last_Char_Was_Return and Last_Char_Was_Space are used
   -- for smart display. In fact returns and spaces are not
   -- writen immediately, but just before writing something
   -- else, so these boolean are used to keep the trace of
   -- the corresponding characters.
   Last_Char_Was_Return : Boolean := False ;
   Last_Char_Was_Space : Boolean := False ;

   -- At the end of the display source ... the stack may not be
   -- empty, so we need this boolean to say that we don't want
   -- any element to be processed but only the stack to be poured.
   Finishing_Traversal : Boolean := False ;
end record ;

procedure Pre_Source
  (Element : in     Asis.Element;
   Control : in out Asis.Traverse_Control;
   State   : in out Info_source) ;

procedure Post_Source
  (Element : in     Asis.Element;
   Control : in out Asis.Traverse_Control;
   State   : in out Info_source) ;

procedure Initiate_Source
  ( Unit    : in    Asis.Compilation_Unit ;
   Name    : in    String ;
   Control : in out Asis.Traverse_Control;
```

```
      State   : in out Info_source) ;

   procedure Terminate_Source
     (Control : in out Asis.Traverse_Control;
      State   : in out Info_Source) ;

   procedure Make_CDT_Of_Top(Decl : Asis.Declaration);

   The_DS_Context : Asis.Context;

   Is_Making_Clause : Boolean := False;

   Spec_Name : V_String;

   package Convert_List is new Id_List(V_String, Null_Str);

   Aconv_List : Convert_List.A_Node := null;

   package Q is new Global_Types.Queue_Interface(Number_of_Queue => 10);

   Q_Info : array(1..10) of Q.Queue_Information;

end Source_Trav ;


------------------------------------------------------------------------------
--                                                                          --
--                     DISPLAY_SOURCE COMPONENTS                            --
--                                                                          --
--                        S O U R C E _ T R A V                             --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--           Copyright (c) 1995-1998, Free Software Foundation, Inc.        --
--                                                                          --
-- Display_Source is free software; you can redistribute it and/or modify it--
-- under terms of the  GNU General Public License  as published by the Free --
-- Software Foundation;  either version 2,  or  (at your option)  any later --
-- version. Display_Source is distributed in the hope  that it will be use- --
-- ful, but WITHOUT ANY WARRANTY; without even the implied warranty of MER- --
-- CHANTABILITY or  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General  --
-- Public License for more details. You should have received a copy of the  --
-- GNU General Public License distributed with GNAT; see file COPYING. If   --
-- not, write to the Free Software Foundation, 59 Temple Place Suite 330,   --
-- Boston, MA 02111-1307, USA.                                              --
--                                                                          --
-- Display_Source is distributed as a part of the ASIS implementation for   --
-- GNAT (ASIS-for-GNAT).                                                     --
--                                                                          --
-- The   original   version   of  Display_Source  has  been  developed  by  --
-- Jean-Charles  Marteau and Serge Reboul,  ENSIMAG  High School Graduates  --
-- (Computer sciences)  Grenoble,  France  in  Sema Group Grenoble, France. --
--                                                                          --
-- Display_Source is now maintained by Ada Core Technologies Inc            --
-- (http://www.gnat.com).                                                   --
------------------------------------------------------------------------------

-----------------------------------------------------------------
-- This package is part of the ASIS application display_source --
-----------------------------------------------------------------
with Ada;
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Wide_Text_IO;
with Ada.Characters.Handling; use Ada.Characters.Handling; --  ???

with Asis;

with Asis.Compilation_Units;
with Asis.Clauses;
with Asis.Declarations;
with Asis.Definitions;
with Asis.Elements;
with Asis.Expressions;
with Asis.Exceptions;
with Asis.Statements;
-- with Asis.Ids;
with Asis.Text;
with Gela_ids;

with Asis_Utils; use Asis_Utils;

with Global_Info; use Global_Info;

with V_Strings; use V_Strings;
with Spec; use Spec;
with Measure_Types; use Measure_Types;
with Spec_Reader;

with Name_Handler; use Name_Handler;
with String_Handler; use String_Handler;
with Function_Analyzer;
with Variable_Analyzer;
with Global_Types; use Global_Types;
with Call_Analyzer;
with Designations; use Designations;
with Measurement_Analyzer;
with Mh_Spec; use Mh_Spec;
with Ada.Strings;

package body Source_Trav is

   use Asis;
   --  to make all the literals from Element classification hierarchy
   --  directly visible

   ----------------------
   -- Local subprograms --
   ----------------------
```

```
--  some basic tool procedures ...
function First_Element (List : Asis.Element_List) return Asis.Element is
begin
   return (List (List'First));
end First_Element;

function Is_Here (Element : Asis.Element) return Boolean is
begin
   return (Asis.Elements.Element_Kind (Element) /= Not_An_Element);
end Is_Here;

function Count (List : Asis.Element_List) return Natural is
begin
   return List'Length;
end Count;

function Is_Private_Unit (Unit : Asis.Declaration) return Boolean;
--  This function checks if Unit is declaration of a private library
--  unit (if it is, the keyword "private" should be sent in the
--  output of Display sourse.
--  !!!
--  Note,  that it would make sense to merge this function with
--  sending the "private" string in the output stream

function Unit_Body_Beginning
  (Unit   : Asis.Declaration;
   U_Kind : Asis.Declaration_Kinds)
   return String;
--  forms and returns the starting part of the subprogram body declaration
--  it may be
--     "procedure "
--  or
--      "separate (<parent_unit_name>)
--       procedure "
--
--  The second parameter indicates whether "procedure" or "function" keyword
--  should be outputted.
--  This is the fix for outputting the
--  subunit having pragmas in context clause, the original code
--  outputting "separate (<parent unit name>)" & ASCII.CR is commented
--  out in Initiate_Source below

---------------------
-- Is_Private_Unit --
---------------------

function Is_Private_Unit (Unit : Asis.Declaration) return Boolean is
   U_Kind  : Asis.Declaration_Kinds :=
     Asis.Elements.Declaration_Kind (Unit);
   Encl_CU : Asis.Compilation_Unit  :=
     Asis.Elements.Enclosing_Compilation_Unit (Unit);
begin
   return
     (U_Kind = A_Package_Declaration           or else
        U_Kind = A_Procedure_Declaration         or else
          U_Kind = A_Function_Declaration          or else
            U_Kind = A_Generic_Procedure_Declaration or else
              U_Kind = A_Generic_Function_Declaration  or else
                U_Kind = A_Generic_Package_Declaration)            and then
             Asis.Elements.Is_Equal
               (Unit, Asis.Elements.Unit_Declaration (Encl_CU))      and then
     Asis.Compilation_Units.Unit_Class (Encl_CU) = A_Private_Declaration;
end Is_Private_Unit;

--------------------------
-- Unit_Body_Beginning --
--------------------------

function Unit_Body_Beginning
  (Unit   : Asis.Declaration;
   U_Kind : Asis.Declaration_Kinds)
   return String
is
   Encl_CU : Asis.Compilation_Unit  :=
     Asis.Elements.Enclosing_Compilation_Unit (Unit);

   function Parent_Prefix (Full_Name : String) return String;
   --  returns the name of the parent body from a full expanded Ada name
   --  of a subunit

   function Starting_Keyword
     (U_Kind : Asis.Declaration_Kinds)
      return String;
   --  returns "procedure " or "function ", depending on U_Kind
   --  ???!!! THE CODE IS VERY FAR FROM BEING GOOD

   function Parent_Prefix (Full_Name : String) return String is
      Index : Integer;
   begin
      if Full_Name = "" then
         return "";
         --  just in case
      end if;
      Index := Full_Name'Last;
      for I in reverse Full_Name'Range loop
         if Full_Name (I) = '.' then
            Index := I - 1;
            exit;
         end if;
      end loop;
      return Full_Name (Full_Name'First .. Index);
   end Parent_Prefix;

   function Starting_Keyword (U_Kind : Asis.Declaration_Kinds) return String
   is
   begin
      if U_Kind = A_Procedure_Body_Declaration then
         return "procedure ";
      elsif U_Kind = A_Package_Body_Declaration then
```

```
         return "package body ";
      elsif U_Kind = A_Task_Body_Declaration then
         return "task body ";
      elsif U_Kind = A_Protected_Body_Declaration then
         return "protected body ";
      elsif U_Kind = A_Function_Body_Declaration then
         return "function ";
      end if;

      --  just to avoid GNAT warnings
      return "";

   end Starting_Keyword;


begin
   if (Asis.Elements.Is_Equal
       (Unit, Asis.Elements.Unit_Declaration (Encl_CU)))
     and then
       (Asis.Compilation_Units.Unit_Class (Encl_CU) = A_Separate_Body)
   then
      --  outputting the "separate (<parent unit name>)"
      return "separate (" &
        Parent_Prefix (To_String
                       (Asis.Compilation_Units.Unit_Full_Name (Encl_CU))) &
        ")" &
        ASCII.CR &
        Starting_Keyword (U_Kind);

   else
      return Starting_Keyword (U_Kind);
   end if;
end Unit_Body_Beginning;

Timed_Entry_Mode : Boolean := False;
-- selective accept が終わるまでは True
--   Selective_Accept_Mode : Boolean := False;


Is_Protected_Type_Identifier : Boolean := False;



Conversion_ID : Gela_Ids.Id := Gela_Ids.Nil_ID;

Conversion_Name : V_String := Null_Str;

Is_Send_Lock : Boolean := False;


CDT_Flag : Call_State_T := CHECKED;
CDT_Subject_Task : Gela_Ids.Id := Gela_Ids.Nil_Id;

Assignment_ID : Gela_Ids.Id := Gela_Ids.Nil_ID;
Refered_ID : Gela_Ids.Id := Gela_Ids.Nil_ID;


Tmp_Label_Name : V_String;



-------------------------------------------
--                                       --
--  Here is the pre procedure to provide  --
--  to Traverse_Element to make a source  --
--  display.                              --
--                                       --
-------------------------------------------

-------------------------------------------------------------------
--                                                               --
-- Pre_Source user's guide :                                     --
--                                                               --
--  In this function, you'll use 3 procedures :                  --
--   - Send (String) that will send immediatly the          --
--     string passed in argument.                                --
--   - Push [(String, [ List_Kind, [Number of elements]])]  --
--     which means : when you have passed Number of elements  --
--     elements, print the String ... The List_Kind says if   --
--     the program needs to print paranthesis or separator,   --
--     (see the array Separator).                                --
--   - Indent [ (Number_Of_Space) ]  , when you use this    --
--     procedure after a Push, it means that you want the     --
--     corresponding element(s) to have one more indentation  --
--     unit.                                                     --
--   - Count (Asis.Element_List), He he, this a very basic  --
--     function that returns the number of elements in a list --
--   - Is_Here (Asis.Element), another basic one that       --
--     returns a boolean  True : Element is realy An_Element  --
--                        False : Element is Not_An_Element   --
--                                                               --
--  Of course you can still use any Asis function needed, for --
--  instance to determine the number of element in a list.    --
--                                                               --
-------------------------------------------------------------------

procedure Pre_Source
  (Element : in     Asis.Element;
   Control : in out Asis.Traverse_Control;
   State   : in out Info_source) is

   ----------------------------------------
   --                                    --
   -- Some tool procedures to make cool --
   -- display of the sources.           --
   --                                    --
   ----------------------------------------

   ------------------------------------------------------------------
```

```
--  Commit is called just before exiting of Pre_Source, if you look
--  well you'll see that the Push procedure pushes them on a temporary
--  stack (Tmp_Stack), that allows programmer to write his pushes in the
--  logical (lexical) order. Then, the commit pours the Tmp_Stack into
--  the main stack (Lexical_Stack) which reverses the order of the elements.
procedure Commit is
   Lex : Lexical_Node;
begin
   while not Node_Stack.Is_Empty (State.Tmp_Stack) loop
      Node_Stack.Pop (State.Tmp_Stack, Lex);
      Node_Stack.Push (State.Lexical_Stack, Lex);
   end loop;
exception
   when others =>
      Ada.Text_IO.Put_Line (" Exception raised in Commit");
      raise;
end Commit;

-----------------------------------------------------------------
-- Indent is to be called just after having Pushed an element on the lexical stack
-- It specifies that you want to see more spaces Ada.Text_IO.Put in begining of line for the last
-- lexical node you pushed. (The parameter is optional)
procedure Indent (Number_Of_Space : Positive := State.Default_Indentation_Element) is
   Lex : A_Lexical_Node := Node_Stack.Upper (State.Tmp_Stack);
begin
   Lex.Indentation := State.Current_Indentation_Reference ;
   Lex.Indentation_Reference := Lex.Indentation_Reference + Number_Of_Space;
exception
   when others =>
      Ada.Text_IO.Put_Line (" Exception raised in Indent");
      raise;
end Indent;

-----------------------------------------------------------------
-- No_Space is used in the same way as Indent, after a Push, it says that
-- there must be no space after the last element of the list.
-- It is used by things like A.B.C  A'First and so on ....
procedure No_Space is
   Lex : A_Lexical_Node := Node_Stack.Upper (State.Tmp_Stack);
begin
   if Lex = null
   then
      return;
   else
      Lex.No_Space := True;
   end if;
exception
   when others =>
      Ada.Text_IO.Put_Line (" Exception raised in No_Space");
      raise;
end No_Space;

-----------------------------------------------------------------
-- indicates that it is a return_list if the first line number of
-- the first element of the list is the same as the last line
-- number of the last element of the list.
-- We use it to deal with long list .... This way we display them
-- the same way than in the original source ...
procedure Check_If_Return_Separator (List : Asis.Element_List) is
   Lex : A_Lexical_Node := Node_Stack.Upper (State.Tmp_Stack);
begin
   if Lex = null
   then
      return;
   else
      --       Lex.Return_List :=
      --         Asis.Text.First_Line_Number (List (List'First)) /=
      --         Asis.Text.Last_Line_Number  (List (List'Last ));
      Lex.Return_List := False;
   end if;
exception
   when others =>
      Ada.Text_IO.Put_Line (" Exception raised in Return_Separator");
      raise;
end;

-----------------------------------------------------------------
-- This function is not designed to be used a lot ...
-- In fact it is used only in A_Function_Call to deal with
-- Infixed operators ...
procedure Infix is
   Lex : A_Lexical_Node := Node_Stack.Upper (State.Tmp_Stack);
begin
   if Lex = null
   then
      return;
   else
      Lex.Infixed_Operator := True;
   end if;
exception
   when others =>
      Ada.Text_IO.Put_Line (" Exception raised in Infix");
      raise;
end Infix;

-----------------------------------------------------------------
-- This function is only used in An_Operator_Symbol to know what to do,
-- i.e if the operator is to be sent or pushed and if the quotes have to
-- be Ada.Text_IO.Put or not ...
function Is_Infix return Boolean is
   Lex : A_Lexical_Node := Node_Stack.Upper (State.Lexical_Stack);
begin
   if Lex = null
   then
      return False;
   else
      return Lex.Infixed_Operator;
   end if;
exception
```

```
   when others =>
      Ada.Text_IO.Put_Line (" Exception raised in Is_Infix");
      raise;
end Is_Infix;

-----------------------------------------------------------------------
-- Push is to be used in the main procedure Pre_Source to push elements
-- in the stack to say what to do. These elements are poped by the Pass_Element
-- function.
procedure Push (A_lexem : String := "";
                A_List_Kind : List_Kinds := Not_In_A_List;
                A_Number_Of_Elements : natural := 1) is

   Tmp_Lex : Lexical_Node :=
     (Lexem => new String' (A_Lexem) ,
      List_Kind => A_List_Kind ,
      Number_Of_Elements => A_Number_Of_Elements ,
      -- The following are the default components
      First_Passed => False ,
      Indentation => State.Current_Indentation_Reference ,
      Indentation_Reference => State.Current_Indentation_Reference,
      No_Space => False,
      Return_List => False,
      Infixed_Operator => False);
begin
   Node_Stack.Push (State.Tmp_Stack, Tmp_Lex);
exception
   when others =>
      Ada.Text_IO.Put_Line (" Exception raised in Push");
      raise;
end Push;

-----------------------------------------------------------------------
-- Send is the text outAda.Text_IO.Put procedure
-- The paramater Parametre_Indentation should not be set
-- when used in procedure Pre_Source ...
procedure Send (Text : String;
                Indentation_Parameter : integer := State.Current_Indentation_Reference;
                No_Space : Boolean := False) is
   Last : Natural := Text'First; -- it is the index of last CR found
begin
   -- We don't print a final space, we'll do it after if it is
   -- allowed by the no_space parameter.

   if Text'Length = 0 then
      return;
      --  we need this for the fix for separate bodies
   end if;

   if Text'Last - Text'First + 1 > 0 and then
     Text (Text'Last) = ' '
   then
      Send (Text (Text'First .. Text'Last - 1), Indentation_Parameter, No_Space);
      State.Last_Char_Was_Space := True;
      return;
   end if;

   if State.Last_Char_Was_Space
   then
      if not No_Space
      then
         Ada.Text_IO.Put (" ");
         State.Horizontal_Position := State.Horizontal_Position + 1;
      end if;
      State.Last_Char_Was_Space := False;
   end if;

   -- Sends the text on the standard outAda.Text_IO.Put
   -- When a ASCII.CR is found it is replaced by a
   -- Ada.Text_IO.Put_Line  (which is in fact ASCII.CR & ASCII.LF),
   -- moreover the indentation is added
   for Index in Text'Range loop
      if Text (Index) = ASCII.CR
      then
         -- Let's print the indentation
         if State.Last_Char_Was_Return
         then
            for Space in 1 .. Indentation_Parameter loop
               Ada.Text_IO.Put (" ");
            end loop;
            State.Horizontal_Position := State.Horizontal_Position +
              Indentation_Parameter;
            -- no need to reset Last_Char_Was_Return to false ...
         end if;
         Ada.Text_IO.Put_Line (Text (Last .. Index - 1));
         Last := Index + 1;
         State.Last_Char_Was_Return := True;
         State.Horizontal_Position := 0;
         State.Vertical_Position := State.Vertical_Position + 1;
      end if;
   end loop;

   if Last in Text'Range
   then
      -- Let's print the indentation
      if State.Last_Char_Was_Return
      then
         for Space in 1 .. Indentation_Parameter loop
            Ada.Text_IO.Put (" ");
         end loop;
         State.Horizontal_Position := State.Horizontal_Position +
           Indentation_Parameter;
         State.Last_Char_Was_Return := False;
      end if;
      Ada.Text_IO.Put (Text (Last .. Text'Last));
      State.Horizontal_Position := State.Horizontal_Position +
        Text'Last - Last + 1;
      if State.Horizontal_Position > State.Max_Line_Length
      then
```

```
         Ada.Text_IO.New_Line;
         State.Last_Char_Was_Return := True;
         State.Horizontal_Position := 0;
         State.Vertical_Position := State.Vertical_Position + 1;
      end if;
   end if;

exception
   when others =>
      Ada.Text_IO.Put_Line (" Exception raised in Send");
      raise;
end Send;

-----------------------------------------------------------------------
procedure Pass_Element;
procedure Pass_Element_1;


-- Pass_Element is called each time we process an element, it counts them
-- and displays the lexem when needed

procedure Pass_Element is

   -- Processes any element of the stack with a
   -- number of elements equal to zero and
   -- decreases of one the first non null it finds.
   -- Eventualy sets First_Passed to False and sends
   -- the corresponding separator if needed ...
   Up : A_Lexical_Node := Node_Stack.Upper (State.Lexical_Stack);
   Trash : Lexical_Node;

begin
   -- In that mode, it's not possible to handle the comments :
   -- for example what difference could be done between those situations :
   --
   -- procedure Hello  -- comment
   -- is begin  .....
   --
   -- procedure Hello is
   -- -- comment
   -- begin
   --   ...
   --

   if Up = null
   then
      -- This happens when the stack is empty
      return;
   end if;
   State.Current_Indentation_Reference := Up.Indentation_Reference;
   -- If there is NO element in the list :
   --  First_Passed = True and there is no point in printing
   --                       the separator (no remaining element)
   --  First_Passed = False and there is no point in printing
   --                        an opening parenthesis (no element in list)
   if Up.List_Kind = Is_Comma_Range_List and
     Up.First_Passed
   then
      Send ("range <> ");
   end if;

   if Up.Number_Of_Elements /= 0
   then
      if Up.First_Passed and
        Up.List_Kind /= Not_In_A_List
      then
         -- Let's print the separator because there is an element after
         if Up.Return_List
         then
            Send (Separator (Up.List_Kind).all & ASCII.CR);
         else
            Send (Separator (Up.List_Kind).all & " ");
         end if;
      elsif Up.List_Kind in Parenthesized_List
      then
         -- Let's print the opening parenthesis
         Send ("(");
         Up.Indentation_Reference := State.Horizontal_Position + 1;
      end if;
      -- Now we'll see if we have to print a closing parenthesis
   elsif Up.First_Passed and
     Up.List_Kind in Parenthesized_List
   then
      -- Let's print the opening parenthesis
      Send (") ");
   end if;

   -- OK then now we are sure we have passed the first one ...
   Up.First_Passed := True;

   -- Now, let's see if we have to print the after string ...
   if Up.Number_Of_Elements = 0
   then
      -- If so we print the lexem that was designed for that purpose.
      if Up.Lexem.all /= ""
      then
         Send (Up.Lexem.all ,
               Up.Indentation ,
               Up.No_Space);
      end if;
      -- And we get rid of it
      Node_Stack.Pop (State.Lexical_Stack, Trash);
      -- Beware !! after that Up is not available !!
      -- (in fact it is but only bad guys would use it ..)
      Up := null;

      if Current_Unit /= null then
         Current_Unit.Is_In_MH := False;
      end if;
```

```
      -- as the counter was 0, we pass another one ...
      Pass_Element_1;
   else
      Up.Number_Of_Elements := Up.Number_Of_Elements - 1;
   end if;
exception
   when others =>
      Ada.Text_IO.Put_Line (" Exception raised in Pass_Element");
      raise;
end Pass_Element;

procedure Pass_Element_1 is
begin
   Pass_Element;
end Pass_Element_1;

-- The following procedures are 'user defined' they are here only
-- to make things simplier ....
function Function_Call_Operator return String is
   Op : String := To_String (Asis.Expressions.Name_Image (Element));
begin
   return Op (Op'First + 1 .. Op'Last - 1) & " ";
end Function_Call_Operator;


function Designated_Probes(Kind : Measurement_T; Current_Block : V_String;
                           Is_Indent : Boolean := True;
                           Elem : Asis.Element := Element)
                           return V_String;

-- Helps displaying labels before statements ..
-- Element should be a Statement ..
-- if not, an Inapropriate_Element is raised
procedure Send_Label (Text : String) is
   Nb_Labels : Natural := Count (Asis.Statements.Label_Names (Element));
   Label_Probes : V_String := Null_Str;
begin
   if Nb_Labels > 0
   then
      declare
         Tmp_Label_Names : Asis.Defining_Name_List
           := Asis.Statements.Label_Names(Element);
      begin
         for Index in 1 .. Nb_Labels loop
            Tmp_Label_Name := To_V(Asis.Declarations.Defining_Name_Image
                                     (Tmp_Label_Names(Index)));
            Label_Probes := Label_Probes &
              Designated_Probes(LABEL_PASSAGE,
                                Current_Unit_Name & "." &
                                  Tmp_Label_Name);
         end loop;
      end;

      Send (To_S(Label_Probes) & "<< ");
      for Index in 2 .. Nb_Labels
      loop
         Push (">>" & ASCII.CR & "<< ");
      end loop;
      Push (">>" & ASCII.CR & Text);
   else
      Send (Text);
   end if;
end Send_Label;

-- Element is a global variable here .. :)
-- use the parameter Text when you need to insert a 'tagged' keyword
-- in the trait string.
function Trait_String (Text : String := "")
                       return String is
begin
   case Asis.Elements.Trait_Kind (Element) is
   when Not_A_Trait =>
      --   return "<<Node Not_A_Trait>> ";
      return ""; -- because i'm fed up with the component bug ...
   when An_Ordinary_Trait =>
      return Text;
   when An_Aliased_Trait =>
      return "aliased ";
   when An_Access_Definition_Trait =>
      return "access ";
   when A_Reverse_Trait =>
      return "reverse ";
   when A_Private_Trait =>
      return Text & "private ";
   when A_Limited_Trait =>
      return Text & "limited ";
   when A_Limited_Private_Trait =>
      return Text & "limited private ";
   when An_Abstract_Trait =>
      return "abstract " & Text;
   when An_Abstract_Private_Trait =>
      return "abstract " & Text & "private ";
   when An_Abstract_Limited_Trait =>
      return "abstract " & Text & "limited ";
   when An_Abstract_Limited_Private_Trait =>
      return "abstract " & Text & "limited private ";
      -- --|A2005 start
   when A_Null_Exclusion_Trait =>
      return "not null";
      -- --|A2005 end
   end case;
end Trait_String;



--    function Is_All(Kind : Measurement_T) return Boolean is
--       use Status_List;
```

```
--       Tmp_List : Status_List.Node_Link := Spec_Reader.Spec_Data(Kind);
--       Tmp_Status : Measurement_Status_Link;
--       Iter : Iterator_T;
--    begin
--       Set_List(Iter, Tmp_List);
--       while not Is_End(Iter) loop
-- -- Put_Line("is_all ?" & Measurement_T'IMAGE(Kind));
--   Tmp_Status := Get_Now(Iter);
--   if Tmp_Status.Selection = All_Place then
-- --    Put_Line("true!");
--      return True;
--   end if;
-- -- Put_Line("next." & Selection_Kind'IMAGE(Tmp_Status.Selection));
--   Set_Next(Iter);
--       end loop;
--       return False;
--    end Is_All;


function Is_Designated(S_List : Status_List.Node_Link;
                       Current_Block : V_String) return Boolean is
   use Status_List;
   Iter : Iterator_T;
   Tmp_Status : Measurement_Status_Link;
   Result : Boolean := False;
begin
   --      Put_Line("==is designated==");
   Set_List(Iter, S_List);
   while not Is_End(Iter) loop

      Tmp_Status := Get_Now(Iter);

      if Tmp_Status.Selection = All_Place then
         --    Put_Line("all ...");
         Result := True;

      elsif Match_Designation(Tmp_Status.Specified_Block_Name, Current_Block) then
         --      Put_Line("wildcard ...(+)");
         --      Put_Line("spec: " & To_S(Tmp_Status.Specified_Block_Name));
         --      Put_Line("current: " & To_S(Current_Block));
         Result := True;
      elsif Match_Designation(Tmp_Status.Specified_Block_Name, "-" & Current_Block) then
         --      Put_Line("wildcard ...(-)");
         --      Put_Line("spec: " & To_S(Tmp_Status.Specified_Block_Name));
         --      Put_Line("current: " & To_S(Current_Block));
         Result := False;
         --   else
         --      Put_Line("wildcard ...(none)");
         --      Put_Line("spec: " & To_S(Tmp_Status.Specified_Block_Name));
         --      Put_Line("current: " & To_S("-" & Current_Block));

      end if;
      Set_Next(Iter);
   end loop;
   return Result;
end Is_Designated;

function Cond_Expression(C_List : Condition_List.Node_Link) return V_String is
   use Condition_List;
   Iter : Iterator_T;
   Temp_Cond : Measurement_Condition_Link;
   Result : V_String := Null_Str;
begin
   Set_List(Iter, C_List);
   while not Is_End(Iter) loop

      Temp_Cond := Get_Now(Iter);

      if Temp_Cond.Selection = All_Place then
         Result := Temp_Cond.Condition_Expression;

      elsif Match_Designation(Temp_Cond.Specified_Block_Name, To_V(Current_Unit_Name)) then
         Result := Temp_Cond.Condition_Expression;
      end if;

      if Match_Designation(Temp_Cond.Specified_Block_Name, "-" & To_V(Current_Unit_Name)) then
         Result := Null_Str;
      end if;
      Set_Next(Iter);
   end loop;
   return Result;
end Cond_Expression;


Is_Needed_Get_Cdt : Boolean := False;

function Parent_ID(Child : Gela_Ids.Id) return Gela_Ids.Id is
   Child_Unit_Item : Unit_Item_Link := Unit_Item_List.Search_Node
     (Unit_Items, Child);
begin
   if Child_Unit_Item = null or else
     Child_Unit_Item.Parent = null then
      return Gela_Ids.Nil_Id;
   end if;
   return Child_Unit_Item.Parent.Element_ID;
end Parent_ID;

Parent_Task_ID : V_String;

Is_Parent_Task : Boolean := False;

Tmp_Guard : Asis.Element;

-- probe となる procedure or function call を作る
function Probe_Call(Kind : Measurement_T;
                    Elem : Asis.Element;
                    Qid : Query_ID;
                    P_List : Parameter_List.Node_Link := null;
                    C_List : Condition_List.Node_Link := null;
```

```
                 Var_Status : Measurement_Component_Link := null;
                 Genuine_Name : V_String := Null_Str;
                 Is_MH : Boolean := False
                )
                 return String is
   use Parameter_List;
   Result : V_String;
   Iter : Iterator_T;
   First_Param : Boolean := True;

   Cond_Exp : V_String := Null_Str;

   Proxy_Name : V_String;

   procedure Make_Proxy_For_Cond is
   begin
      Result := Result & "function " & Proxy_Name &
        " return Boolean is" & ASCII.CR;
      Result := Result & "begin" & ASCII.CR;
      Result := Result & "if " & Cond_Exp & " then" & ASCII.CR;
      Result := Result & "declare" & ASCII.CR;

      Result := Result & "  " & Probe_Call(Kind, Elem, Qid, P_List,  null,
                                           Var_Status, Genuine_Name,
                                           Is_Mh);
      Result := Result & "begin null; end;" & ASCII.CR;
      Result := Result & "end if;" & ASCII.CR;
      Result := Result & "return True;" & ASCII.CR;
      Result := Result & "end " & Proxy_Name & ";" & ASCII.CR;
   end Make_Proxy_For_Cond;


begin
   Cond_Exp := Cond_Expression(C_List);

   if Place_Kind_Of_T(Kind) = IN_DECLARATION then
      if Cond_Exp /= Null_Str then
         Proxy_Name := To_V(Unique_Identifier);
         Make_Proxy_For_Cond;
      end if;
      Result := Result & To_V(Unique_Identifier & " : Boolean := ");
   else
      if Cond_Exp /= Null_Str then
         Result := Result & "if " & Cond_Exp & " then " & ASCII.CR;
      end if;
   end if;
   if Cond_Exp /= Null_Str and then
     Place_Kind_Of_T(Kind) = IN_DECLARATION then
      Result := Result & Proxy_Name & ";" & ASCII.CR;
   else

      if Kind in ASSIGNED_VARIABLE..REFERED_VARIABLE then
         if Var_Status = null then
            Put_Line("internal: variable measurement must have Var_Status.");
            raise Program_Error;
         else
            Result := Result & Var_Status.Call_Name;
         end if;
      else
         if Is_MH then
            Result := Result & "MHP_";
         end if;
         Result := Result & Measurement_T'IMAGE(Kind);
         if Qid /= 0 then
            declare
               Qid_Image : String := Query_Id'IMAGE(Qid);
            begin
               Result := Result & Qid_Image(Qid_Image'FIRST + 1 ..
                                            Qid_Image'LAST);
            end;
         end if;
      end if;
      if P_List /= null then
         Result := Result & "(";
         Set_List(Iter, P_List);
         while not Is_End(Iter) loop
            if not First_Param then
               Result := Result & ", ";
            end if;
            First_Param := False;
            case Get_Now(Iter) is
            when NOW =>
               Result := Result & "Clock";
            when BLOCK_NAME =>
               Result := Result & """" & Current_Unit_Name & """";
            when CALLEE_TASK =>
               declare
                  --     Entry_Name : Asis.Expression := Asis.Statements.Called_Name(Element);
                  Entry_Name : Asis.Expression := Get_Called_Name(Elem);
               begin
                  case Asis.Elements.Expression_Kind(Entry_Name) is
                  when An_Identifier =>
                     Result := Result & "Current_Task";
                  when A_Selected_Component =>
                     Result := Result &
                       Eliminate_Space(To_String(Asis.Text.Element_Image(Asis.Expressions.Prefix(Entry_Name)))) & "'Identity";
                  when An_Indexed_Component =>
                     Result := Result &
                       Eliminate_Space(To_String(Asis.Text.Element_Image(Asis.Expressions.Prefix(Asis.Expressions.Prefix(Entry_Name))))) & "'Identity";
                  when others =>
                     Put_Line("invalid entry name.");
                     raise Program_Error;
                  end case;
               end;
            when ENTRY_NAME | SUBPROGRAM_NAME =>
               declare
                  --     Entry_Name : Asis.Expression := Asis.Statements.Called_Name(Element);
                  Entry_Name : Asis.Expression := Get_Called_Name(Elem);
               begin
```

```
      case Asis.Elements.Expression_Kind(Entry_Name) is
      when An_Identifier =>

         Result := Result & """" & To_String(Asis.Expressions.Name_Image(Entry_Name)) & """";
      when A_Selected_Component =>
         Result := Result & """" & To_String(Asis.Expressions.Name_Image(Asis.Expressions.Selector(Entry_Name))) & """";
      when An_Indexed_Component =>
         null;
      when others =>
         Put_Line("invalid entry name.");
         raise Program_Error;
      end case;
   end;
when CALLEE_PROTECTED_ID =>
   declare
      Callee_Unit : Asis.Declaration :=
        Asis.Elements.Enclosing_Element(
          Asis.Elements.Enclosing_Element
            (Asis.Statements.Corresponding_Called_Entity
               (Elem)));
   begin
      case Asis.Elements.Declaration_Kind(Callee_Unit) is
      when A_Single_Protected_Declaration =>
         Result := Result
           & To_String
           (Asis.Declarations.Defining_Name_Image
              (First_Element (Asis.Declarations.Names
               (Callee_Unit))))
           & "_ID";
      when A_Protected_Type_Declaration =>
         Result := Result
           & To_String
           (Asis.Declarations.Defining_Name_Image
              (First_Element (Asis.Declarations.Names
               (Callee_Unit))))
           & ".ID";
      when others =>
         Put_Line("Callee_Protected_ID is for only protected objects.");
         raise Program_Error;
      end case;
   end;
when FULL_SUBPROGRAM_NAME =>
   declare
      Called_Entity : Asis.Declaration;
   begin
      case Asis.Elements.Element_Kind(Elem) is
      when A_Statement =>
         Called_Entity := Asis.Statements.Corresponding_Called_Entity(Elem);
      when An_Expression =>
         Called_Entity := Asis.Expressions.Corresponding_Called_Function(Elem);
      when others =>
         Put_Line("Error in FULL_SUBPROGRAM_NAME.");
         raise Program_Error;
      end case;
      --     Result := Result &
      --       Unit_Name_List.Search_Node
      --       (Unit_Names, Gela_Ids.Create_ID(Called_entity));
      Result := Result & To_V(""""  & Declaration_Full_Name(Called_Entity) & """");
   end;
when IS_PROTECTED_BLOCK =>
   declare
      Enclosing_Unit : Asis.Declaration :=
        Asis.Elements.Enclosing_Element
          (Asis.Elements.Enclosing_Element
               (Elem));
   begin
      case Asis.Elements.Declaration_Kind(Enclosing_Unit) is
      when A_Single_Protected_Declaration |
           A_Protected_Type_Declaration =>
         Result := Result & "True";
      when others =>
         Result := Result & "False";
      end case;
   end;
when THIS_UNIT_CLASS =>
   Result := Result
     & Unit_Class'IMAGE(Current_Unit.This_Unit_Class);

when CALLER_TASK =>
   Result := Result
     & To_String(Asis.Expressions.Name_Image
     (Asis.Statements.Accept_Entry_Direct_Name
        (Elem))) & "'Caller";
when ABORT_TASKS =>
   declare
      Aborted_Tasks : Asis.Expression_List
        := Asis.Statements.Aborted_Tasks(Elem);
      First_Flag : Boolean := True;
   begin
      Result := Result & "Task_ID_List'(";
      for I in Aborted_Tasks'RANGE loop
         if Aborted_Tasks'LENGTH = 1 then
            Result := Result & "1=> ";
         end if;
         if not First_Flag then
            Result := Result & ", ";
         else
            First_Flag := False;
         end if;
         Result := Result &
           Eliminate_Space
           (To_String
              (Asis.Text.Element_Image(Aborted_Tasks(I)))) &
           "'Identity";
      end loop;
      Result := Result & ")";
   end;
when CDT_OF_TASK =>
   --    if Call_State_List."="(Current_Unit.Is_Calling_List, null) then
```

```
   --       Result := Result & "Null_CDT";
   --    else
   --     Result := Result & "(";
   declare
      Tmp_Result : V_String := Null_Str;
      Iter : Call_State_List.A_Node
        := Current_Unit.Is_Calling_List;
      First_Flag : Boolean := True;
      Entry_Item : Unit_Item_Link;
      Counter : Integer := 1;
   begin
      while Call_State_List."/="(Iter, null) loop
         if Iter.Object.Has_CDT
           and Iter.Object.State = CDT_Flag then
            if CDT_Flag /= DECLARED
              or Gela_Ids.Is_Equal(Parent_ID(Iter.Index),
                                   CDT_Subject_Task)
              or Gela_Ids.Is_Equal(Iter.Object.Var_Decl,
                                   CDT_Subject_Task) then
               Is_Needed_Get_Cdt := True;
               if not First_Flag then
                  Tmp_Result := Tmp_Result & ", " & ASCII.CR;
               else
                  First_Flag := False;
               end if;

               Entry_Item :=
                 Unit_Item_List.Search_Node
                   (Unit_Items,
                    Iter.Index);

               if Iter.Object.Task_Name /= Null_Str then
                  Tmp_Result := Tmp_Result & Integer'IMAGE(Counter) & "=> (" &
                    Iter.Object.Task_Name & "'Identity, ";
               else
                  if Entry_Item /= null and then
                    Entry_Item.Parent /= null then
                     Tmp_Result := Tmp_Result & Integer'IMAGE(Counter) & "=> (" &
                       Entry_Item.Parent.Name & "'identity, ";
                  else
                     Put_Line("Illegal CDT_OF_TASK.");
                     raise Program_Error;
                  end if;
               end if;
               Counter := Counter + 1;

               Tmp_Result := Tmp_Result & """" & Entry_Item.Name
                 & """ & ("
                 & Natural'IMAGE(Length(Entry_Item.Name) +1)
                 & ".." & Natural'IMAGE(Max_Entry_Name) &
                 " => ascii.nul), " &
                 Natural'IMAGE(Length(Entry_Item.Name))
                 & ")";
            end if;

         end if;
         Iter := Iter.Next;
      end loop;
      if Tmp_Result /= Null_Str then
         Result := Result & "(" & Tmp_Result & ")";
      else
         Result := Result & "Null_CDT";
      end if;
   end;
   --  Result := Result & ")";
when CDT_OF_TASK_TYPE =>
   --    if Call_State_List."="(Current_Unit.Is_Calling_List,
   --   null) then
   --       Result := Result & "Null_CDT_For_TT";
   --    else
   --       Result := Result & "(";
   declare
      Tmp_Result : V_String := Null_Str;
      Iter : Call_State_List.A_Node
        := Current_Unit.Is_Calling_List;
      First_Flag : Boolean := True;
      Entry_Item : Unit_Item_Link;
      Counter : Integer := 1;
   begin
      while Call_State_List."/="(Iter, null) loop
         --Put_Line("== not null!");
         --  Put_Line(Boolean'IMAGE(Iter.Object.Has_CDT_For_TT));
         --  Put_Line(Call_State_T'IMAGE(Iter.Object.State));
         if Iter.Object.Has_CDT_For_TT
           and Iter.Object.State = CDT_Flag then
            if CDT_Flag /= DECLARED
              or Gela_Ids.Is_Equal(Parent_ID(Iter.Index),
                                   CDT_Subject_Task)
              or Gela_Ids.Is_Equal(Iter.Object.Var_Decl,
                                   CDT_Subject_Task) then
               Is_Needed_Get_Cdt := True;
               if not First_Flag then
                  Tmp_Result := Tmp_Result & ", ";
               else
                  First_Flag := False;
               end if;
               Entry_Item :=
                 Unit_Item_List.Search_Node
                   (Unit_Items,
                    Iter.Index);
               if Entry_Item /= null and then
                 Entry_Item.Parent /= null then
                  Tmp_Result := Tmp_Result &
                    Integer'IMAGE(Counter) & "=> (" &
                    Unit_ID'IMAGE(Entry_Item.Parent.ID) & ", ";
               else
                  Put_Line("Illegal CDT_OF_TASK_TYPE.");
                  raise Program_Error;
               end if;
```

```
                        Counter := Counter + 1;
                        Tmp_Result := Tmp_Result & """" & Entry_Item.Name
                          & """ & ("
                          & Natural'IMAGE(Length(Entry_Item.Name) +1)
                          & ".." & Natural'IMAGE(Max_Entry_Name) &
                          " => ascii.nul), " &
                          Natural'IMAGE(Length(Entry_Item.Name)) &
                          ")";
                     end if;
                  end if;
                  Iter := Iter.Next;
               end loop;
               if Tmp_Result /= Null_Str then
                  Result := Result & "(" & Tmp_Result & ")";
               else
                  Result := Result & "Null_CDT_For_TT";
               end if;
            end;
            --  Result := Result & ")";
            --  end if;
         when UNIT_ID_OF_TASK =>
            -- task type 以外なら 0
            if Current_Unit.Decl_Kind = A_Task_Type_Declaration then
               Result := Result & Unit_ID'IMAGE(Current_Unit.ID);
            else
               Result := Result & Unit_ID'IMAGE(0);
            end if;
         when PARENT_TASK =>
            Parent_Task_ID := To_V(Unique_Identifier);
            Is_Parent_Task := True;
            Result := Result & Parent_Task_ID;
         when TARGET =>
            if Genuine_Name = Null_Str then
               Put_Line("internal: genuine_name must not be null_str.");
               raise Program_Error;
            end if;
            if Var_Status.Conversion_Type /= Null_Str then
               Result := Result & Var_Status.Conversion_Type & "(" &
                 Genuine_Name & ")";
            else
               Result := Result & Genuine_Name;
            end if;
         when LINE_NUMBER =>
            Result := Result & Asis.Text.Line_Number'IMAGE
              (Asis.Text.First_Line_Number(Elem));
         when COMPILATION_UNIT_NAME =>
            Result := Result & """" & To_String
              (Asis.Compilation_Units.Unit_Full_Name
                 (Asis.Elements.Enclosing_Compilation_Unit(Elem)))
              & """";
         when MY_TASK =>
            Result := Result & "Current_Task";
         when MY_TASK_IMAGE =>
            Result := Result & "Image(Current_Task)";
         when LABEL_NAME =>
            Result := Result & """" & To_S(Tmp_Label_Name) & """";
         when STATEMENT_NAME =>
            Result := Result & """" &
              To_V(Asis.Declarations.Defining_Name_Image
                   (Asis.Statements.Statement_Identifier(Elem))) & """";
         when IS_OPEN_ACCEPT =>
            if Asis.Elements.Is_Nil(Tmp_Guard) then
               Result := Result & "True";
            else
               Result := Result &
                 To_String(Asis.Text.Element_Image(Tmp_Guard));
            end if;
         when others =>
            null;
         end case;
         Set_Next(Iter);
      end loop;
      Result := Result & ")";
   end if;
   Result := Result & ";" & ASCII.CR;
   if Cond_Exp /= Null_Str then
      Result := Result & "end if;" & ASCII.CR;
   end if;
end if;
return To_S(Result);
end Probe_Call;


function Designated_Probes(Kind : Measurement_T; Current_Block : V_String;
                           Is_Indent : Boolean := True;
                           Elem : Asis.Element := Element)
                           return V_String is


   function Condition_Traverse(S_List : Status_List.Node_Link) return Boolean
   is
      use Status_List;
      Iter : Iterator_T;
   begin

      Set_List(Iter, S_List);
      while not Is_End(Iter) loop
         if Special_Condition(Get_Now(Iter).all, Elem, To_V(Current_Unit_Name)) then

            return True;
         end if;
         Set_Next(Iter);
      end loop;
      return False;
   end Condition_Traverse;


   use Component_List;
   use Measurement_Analyzer;
```

```
   Result : V_String := Null_Str;
   Iter : Iterator_T;
   Tmp_Component : Measurement_Component_Link;

   Now_Unit : Unit_Item_Link := Current_Unit;
begin

   Is_Needed_Get_Cdt := False;
   Set_List(Iter, Spec_Reader.Spec_Data(Kind));
   while not Is_End(Iter) loop
      Tmp_Component := Get_Now(Iter);
      if Is_Designated(Tmp_Component.Status, Current_Block) and then
        Condition_Traverse(Tmp_Component.Status) then
         if Is_Indent then
            for I in 1..State.Default_Indentation_Element loop
               Result := Result & " ";
            end loop;
         end if;
         Result := Result & Probe_Call(Kind, Elem, Tmp_Component.Qid,
                                       Tmp_Component.Parameters,
                                       Tmp_Component.Condition);
      end if;
      Set_Next(Iter);
   end loop;

   while Now_Unit /= null loop
      if Now_Unit.MH_Data_Link /= null and then
        Now_Unit.MH_Data_Link(Kind) /= null and then
        (Now_Unit.MH_Data_Link(Kind).Status = NORMAL or Now_Unit = Current_Unit) then
         if Is_Indent then
            for I in 1..State.Default_Indentation_Element loop
               Result := Result & " ";
            end loop;
         end if;
         Result := Result &
           Probe_Call(Kind, Elem, 0,
                      Now_Unit.MH_Data_Link(Kind).Parameters, null,
                      null, Null_Str, True);
         exit;
      else
         Now_Unit := Now_Unit.Parent;
      end if;
   end loop;

   if Kind = GET_CDT and Is_Needed_Get_Cdt = False then
      return Null_Str;
   end if;
   return Result;
end Designated_Probes;


function Begin_Label(Has_Declaration : Boolean := True) return String is
   Result : V_String;
begin
   if Has_Declaration then
      case Asis.Elements.Declaration_Kind(Element) is
      when A_Task_Body_Declaration =>
         Result := Result & Designated_Probes(TASK_ACTIVATION_COMPLETION, To_V(Current_Unit_Name));
      when A_Package_Body_Declaration =>
         Result := Result & Designated_Probes(LIBRARY_PACKAGE_ELABORATION_COMPLETION, To_V(Current_Unit_Name));
      when others =>
         Result := Result & Designated_Probes(BLOCK_ELABORATION_COMPLETION, To_V(Current_Unit_Name));
      end case;

      Result := Result & Current_Unit.MH_Body_Str;
      -- Result := Result & Measurement_Analyzer.Analyze(Element);
   end if;

   --       if Is_All(BLOCK_ELABORATION_COMPLETION) and Has_Declaration then
   --   for I in 1..State.Default_Indentation_Element loop
   --      Result := Result & " ";
   --   end loop;
   --   Result := Result & Probe_Call(BLOCK_ELABORATION_COMPLETION);
   --       end if;
   Result := Result & "begin" & ASCII.CR;

   Result := Result & Designated_Probes(BLOCK_EXECUTION_START, To_V(Current_Unit_Name));
   --       if Is_All(BLOCK_EXECUTION_START) then
   --   for I in 1..State.Default_Indentation_Element loop
   --      Result := Result & " ";
   --   end loop;
   --   Result := Result & Probe_Call(BLOCK_EXECUTION_START);
   --       end if;
   return To_S(Result);
end Begin_Label;

function MH_Spec return String is
begin
   return Measurement_Analyzer.Analyze(Element);
end MH_Spec;


function Is_Label(Has_MH : Boolean := False) return String is
   Result : V_String := To_V("is" & ASCII.CR);
begin
   if Has_MH then
      Result := Result & MH_Spec;
   end if;
   case Asis.Elements.Declaration_Kind(Element) is
   when A_Task_Body_Declaration =>
      Result := Result & Designated_Probes(TASK_ACTIVATION_START, To_V(Current_Unit_Name));
      Result := Result & Designated_Probes(GET_CDT, To_V(Current_Unit_Name));
   when A_Package_Declaration =>
      Result := Result & Designated_Probes(LIBRARY_PACKAGE_ELABORATION_START, To_V(Current_Unit_Name));
   when others =>

      Result := Result & Designated_Probes(BLOCK_ELABORATION_START, To_V(Current_Unit_Name));
      -- environment task に限っては、main procedure が CDT を渡す
      if Current_Unit.Parent = Environment_Task then
```

```
         Result := Result & Designated_Probes(GET_CDT, To_V(Current_Unit_Name));
      end if;
   end case;
   --       if Is_All(BLOCK_ELABORATION_START) then
   --   for I in 1..State.Default_Indentation_Element loop
   --      Result := Result & " ";
   --   end loop;
   --   Result := Result & Probe_Call(BLOCK_ELABORATION_START);
   --       end if;
   return To_S(Result);
end Is_Label;


function Declare_Label return String is
   Result : V_String := To_V("declare" & ASCII.CR);
begin
   Result := Result & Designated_Probes(BLOCK_ELABORATION_START, To_V(Current_Unit_Name));
   return To_S(Result);
end Declare_Label;


function End_Label return String is
   Result : V_String;
begin
   Result := Result & Designated_Probes(BLOCK_EXECUTION_COMPLETION, To_V(Current_Unit_Name));
   --       if Is_All(BLOCK_EXECUTION_COMPLETION) then
   --   for I in 1..State.Default_Indentation_Element loop
   --      Result := Result & " ";
   --   end loop;
   --   Result := Result & Probe_Call(BLOCK_EXECUTION_COMPLETION);
   --       end if;
   Result := Result & "end ";
   return To_S(Result);
end End_Label;


function Return_Label return String is
   Result : V_String;
begin
   Result := Result & Designated_Probes(BLOCK_EXECUTION_COMPLETION, To_V(Current_Unit_Name));
   Result := Result & "return ";
   return To_S(Result);
end Return_Label;

function Exception_Label return String is
   Result : V_String;
begin
   Result := Result & Designated_Probes(BLOCK_EXECUTION_COMPLETION, To_V(Current_Unit_Name));
   --       if Is_All(BLOCK_EXECUTION_COMPLETION) then
   --   for I in 1..State.Default_Indentation_Element loop
   --      Result := Result & " ";
   --   end loop;
   --   Result := Result & Probe_Call(BLOCK_EXECUTION_COMPLETION);
   --       end if;
   Result := Result & "exception" & ASCII.CR;
   return To_S(Result);
end Exception_Label;

function Do_Label return String is
   Result : V_String := To_V("do" & ASCII.CR);
begin
   Result := Result &
     Designated_Probes(RENDEZVOUS_START, To_V(Current_Unit_Name));
   return To_S(Result);
end Do_Label;

function End_Label_In_Accept return String is
   Result : V_String;
begin
   Result := Designated_Probes(RENDEZVOUS_END, To_V(Current_Unit_Name));
   Result := Result & "end ";
   return To_S(Result);
end End_Label_In_Accept;

function Exception_Label_In_Accept return String is
   Result : V_String;
begin
   Result := Result & Designated_Probes(RENDEZVOUS_END, To_V(Current_Unit_Name));
   Result := Result & "exception" & ASCII.CR;
   return To_S(Result);
end Exception_Label_In_Accept;

function Do_Or_No_Label return String is
   Result, Start_Result, End_Result : V_String;
begin
   Start_Result := Designated_Probes(RENDEZVOUS_START, To_V(Current_Unit_Name));
   End_Result := Designated_Probes(RENDEZVOUS_END, To_V(Current_Unit_Name));
   if Start_Result /= Null_Str or End_Result /= Null_Str then
      Result := To_V(String'("do ")) & Start_Result & End_Result
        & To_V(String'("end;" & ASCII.CR));
   else
      Result := To_V(";" & ASCII.CR);
   end if;
   return To_S(Result);
end Do_Or_No_Label;

function Abort_Label return String is
   Result : V_String;
begin
   Result := Result
     & Designated_Probes(ABORT_START, To_V(Current_Unit_Name));
   Result := Result & "abort ";
   return To_S(Result);
end Abort_Label;

function Statement_Identifier_Label return String is
   Result : V_String := Null_Str;
begin
   Result := Result
```

```
     & Designated_Probes(STATEMENT_EXECUTION_START,
     To_V(Current_Unit_Name) & "." &
       To_V(Asis.Declarations.Defining_Name_Image
       (Asis.Statements.Statement_Identifier(Element))));
   return To_S(Result);
end Statement_Identifier_Label;

function Statement_Completion_Label return String is
   Result : V_String := Null_Str;
begin
   Result := Result
     & Designated_Probes(STATEMENT_EXECUTION_COMPLETION,
     To_V(Current_Unit_Name) & "." &
       To_V(Asis.Declarations.Defining_Name_Image
       (Asis.Statements.Statement_Identifier(Element))));
   return To_S(Result);
end Statement_Completion_Label;


function Declared_CDT(Elem : Asis.Element) return String is
   Result : V_String := Null_Str;
begin
   CDT_Flag := DECLARED;
   Result := Result & Designated_Probes(GET_CDT, To_V(Current_Unit_Name));
   CDT_Flag := CHECKED;
   return To_S(Result);
end Declared_CDT;


function Is_Protected(Element : Asis.Declaration) return Boolean is
   Enc_Element : Asis.Element := Asis.Elements.Enclosing_Element(Element);
begin
   if Is_Here(Enc_Element) then
      Enc_Element := Asis.Elements.Enclosing_Element(Enc_Element);
   else
      return False;
   end if;
   --      Put_Line(To_String(Asis.Elements.Debug_Image(Enc_Element)));
   if Is_Here(Enc_Element) and then Asis.Elements.Element_Kind(Enc_Element)
   --  = A_Definition and then
   --  Asis.Elements.Definition_Kind(Element)
   --  = A_Protected_Definition then
     = A_Declaration and then
     (Asis.Elements.Declaration_Kind(Enc_Element)
      = A_Protected_Type_Declaration or
        Asis.Elements.Declaration_Kind(Enc_Element)
      = A_Single_Protected_Declaration) then
      return True;
   end if;
   return False;
end Is_Protected;


function Is_Procedure(Element : Asis.Statement) return Boolean is
   Decl_Element : Asis.Element := Asis.Statements.Corresponding_Called_Entity(Element);
begin
   if Is_Here(Decl_Element) and then Asis.Elements.Element_Kind(Decl_Element)
     = A_Declaration and then
     (Asis.Elements.Declaration_Kind(Decl_Element)
      = A_Procedure_Declaration or
        Asis.Elements.Declaration_Kind(Decl_Element)
      = A_Procedure_Body_Declaration) then
      return True;
   end if;
   return False;
end Is_Procedure;


function Is_Protected_With_ID(Element : Asis.Declaration) return Boolean is
   Enc_Element : Asis.Element := Asis.Elements.Enclosing_Element(Element);
begin
   if Is_Here(Enc_Element) and then Asis.Elements.Element_Kind(Element) =
     A_Declaration and then
     (Asis.Elements.Declaration_Kind(Element)
      = A_Protected_Type_Declaration or
        Asis.Elements.Declaration_Kind(Element)
      = A_Single_Protected_Declaration) and then
     Protected_With_Id_List.Search_Node
       (Pid_Flags, Gela_Ids.Create_ID(Enc_Element)) then
      return True;
   end if;
   return False;
end Is_Protected_With_ID;

function Protected_Call_Ident return String is
begin
   if Protected_With_Id_List.Search_Node
     (Conv_Flags, Gela_Ids.Create_ID(Element)) then
      Protected_With_Id_List.Delete_Node
        (Conv_Flags, Gela_Ids.Create_ID(Element));
      return "Self.";
   else
      return "";
   end if;
end Protected_Call_Ident;


procedure Set_Conv_Flag is
   Callee_Name : Asis.Expression := Get_Called_Name(Element);
begin
   if Is_Protected_With_ID(Element) then
      case Asis.Elements.Expression_Kind(Callee_Name) is
      when An_Identifier =>
         Protected_With_ID_List.Set_Node(Conv_Flags, Gela_Ids.Create_ID(Callee_Name), True);
      when A_Selected_Component =>
         Protected_With_ID_List.Set_Node(Conv_Flags, Gela_Ids.Create_ID(Asis.Expressions.Selector(Callee_Name)), True);
      when An_Indexed_Component =>
         null;
```

```
      when others =>
         Put_Line("invalid entry name.");
         raise Program_Error;
      end case;
   end if;
end Set_Conv_Flag;



function Is_Attach return Boolean is
   use Specified_Target_List;
   Iter : Iterator_T;
   Tmp_Target : Specified_Target_Link;
   Result : Boolean := False;
begin
   Set_List(Iter, Prot_Convert_List);
   while not Is_End(Iter) loop
      Tmp_Target := Get_Now(Iter);
      if Tmp_Target.Selection = ALL_PLACE or else
        Match_Designation(Tmp_Target.Name, To_V(Current_Unit_Name)) then
         Result := True;
      elsif Match_Designation(Tmp_Target.Name, To_V("-" & Current_Unit_Name))
      then
         Result := False;
      end if;
      Set_Next(Iter);
   end loop;
   return Result;
end Is_Attach;


function Single_Protected_ID return String is
   Unit_name : V_String := To_V(Asis.Declarations.Defining_Name_Image
                               (First_Element (Asis.Declarations.Names (Element))));
   Result : V_String := Null_Str;
begin
   if Is_Attach then
      Result := Unit_Name & "_ID : PID.Protected_ID := PID.New_ID(""" &
        Current_Unit_Name & """);" & ASCII.CR;
   end if;

   return To_S(Result);
end Single_Protected_ID;

function Protected_Type_ID return String is
   Unit_name : V_String := To_V(Asis.Declarations.Defining_Name_Image
                               (First_Element (Asis.Declarations.Names (Element))));
   Result : V_String := Null_Str;
begin
   if Is_Attach then
      Result := "type " & Unit_Name & " is record " & ASCII.CR &
        "  ID : PID.Protected_ID := PID.New_ID(""" &
        Current_Unit_Name & """);" & ASCII.CR &
        "  Self : " & Unit_Name & "_Entity;" & ASCII.CR &
        "end record;" & ASCII.CR;
   end if;
   return To_S(Result);
end Protected_Type_ID;


function Protected_Identifier return String is
begin
   if Is_Attach then
      return "_Entity";
   else
      return "";
   end if;
end;



function Make_Proxy(Now_Function : Function_Analyzer.Use_Function_Link)
                    return String is
   use Function_Analyzer;
   use Use_Function_List;
   Result : V_String := Null_Str;
   Caller_Params : V_String := Null_Str;

   -- インデント
   procedure Insert_Indent(Addition : in Integer := 0) is
   begin
      for I in 1..State.Default_Indentation_Element+Addition loop
         Result := Result & " ";
      end loop;
   end Insert_Indent;

   procedure Probe_Calls(Kind : Function_Measurement_T) is
      Iter : Function_Probe_List.Iterator_T;
      Tmp_Function_Probe : Function_Probe;
   begin
      Function_Probe_List.Set_List(Iter, Now_Function.Probes(Kind));
      while not Function_Probe_List.Is_End(Iter) loop
         Insert_Indent(2);
         Tmp_Function_Probe := Function_Probe_List.Get_Now(Iter);
         Result := Result & Probe_Call
           (Kind, Now_Function.Caller_Element,
            0,
            Tmp_Function_Probe.P_List,
            Tmp_Function_Probe.C_List);
         Function_Probe_List.Set_Next(Iter);
      end loop;
   end Probe_Calls;

begin
   --    Insert_Indent;
   Result := Result & "function " & Now_Function.Proxy_Name;
   -- Put_Line(To_String(Asis.Elements.Debug_Image(Now_Function.Callee_Element)));
   declare
      Params : Asis.Parameter_Specification_List := Asis.Declarations.Parameter_Profile(Now_Function.Callee_Element);
```

```
      First_Flag : Boolean := True;
   begin
      --   Put_Line("params : " & Integer'IMAGE(Params'LENGTH));
      --   Put_Line(To_String(Asis.Elements.Debug_Image(Now_Function.Callee_Element)));
      --   Put_Line(Boolean'IMAGE(Asis.Elements.Is_Part_Of_Implicit(Now_Function.Callee_Element)));
      if Params'LENGTH >= 1 then
         Result := Result & "(";
         Caller_Params := Caller_Params & "(";
         for I in Params'RANGE loop
            if First_Flag then
               First_Flag := False;
            else
               Result := Result & "; ";
               Caller_Params := Caller_Params & ", ";
            end if;
            Result := Result & Eliminate_Top_Space(To_String(Asis.Text.Element_Image(Params(I))));
            --       Result := Result & To_String(Asis.Declarations.Defining_Name_Image(First_Element(Asis.Declarations.Names(Params(I)))));
            Caller_Params := Caller_Params & To_String(Asis.Declarations.Defining_Name_Image(First_Element(Asis.Declarations.Names(Params(I)))));
         end loop;
         Result := Result & ")";
         Caller_Params := Caller_Params & ")";
      end if;
   end;

   --      Result := Result & " return " & Eliminate_Space(To_String(Asis.Text.Element_Image(Asis.Declarations.Result_Profile(Now_Function.Callee_Element)))) & " is" & ASCII.CR;
   Result := Result & " return " & Exp_Name(Asis.Declarations.Result_Profile(Now_Function.Callee_Element)) & " is" & ASCII.CR;


   --      Insert_Indent;

   Result := Result & "begin " & ASCII.CR ;

   Probe_Calls(FUNCTION_CALL);
   Probe_Calls(PROTECTED_FUNCTION_CALL);
   Insert_Indent;
   Result := Result & "declare" & ASCII.CR;

   Insert_Indent(2);
   Result := Result & "PROXY_Result : " & Exp_Name(Asis.Declarations.Result_Profile(Now_Function.Callee_Element)) & " := "
     & Now_Function.Genuine_Name & Caller_Params & ";" & ASCII.CR;

   -- インデント
   Insert_Indent;

   Result := Result & "begin " & ASCII.CR ;


   --      Insert_Indent(2);
   --      Result := Result & "PROXY_Result := "
   --& Now_Function.Genuine_Name & Caller_Params & ";" & ASCII.CR;
   Probe_Calls(FUNCTION_CALL_COMPLETION);
   Probe_Calls(PROTECTED_FUNCTION_CALL_COMPLETION);

   -- インデント
   Insert_Indent(2);
   Result :=  Result & "return PROXY_Result;" & ASCII.CR;
   -- インデント
   Insert_Indent;
   Result := Result & "end;" & ASCII.CR;
   Result := Result & "end " & Now_Function.Proxy_Name
     & ";" & ASCII.CR;
   Insert_Indent;
   return To_S(Result);
end Make_Proxy;

function Make_Proxy_For_Block return String is
   use Function_Analyzer;
   use Use_Function_List;
   Result : V_String := Null_Str;
   Now_Node : A_Node;
begin
   Analyze(Element, To_V(Current_Unit_Name));
   Now_Node := Current_Unit.Used_Functions;
   while Now_Node /= null loop
      if Now_Node.Object = null then Put_Line("null!"); end if;

      if not Now_Node.Object.Is_In_Decl then
         Result := Result & Make_Proxy(Now_Node.Object);
      end if;
      Now_Node := Now_Node.Next;
   end loop;
   return To_S(Result);
end Make_Proxy_For_Block;

function Make_Proxy_For_Decl return String is
   use Function_Analyzer;
   use Function_Set;
   Result : V_String := Null_Str;
   Now_Node : Function_Set.Node_Link;
   Iter : Iterator_T;
   test : Gela_Ids.Id;
begin
   --      Put_Line(To_String(Gela_Ids.Debug_Image(Gela_Ids.Create_ID(Element))));
   --      if Gela_Ids.Is_Nil(Gela_Ids.Create_ID(Element)) then
   --
   -- raise Program_Error;
   --      end if;
   test := Gela_Ids.Create_ID(Element);
   Now_Node := Function_Set_List.Search_Node
     (Current_Unit.Decl_Used_Functions, Gela_Ids.Create_ID(Element));
   Set_List(Iter, Now_Node);

   while not Is_End(Iter) loop
      Result := Result & Make_Proxy(Get_Now(Iter));
      Set_Next(Iter);
   end loop;
   return To_S(Result);
end Make_Proxy_For_Decl;
```

```
--    function Make_Variable_Package(Full_Name, Var_Name, Type_Name : String)
--    return String is
--       Result, Assign_Probes, Refer_Probes : V_String := Null_Str;
--       -- インデント
--       procedure Insert_Indent(Addition : in Integer := 0) is
--       begin
--   for I in 1..State.Default_Indentation_Element+Addition loop
--      Result := Result & " ";
--   end loop;
--       end Insert_Indent;
--    begin
--       Assign_Probes := Variable_Probes(

--       Insert_Indent;
--       Result := Result & "package " & Var_Name & "_Manager is" & ASCII.CR;

--       Insert_Indent(2);
--       Result := Result & Var_Name & "_Clone : " & Type_Name & ";" & ASCII.CR;
--       Insert_Indent(2);


--       Result := Result & "function Get return " & Type_Name " is" & ASCII.CR;
--       Insert_Indent(2);
--       Result := Result & "begin" & ASCII.CR;

--       Insert_Indent(4);
--       Result := Result & "return " & Var_Name & ";" & ASCII.CR;
--       Insert_Indent(2);
--       Result := Result & "end Get;" & ASCII.CR;


--       Insert_Indent(2);
--       Result := Result & "procedure Set(Tmp : in " & Type_Name & ") is"
--  & ASCII.CR;
--       Insert_Indent(2);
--       Result := Result & "begin" & ASCII.CR;
--       Insert_Indent(4);
--       Result := Result & Var_Name ":= Tmp;" & ASCII.CR;



--       Insert_Indent(2);
--       Result := Result & "end Set;" & ASCII.CR;

--       Insert_Indent;
--       Result := Result & "end " & Var_Name & "Manager;" & ASCII.CR;

--       return To_S(Result);
--    end Make_Variable_Package;


function Make_Var(Now_Variable : Variable_Analyzer.Use_Variable_Link)
                 return String is
   use Variable_Analyzer;
   use Use_Variable_List;
   Result : V_String := Null_Str;
   Caller_Params : V_String := Null_Str;


   procedure Insert_Indent(Addition : in Integer := 0) is
   begin
      for I in 1..State.Default_Indentation_Element+Addition loop
         Result := Result & " ";
      end loop;
   end Insert_Indent;

   procedure Probe_Calls(Kind : Variable_Measurement_T) is
      Iter : Component_List.Iterator_T;
   begin
      Component_List.Set_List(Iter, Now_Variable.Probes(Kind));
      while not Component_List.Is_End(Iter) loop
         Insert_Indent(2);
         Result := Result & Probe_Call
           (Kind, Now_Variable.Using_Element,
            Component_List.Get_Now(Iter).Qid,
            Component_List.Get_Now(Iter).Parameters,
            null,
            Component_List.Get_Now(Iter),
            Now_Variable.Genuine_Name);
         Component_List.Set_Next(Iter);
      end loop;
   end Probe_Calls;

begin
   --    Insert_Indent;
   Result := Result & "package " & Now_Variable.Proxy_Name;
   -- Put_Line(To_String(Asis.Elements.Debug_Image(Now_Function.Callee_Element)));

   Result := Result & " is" & ASCII.CR;

   Insert_Indent(2);

   Result := Result & "Clone : " & Now_Variable.Type_Name & ";" & ASCII.CR;



   Insert_Indent(2);


   Result := Result & "protected Mutex is" & ASCII.CR;
   Insert_Indent(4);

   Result := Result & "function Get return " & Now_Variable.Type_Name
     & ";" & ASCII.CR;

   Insert_Indent(4);
   Result := Result & "procedure Set(Tmp : in " &
     Now_Variable.Type_Name & ") ;" & ASCII.CR;
```

```
   Insert_Indent(2);
   Result := Result & "end Mutex;" & ASCII.CR & ASCII.CR;


   Insert_Indent(2);


   Result := Result & "protected body Mutex is" & ASCII.CR;
   Insert_Indent(4);


   Result := Result & "function Get return " & Now_Variable.Type_Name
     & " is" & ASCII.CR;
   Insert_Indent(4);
   Result := Result & "begin" & ASCII.CR;

   Probe_Calls(REFERED_VARIABLE);

   Insert_Indent(6);
   Result := Result & "return " & Now_Variable.Genuine_Name & ";"
     & ASCII.CR;
   Insert_Indent(4);
   Result := Result & "end Get;" & ASCII.CR;


   Insert_Indent(4);
   Result := Result & "procedure Set(Tmp : in " &
     Now_Variable.Type_Name & ") is"
     & ASCII.CR;
   Insert_Indent(4);
   Result := Result & "begin" & ASCII.CR;
   Insert_Indent(6);
   Result := Result & Now_Variable.Genuine_Name & ":= Tmp;" & ASCII.CR;


   Probe_Calls(ASSIGNED_VARIABLE);

   Insert_Indent(4);
   Result := Result & "end Set;" & ASCII.CR;

   Insert_Indent(2);
   Result := Result & "end Mutex;" & ASCII.CR;



   Insert_Indent;
   Result := Result & "end " & Now_Variable.Proxy_Name
     & ";" & ASCII.CR;
   Insert_Indent;
   return To_S(Result);
end Make_Var;

function Make_Var_For_Block return String is
   use Variable_Analyzer;
   use Use_Variable_List;
   Result : V_String := Null_Str;
   Now_Node : A_Node;
begin
   Analyze(Element, To_V(Current_Unit_Name));

   Now_Node := Current_Unit.Used_Variables;
   while Now_Node /= null loop
      if Now_Node.Object = null then Put_Line("null!"); end if;
      if not Now_Node.Object.Is_In_Decl then
         Result := Result & Make_Var(Now_Node.Object);
      end if;
      Now_Node := Now_Node.Next;
   end loop;
   return To_S(Result);
end Make_Var_For_Block;

function Make_Var_For_Decl return String is
   use Variable_Analyzer;
   use Variable_Set;
   Result : V_String := Null_Str;
   Now_Node : variable_Set.Node_Link;
   Iter : Iterator_T;
begin
   --      Put_Line(To_String(Gela_Ids.Debug_Image(Gela_Ids.Create_ID(Element))));
   --      if Gela_Ids.Is_Nil(Gela_Ids.Create_ID(Element)) then
   --
   -- raise Program_Error;
   --      end if;
   Now_Node := Variable_Set_List.Search_Node
     (Current_Unit.Decl_Used_Variables, Gela_Ids.Create_ID(Element));
   Set_List(Iter, Now_Node);

   while not Is_End(Iter) loop
      Result := Result & Make_Var(Get_Now(Iter));
      Set_Next(Iter);
   end loop;
   return To_S(Result);
end Make_Var_For_Decl;

procedure Process_Procedure is
   Pre_Assigns, Post_Assigns : V_String := Null_Str;
   Params : Asis.Association_List := Asis.Statements.Call_Statement_Parameters (Element, False);
   Modes : Mode_Kinds_Array
     := Get_Mode_Array
       (Asis.Statements.Corresponding_Called_Entity(Element));
   Ident : Asis.Expression;
   Tmp_Use_Variable : Variable_Analyzer.Use_Variable_Link := null;
   M_Kind : Asis.Mode_Kinds;
begin
   for I in Params'RANGE loop
      Ident := Asis_Utils.Get_Identifier
        (Asis.Expressions.Actual_Parameter(Params(I)));

      -- M_Kind := Asis_Utils.Get_Mode(Params(I));
```

```
         M_Kind := Modes(I);
         if Asis.Elements.Expression_Kind(Ident) = An_Identifier then
            Tmp_Use_Variable := Variable_Analyzer.Use_Variable_List.Search_Node
              (Current_Unit.Used_Variables,
               Gela_Ids.Create_ID
                 (Au_Corresponding_Name_Definition(Ident)));
         else
            Tmp_Use_Variable := null;
         end if;

         if Variable_Analyzer."/="(Tmp_Use_Variable, null) then
            if M_Kind = An_In_Out_Mode then
               Pre_Assigns := Pre_Assigns & Tmp_Use_Variable.Proxy_Name
                 & ".Clone := " & Tmp_Use_Variable.Proxy_Name &
                 ".Mutex.Get;" & ASCII.CR;
            end if;
            if M_Kind = An_Out_Mode or M_Kind = An_In_Out_Mode then
               Post_Assigns := Post_Assigns & Tmp_Use_Variable.Proxy_Name
                 & ".Mutex.Set(" & Tmp_Use_Variable.Proxy_Name &
                 ".Clone);" & ASCII.CR;
               Convert_List.Set_Node(Aconv_List,
                                     Gela_Ids.Create_ID(Params(I)),
                                     Tmp_Use_Variable.Proxy_Name & ".Clone");
            else

               Convert_List.Set_Node(Aconv_List,
                                     Gela_Ids.Create_ID(Params(I)),
                                     Tmp_Use_Variable.Proxy_Name & ".Get");

            end if;
         end if;
      end loop;

      if Is_Protected(Asis.Statements.Corresponding_Called_Entity(Element)) then

         Send_Label (To_S(Designated_Probes(PROTECTED_PROCEDURE_CALL,
           To_V(Current_Unit_Name),
           False) & Pre_Assigns));
         Push;
         Push (";" & ASCII.CR & To_S(Post_Assigns & Designated_Probes(PROTECTED_PROCEDURE_CALL_COMPLETION, To_V(Current_Unit_Name), False)),
               Is_Comma_List,
               Count (Params));
         Set_Conv_Flag;
      else
         Send_Label (To_S(Designated_Probes(PROCEDURE_CALL,
           To_V(Current_Unit_Name),
           False) & Pre_Assigns));
         Push;
         Push (";" & ASCII.CR & To_S(Post_Assigns & Designated_Probes(PROCEDURE_CALL_COMPLETION, To_V(Current_Unit_Name), False)),
               Is_Comma_List,
               Count (Params));
      end if;

   end Process_Procedure;

   -- to keep the size of some lists
   L, M, N : Integer := 0;

begin
   --   Put_Line(To_String(Asis.Elements.Debug_Image(Element)));
   if not Is_Send_Lock then
      Pass_Element;
   end if;
   if State.Finishing_Traversal
   then
      return;
   end if;

   if Is_Making_Clause then
      Send ("with Global_Types; use Global_Types;" & Ascii.CR);
      Send ("with Ada.Calendar; use Ada.Calendar;" & Ascii.CR);
      Send ("with Ada.Task_Identification; use Ada.Task_Identification;"
            & Ascii.CR);
      Send ("with Pid;" & Ascii.CR);
      if Spec_Reader.Has_Spec then
         Send (To_S("with " & Spec_Name & "; use " & Spec_Name & ";") & Ascii.CR);
      end if;
      Is_Making_Clause := False;
   end if;

   ------< begining of the case >------
   case Asis.Elements.Element_Kind (Element) is
   when Not_An_Element =>
      null;
   when A_Pragma =>
      if Current_Unit.Is_In_MH then
         Control := Abandon_Children;
         Send (To_S(Null_Str));
      else

         case Asis.Elements.Pragma_Kind (Element) is
         when Not_A_Pragma =>
            Ada.Text_IO.Put ("<<Node Not_A_Pragma>>");
         when others =>
            if Is_Equal(Asis.Elements.Pragma_Name_Image(Element),
                        Mh_Reserved_Word) then
               Control := Abandon_Children;
               Send (To_S(Null_Str));
               Current_Unit.Is_In_MH := True;
            else

               Send ("pragma " &
                       To_String (Asis.Elements.Pragma_Name_Image (Element)) & " ");
               L := Count (Asis.Elements.Pragma_Argument_Associations (Element));
               Push (";" & ASCII.CR,
                     Is_Comma_List,
                     L);
            end if;
         end case;
```

```
   end if;

when A_Defining_Name =>
   case Asis.Elements.Defining_Name_Kind (Element) is
      when Not_A_Defining_Name =>
         Ada.Text_IO.Put ("<<Node Not_A_Defining_Name>>");
      when A_Defining_Identifier |
           A_Defining_Character_Literal |
           A_Defining_Enumeration_Literal =>
         -- Rename main procedure -- by T.E --
         case Asis.Elements.Declaration_Kind(Asis.Elements.Enclosing_Element(Element)) is
         when A_Procedure_Body_Declaration =>
            Send
              ("dd_" &To_String (Asis.Declarations.Defining_Name_Image (Element))
               & " ");
         when others =>
            Send
              (To_String (Asis.Declarations.Defining_Name_Image (Element))
               & " ");
         end case;
      when A_Defining_Operator_Symbol =>
         Send
           (To_String (Asis.Declarations.Defining_Name_Image (Element))
            & " ");
      when A_Defining_Expanded_Name =>
         Send
           (To_String (Asis.Declarations.Defining_Name_Image (Element))
            & " ");
         -- don't process the prefix and selector ...
         Control := Asis.Abandon_Children;
   end case;
when A_Declaration =>
   Send(Make_Proxy_For_Decl & Make_Var_For_decl);

   case Asis.Elements.Declaration_Kind (Element) is
      when Not_A_Declaration =>                      -- An unexpected element
      Ada.Text_IO.Put ("<<Node Not_A_Declaration>>");
      when An_Ordinary_Type_Declaration =>           -- 3.2.1
         Send ("type ");
         if Is_Here (Asis.Declarations.Discriminant_Part (Element))
         then
            Push;
         end if;
         Push ("is ");
         Push (";" & ASCII.CR);
      when A_Task_Type_Declaration =>                -- 3.2.1
         -- Name_Handler
         Enter_Unit_Spec(To_V(Asis.Declarations.Defining_Name_Image
           (First_Element (Asis.Declarations.Names (Element)))),
           Element,
           A_Task_Type_Declaration, TASKS);
         --       Regist_Name(Gela_Ids.Create_Id(Element));
         -- Name_Handler
         Send ("task type ");
         if Is_Here (Asis.Declarations.Discriminant_Part (Element))
         then
            Push;
         end if;
         if Is_Here (Asis.Declarations.Type_Declaration_View (Element))
         then
            Push ("is" & ASCII.CR);
            Push (To_String (Asis.Declarations.Defining_Name_Image
              (First_Element (Asis.Declarations.Names (Element)))) &
                ";" & ASCII.CR); -- There is at least the name
         else
            Push (";" & ASCII.CR); -- There is at least the name
         end if;
      when A_Protected_Type_Declaration =>           -- 3.2.1
         -- Name_Handler
         Enter_Unit_Spec(To_V(Asis.Declarations.Defining_Name_Image
           (First_Element (Asis.Declarations.Names (Element)))),
           Element,
           A_Protected_Type_Declaration,
           PROTECTED_OBJECTS);
         --       Regist_Name(Gela_Ids.Create_Id(Element));
         -- Name_Handler
         Send ("protected type ");
         if Is_Here (Asis.Declarations.Discriminant_Part (Element))
         then
            Push;
         end if;
         Push ("is" & ASCII.CR);
         Push (To_String (Asis.Declarations.Defining_Name_Image
           (First_Element (Asis.Declarations.Names (Element)))) &
             Protected_Identifier &
             ";" & ASCII.CR & Protected_Type_ID);
         Is_Protected_Type_Identifier := True;
         Protected_With_Id_List.Set_Node(Pid_Flags,
                                         Gela_Ids.Create_ID(Element),
                                         True);

      when An_Incomplete_Type_Declaration =>         -- 3.2.1
         Send ("type ");
         if Is_Here (Asis.Declarations.Discriminant_Part (Element))
         then
            Push;
         end if;
         Push (";" & ASCII.CR);
         ------------------------------------------------------------Ada2005
      when A_Tagged_Incomplete_Type_Declaration =>
         null;
         ------------------------------------------------------------Ada2005
      when A_Private_Type_Declaration =>             -- 3.2.1 -> Trait_Kinds
         Send ("type ");
         if Is_Here (Asis.Declarations.Discriminant_Part (Element))
         then
            Push;
         end if;
```

```
   Push ("is "); -- The trait is writen after ....
   Push (";" & ASCII.CR);
when A_Private_Extension_Declaration =>          -- 3.2.1 -> Trait_Kinds
   Send ("type ");
   if Is_Here (Asis.Declarations.Discriminant_Part (Element))
   then
      Push;
   end if;
   case Asis.Elements.Trait_Kind (Element) is
      when An_Abstract_Trait |
           An_Abstract_Private_Trait |
           An_Abstract_Limited_Trait |
           An_Abstract_Limited_Private_Trait =>
         Push ("is abstract new ");
      when others =>
         Push ("is new ");
   end case;
   Push (";" & ASCII.CR);

when A_Subtype_Declaration =>                    -- 3.2.2
   Send ("subtype ");
   Push ("is ");
   Push (";" & ASCII.CR);
when A_Variable_Declaration =>                   -- 3.3.1 -> Trait_Kinds

   CDT_Subject_Task := Gela_Ids.Create_ID(Element);
   Push (": " & Trait_String,
         Is_Comma_No_Parenthesis_List,
         Count (Asis.Declarations.Names (Element)));
   if Is_Here (Asis.Declarations.Initialization_Expression (Element))
   then
      Push (":= ");
   end if;
   Push (";" & ASCII.CR & Declared_CDT(Element));
when A_Constant_Declaration =>                   -- 3.3.1 -> Trait_Kinds
   Push (": " & Trait_String & "constant ",
         Is_Comma_No_Parenthesis_List,
         Count (Asis.Declarations.Names (Element)));
   if Is_Here (Asis.Declarations.Initialization_Expression (Element))
   then
      Push (":= ");
   end if;
   Push (";" & ASCII.CR);
when A_Deferred_Constant_Declaration =>       -- 3.3.1 -> Trait_Kinds
   Push (": " & Trait_String & "constant ",
         Is_Comma_No_Parenthesis_List,
         Count (Asis.Declarations.Names (Element)));
   Push (";" & ASCII.CR);
when A_Single_Task_Declaration =>                 -- 3.3.1
   CDT_Subject_Task := Gela_Ids.Create_ID(Element);
   declare
      Send_CDT : String := Declared_CDT(Element);
   begin
      -- Name_Handler
      Enter_Unit_Spec(To_V(Asis.Declarations.Defining_Name_Image
        (First_Element (Asis.Declarations.Names (Element)))),
        Element,
        A_Single_Task_Declaration, TASKS);
      --       Regist_Name(Gela_Ids.Create_Id(Element));
      -- Name_Handler

      Send ("task ");
      if Is_Here (Asis.Declarations.Object_Declaration_View (Element)) then
         Push ("is" & ASCII.CR);
         Push (To_String (Asis.Declarations.Defining_Name_Image
           (First_Element (Asis.Declarations.Names (Element)))) &
             ";" & ASCII.CR & Send_CDT);
      else
         Push (";" & ASCII.CR & Send_CDT);
      end if;
   end;
when A_Single_Protected_Declaration =>           -- 3.3.1
   -- Name_Handler
   Enter_Unit_Spec(To_V(Asis.Declarations.Defining_Name_Image
     (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Single_Protected_Declaration, PROTECTED_OBJECTS);
   --       Regist_Name(Gela_Ids.Create_Id(Element));
   -- Name_Handler
   Send (Single_Protected_ID & "protected ");
   Push ("is" & ASCII.CR);
   Indent;
   Push (To_String (Asis.Declarations.Defining_Name_Image
     (First_Element (Asis.Declarations.Names (Element))))
     & ";" & ASCII.CR);
   Protected_With_Id_List.Set_Node(Pid_Flags,
                                   Gela_Ids.Create_ID(Element),
                                   True);
when An_Integer_Number_Declaration =>            -- 3.3.2
   Push (": constant :=  ",
         Is_Comma_No_Parenthesis_List,
         Count (Asis.Declarations.Names (Element)));
   Push (";" & ASCII.CR);
when A_Real_Number_Declaration =>                -- 3.3.2
   Push (": constant :=  ",
         Is_Comma_No_Parenthesis_List,
         Count (Asis.Declarations.Names (Element)));
   Push (";" & ASCII.CR);
when An_Enumeration_Literal_Specification =>     -- 3.5.1
   Push;
when A_Discriminant_Specification =>             -- 3.7   -> Trait_Kinds
   Push (": " & Trait_String,
         Is_Comma_No_Parenthesis_List,
         Count (Asis.Declarations.Names (Element)));
   if Is_Here (Asis.Declarations.Initialization_Expression (Element))
   then
      Push (":= ");
   end if;
   Push;
```

```
when A_Component_Declaration =>                 -- 3.8
   Push (": ",
         Is_Comma_No_Parenthesis_List,
         Count (Asis.Declarations.Names (Element)));
   if Is_Here (Asis.Declarations.Initialization_Expression (Element))
   then
      Push (":= ");
   end if;
   Push (";" & ASCII.CR);
when A_Loop_Parameter_Specification =>          -- 5.5   -> Trait_Kinds
   Push ("in " & Trait_String);
   Push;
   --|Ada 2012 start
when A_Generalized_Iterator_Specification =>
   null;
when An_Element_Iterator_Specification =>
   null;
   --|Ada 2012 end

when A_Procedure_Declaration =>                 -- 6.1   -> Trait_Kinds
   -- Name_Handler
   declare
      This_Unit_Class : Unit_Class;
   begin
      if Is_Protected(Element) then
         This_Unit_Class := PROTECTED_FUNCTIONS;
      else
         This_Unit_Class := FUNCTIONS;
      end if;
      Enter_Unit_Spec
        (To_V(Asis.Declarations.Defining_Name_Image
         (First_Element (Asis.Declarations.Names (Element)))),
         Element,
         A_Procedure_Body_Declaration, This_Unit_Class);
   end;
   --       Regist_Name(Gela_Ids.Create_Id(Element));
   -- Name_Handler

   if Is_Private_Unit (Element) then
      Send ("private ");
   end if;
   Send ("procedure ");
   Push;
   case Asis.Elements.Trait_Kind (Element) is
      when Not_A_Trait =>
         Ada.Text_IO.Put ("<<Node Not_A_Trait in A_Procedure_Declaration>>");
      when An_Abstract_Trait =>
         Push ("is abstract;" & ASCII.CR,
               Is_Semi_Colon_List,
               Count (Asis.Declarations.Parameter_Profile (Element)));
      when others =>
         Push (";" & ASCII.CR,
               Is_Semi_Colon_List,
               Count (Asis.Declarations.Parameter_Profile (Element)));
   end case;
when A_Function_Declaration =>                  -- 6.1   -> Trait_Kinds
   -- Name_Handler
   declare
      This_Unit_Class : Unit_Class;
   begin
      if Is_Protected(Element) then
         This_Unit_Class := PROTECTED_FUNCTIONS;
      else
         This_Unit_Class := FUNCTIONS;
      end if;
      Enter_Unit_Spec
        (To_V(Asis.Declarations.Defining_Name_Image
         (First_Element (Asis.Declarations.Names (Element)))),
         Element,
         A_Function_Body_Declaration,
         This_Unit_Class);
   end;
   --       Regist_Name(Gela_Ids.Create_Id(Element));
   -- Name_Handler

   if Is_Private_Unit (Element) then
      Send ("private ");
   end if;
   Send ("function ");
   Push ("");
   Push ("return ",
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
   case Asis.Elements.Trait_Kind (Element) is
      when Not_A_Trait =>
         Ada.Text_IO.Put ("<<Node Not_A_Trait in A_Procedure_Declaration>>");
      when An_Abstract_Trait =>
         Push ("is abstract;" & ASCII.CR);
      when others =>
         Push (";" & ASCII.CR);
   end case;
when A_Parameter_Specification =>               -- 6.1   -> Trait_Kinds
   case Asis.Elements.Mode_Kind (Element) is
      when Not_A_Mode =>
         Push ("<<Not_A_Mode !!!>>");
      when A_Default_In_Mode =>
         -- it is the only mode that can be access...
         Push (": " & Trait_String,
               Is_Comma_No_Parenthesis_List,
               Count (Asis.Declarations.Names (Element)));
      when An_In_Mode =>
         Push (": in ",
               Is_Comma_No_Parenthesis_List,
               Count (Asis.Declarations.Names (Element)));
      when An_Out_Mode =>
         Push (": out ",
               Is_Comma_No_Parenthesis_List,
               Count (Asis.Declarations.Names (Element)));
      when An_In_Out_Mode =>
```

```
         Push (": in out ",
               Is_Comma_No_Parenthesis_List,
               Count (Asis.Declarations.Names (Element)));
   end case;
   if (Is_Here (Asis.Declarations.Initialization_Expression (Element))) then
      Push (":= ");
   end if;
   Push;
when A_Procedure_Body_Declaration =>             -- 6.3
   -- Name_Handler
   declare
      This_Unit_Class : Unit_Class;
   begin
      if Current_Unit.This_Unit_Class = MAIN_TASK then
         This_Unit_Class := MAIN_PROCEDURE;
      elsif Is_Protected(Element) then
         This_Unit_Class := PROTECTED_PROCEDURES;
      else
         This_Unit_Class := PROCEDURES;
      end if;
      Enter_Unit_Body
        (To_V(Asis.Declarations.Defining_Name_Image
         (First_Element (Asis.Declarations.Names (Element)))),
         Element,
         A_Procedure_Body_Declaration,
         This_Unit_Class);
   end;

   --       Regist_Name(Gela_Ids.Create_Id(Asis.Declarations.Corresponding_Declaration(Element)));
   -- Name_Handler

   Send (Unit_Body_Beginning (Element, A_Procedure_Body_Declaration));
   --Send ("procedure ");
   Push ("");
   --              Push ("is" & ASCII.CR,
   Push (Is_Label(True),
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
   Indent;
   --               Push("begin" & ASCII.CR,
   Push(Make_Proxy_For_Block & Make_Var_For_Block & Begin_Label ,
        Not_In_A_List,
        Count (Asis.Declarations.Body_Declarative_Items (Element , True)));
   Indent;
   L := Count (Asis.Declarations.Body_Statements (Element, True));
   M := Count (Asis.Declarations.Body_Exception_Handlers (Element, True));
   -- Is_Name_Repeated is not implemented
   if M = 0
   then
      --                  Push ("end " &
      if Current_Unit.This_Unit_Class = MAIN_PROCEDURE then
         Push (End_Label & "dd_" & To_String (Asis.Declarations.Defining_Name_Image
           (First_Element (Asis.Declarations.Names (Element)))) &
             ";" & ASCII.CR, Not_In_A_List, L);
      else
         Push (End_Label &
             To_String (Asis.Declarations.Defining_Name_Image
             (First_Element (Asis.Declarations.Names (Element)))) &
             ";" & ASCII.CR, Not_In_A_List, L);
      end if;
   else
      --                  Push ("exception" & ASCII.CR,
      Push (Exception_Label,
            Not_In_A_List,
            L);
      Indent;

      if Current_Unit.This_Unit_Class = MAIN_PROCEDURE then
         Push ("end " & "dd_" & To_String (Asis.Declarations.Defining_Name_Image
           (First_Element (Asis.Declarations.Names (Element)))) &
             ";" & ASCII.CR, Not_In_A_List, M);
      else
         Push ("end " & To_String (Asis.Declarations.Defining_Name_Image
           (First_Element (Asis.Declarations.Names (Element)))) &
             ";" & ASCII.CR, Not_In_A_List, M);
      end if;
   end if;
   Indent;
when A_Function_Body_Declaration =>              -- 6.3
   -- Name_Handler
   declare
      This_Unit_Class : Unit_Class;
   begin
      if Is_Protected(Element) then
         This_Unit_Class := PROTECTED_FUNCTIONS;
      else
         This_Unit_Class := FUNCTIONS;
      end if;

      Enter_Unit_Body
        (To_V(Asis.Declarations.Defining_Name_Image
         (First_Element (Asis.Declarations.Names (Element)))),
         Element,
         A_Function_Body_Declaration,
         This_Unit_Class);
   end;
   --       Regist_Name(Gela_Ids.Create_Id(Asis.Declarations.Corresponding_Declaration(Element)));
   -- Name_Handler
   Send (Unit_Body_Beginning (Element, A_Function_Body_Declaration));
   --Send ("function ");
   Push ("");
   Push ("return ",
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
   --              Push ("is" & ASCII.CR);
   Push (Is_Label(True));
   Indent;
   --               Push("begin" & ASCII.CR,
   Push(Make_Proxy_For_Block & Make_Var_For_Block & Begin_Label,
```

```
      Not_In_A_List,
      Count (Asis.Declarations.Body_Declarative_Items (Element , True)));
   Indent;
   L := Count (Asis.Declarations.Body_Statements (Element, True));
   M := Count (Asis.Declarations.Body_Exception_Handlers (Element, True));
   -- Is_Name_Repeated is not implemented
   if M = 0
   then
      Push ("end " &
        --                  Push (End_Label &
        To_String (Asis.Declarations.Defining_Name_Image
        (First_Element (Asis.Declarations.Names (Element)))) &
        ";" & ASCII.CR ,
      Not_In_A_List,
      L);
   else
      Push ("exception" & ASCII.CR,
           --                  Push (Exception_Label,
           Not_In_A_List,
           L);
      Indent;
      Push ("end " &
          To_String (Asis.Declarations.Defining_Name_Image
          (First_Element (Asis.Declarations.Names (Element)))) &
          ";" & ASCII.CR ,
        Not_In_A_List,
        M);
   end if;
   Indent;
   -- --|A2005 start
when A_Return_Variable_Specification =>
   null;
when A_Return_Constant_Specification =>
   null;
when A_Null_Procedure_Declaration =>             -- 6.7
   null;
   --  --|A2005 end
   --  --|A2012 start
when An_Expression_Function_Declaration =>     -- 6.8
   null;
   --  --|A2012 end

when A_Package_Declaration =>                    -- 7.1
   -- Name_Handler
   Enter_Unit_Spec(To_V(Asis.Declarations.Defining_Name_Image
     (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Package_Body_Declaration,
     LIBRARY_PACKAGES);
   --       Regist_Name(Gela_Ids.Create_Id(Element));
   -- Name_Handler

   if Is_Private_Unit (Element) then
      Send ("private ");
   end if;
   Send ("package ");
   --              Push ("is" & ASCII.CR);
   Push (Is_Label);
   L := Count (Asis.Declarations.Visible_Part_Declarative_Items (Element, True));
   if Asis.Declarations.Is_Private_Present (Element)
   then
      Push ("private" & ASCII.CR,
           Not_In_A_List,
           L);
      Indent;
      Push ("end " &
          To_String (Asis.Declarations.Defining_Name_Image
          (First_Element (Asis.Declarations.Names (Element)))) &
          ";" & ASCII.CR ,
        Not_In_A_List,
        Count (Asis.Declarations.Private_Part_Declarative_Items (Element, True)));
   else
      Push ("end " &
          To_String (Asis.Declarations.Defining_Name_Image
          (First_Element (Asis.Declarations.Names (Element)))) &
          ";" & ASCII.CR ,
        Not_In_A_List,
        L);
   end if;
   Indent;
when A_Package_Body_Declaration =>               -- 7.2
   -- Name_Handler
   Enter_Unit_Body
     (To_V(Asis.Declarations.Defining_Name_Image
      (First_Element (Asis.Declarations.Names (Element)))),
      Element,
      A_Package_Body_Declaration,
      LIBRARY_PACKAGES);
   if Component_List."/="(Spec_Reader.Spec_Data(GET_CDT),
                          null) then
      Call_Analyzer.Find_Calls(Element);
   end if;
   --       Regist_Name(Gela_Ids.Create_Id(Asis.Declarations.Corresponding_Declaration(Element)));
   -- Name_Handler
   Send (Unit_Body_Beginning (Element, A_Package_Body_Declaration));
   --Send ("package body ");
   L := Count (Asis.Declarations.Body_Declarative_Items (Element, True));
   M := Count (Asis.Declarations.Body_Statements (Element, True));
   N := Count (Asis.Declarations.Body_Exception_Handlers (Element, True));
   declare
      End_String : String := "end " &
        To_String (Asis.Declarations.Defining_Name_Image
                   (First_Element (Asis.Declarations.Names (Element)))) &
        ";" & ASCII.CR;
      End_String2 : String := End_Label &
        To_String (Asis.Declarations.Defining_Name_Image
                   (First_Element (Asis.Declarations.Names (Element)))) &
        ";" & ASCII.CR;
   begin
```

```
      if L = 0
      then
         if M = 0  -- then N = 0 too
         then
            Push ("is" & ASCII.CR & ASCII.CR & End_String);
         else
            --                          Push ("is" & ASCII.CR & "begin" & ASCII.CR);
            Push ("is" & ASCII.CR & MH_Spec &
                    Make_Proxy_For_Block & Make_Var_For_Block
                  & Begin_Label);
            if N = 0
            then
               --                          Push (End_String, Not_In_A_List, M);
               Push (End_String2, Not_In_A_List, M);
               Indent;
            else
               --                          Push ("exception" & ASCII.CR, Not_In_A_List, M);
               Push (Exception_Label, Not_In_A_List, M);
               Indent;
               Push (End_String, Not_In_A_List, N);
               Indent;
            end if;
         end if;
      else
         Push ("is" & ASCII.CR);
         if M = 0
         then
            Push (End_String, Not_In_A_List, L);
            Indent;
         else
            --                          Push ("begin" & ASCII.CR, Not_In_A_List, L);
            Push (MH_Spec & Make_Proxy_For_Block & Make_Var_For_Block & Begin_Label, Not_In_A_List, L);
            Indent;
            if N = 0
            then
               --                          Push (End_String, Not_In_A_List, M);
               Push (End_String2, Not_In_A_List, M);
               Indent;
            else
               --                          Push ("exception" & ASCII.CR, Not_In_A_List, M);
               Push (Exception_Label & ASCII.CR, Not_In_A_List, M);
               Indent;
               Push (End_String, Not_In_A_List, N);
               Indent;
            end if;
         end if;
      end if;
   end;
when An_Object_Renaming_Declaration =>           -- 8.5.1
   Push (": ");
   Push ("renames ");
   Push (";" & ASCII.CR);
when An_Exception_Renaming_Declaration =>        -- 8.5.2
   Push (": exception renames ");
   Push (";" & ASCII.CR);
when A_Package_Renaming_Declaration =>           -- 8.5.3
   Send ("package ");
   Push ("renames ");
   Push (";" & ASCII.CR);
when A_Procedure_Renaming_Declaration =>         -- 8.5.4
   Send ("procedure ");
   Push;
   Push ("renames ",
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
   Push (";" & ASCII.CR);
when A_Function_Renaming_Declaration =>          -- 8.5.4
   Send ("function ");
   Push;
   Push ("return ",
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
   Push ("renames ");
   Push (";" & ASCII.CR);
when A_Generic_Package_Renaming_Declaration =>   -- 8.5.5
   Send ("generic package ");
   Push ("renames ");
   Push (";" & ASCII.CR);
when A_Generic_Procedure_Renaming_Declaration => -- 8.5.5
   Send ("generic procedure ");
   Push ("renames ");
   Push (";" & ASCII.CR);
when A_Generic_Function_Renaming_Declaration =>  -- 8.5.5
   Send ("generic function ");
   Push ("renames ");
   Push (";" & ASCII.CR);
when A_Task_Body_Declaration =>                  -- 9.1
   -- Name_Handler
   Enter_Unit_Body(To_V(Asis.Declarations.Defining_Name_Image
     (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Task_Body_Declaration, TASKS);
   --       Regist_Name(Gela_Ids.Create_Id(Asis.Declarations.Corresponding_Declaration(Element)));
   -- Name_Handler
   declare
      Is_Label_To_Push : String := Is_Label(True);
      Get_Parent_Task_ID : V_String := Null_Str;
   begin
      if Is_Parent_Task then
         Get_Parent_Task_ID := Parent_Task_ID
           & ": Task_ID := Current_Task;" & ASCII.CR;
         Is_Parent_Task := False;
      end if;
      Send (To_S(Get_Parent_Task_ID & Unit_Body_Beginning(Element, A_Task_Body_Declaration)));
      --  Send ("task body ");
      --                Push ("is" & ASCII.CR);
      Push (Is_Label_To_Push);
   end;
   --               Push("begin" & ASCII.CR,
```

```
   Push(Make_Proxy_For_Block & Make_Var_For_Block & Begin_Label,
        Not_In_A_List,
        Count (Asis.Declarations.Body_Declarative_Items (Element, True)));
   Indent;
   L := Count (Asis.Declarations.Body_Statements (Element, True));
   M := Count (Asis.Declarations.Body_Exception_Handlers (Element, True));
   -- Is_Name_Repeated is not implemented
   if M = 0
   then
      --                  Push ("end " &
      Push (End_Label &
          To_String (Asis.Declarations.Defining_Name_Image
          (First_Element (Asis.Declarations.Names (Element)))) &
          ";" & ASCII.CR ,
        Not_In_A_List,
        L);
   else
      --                  Push ("exception" & ASCII.CR,
      Push (Exception_Label,
            Not_In_A_List,
            L);
      Indent;
      Push ("end " &
          To_String (Asis.Declarations.Defining_Name_Image
          (First_Element (Asis.Declarations.Names (Element)))) &
          ";" & ASCII.CR ,
        Not_In_A_List,
        M);
   end if;
   Indent;

when A_Protected_Body_Declaration =>               -- 9.4
   -- Name_Handler
   Enter_Unit_Body(To_V(Asis.Declarations.Defining_Name_Image
     (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     A_Protected_Body_Declaration, PROTECTED_OBJECTS);
   --       Regist_Name(Gela_Ids.Create_Id(Asis.Declarations.Corresponding_Declaration(Element)));
   -- Name_Handler
   Send (Unit_Body_Beginning (Element, A_Protected_Body_Declaration));
   --  Send ("protected body ");
   Push ("is" & ASCII.CR);
   Push ("end " &
       To_String (Asis.Declarations.Defining_Name_Image
       (First_Element (Asis.Declarations.Names (Element)))) &
       ";" & ASCII.CR ,
     Not_In_A_List,
     Count (Asis.Declarations.Protected_Operation_Items (Element, True)));
   Indent;
when An_Entry_Declaration =>                      -- 9.5.2
   -- Name_Handler
   Enter_Unit_Spec(To_V(Asis.Declarations.Defining_Name_Image
     (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     An_Entry_Body_Declaration, TASKS);
   Send ("entry ");
   L := Count (Asis.Declarations.Parameter_Profile (Element));
   if L /= 0
   then
      Push;
      if Is_Here (Asis.Declarations.Entry_Family_Definition (Element))
      then
         Push ("", Is_Comma_List);
         -- we say comma list in order to have the parenthesis
      end if;
      Push (";" & ASCII.CR,
            Is_Semi_Colon_List, L);
   else
      if Is_Here (Asis.Declarations.Entry_Family_Definition (Element))
      then
         Push;
         Push (";" & ASCII.CR, Is_Comma_List);
         -- we say comma list in order to have the parenthesis
      else
         Push (";" & ASCII.CR);
      end if;
   end if;
when An_Entry_Body_Declaration =>          -- 9.5.2
   -- Name_Handler
   Enter_Unit_Body(To_V(Asis.Declarations.Defining_Name_Image
     (First_Element (Asis.Declarations.Names (Element)))),
     Element,
     An_Entry_Body_Declaration, PROTECTED_ENTRY_BODIES);
   -- entry body
   -- Name_Handler
   Send ("entry ");
   if Is_Here (Asis.Declarations.Entry_Index_Specification (Element))
   then
      Push;
      Push ("", Is_Comma_List);
   else
      Push;
   end if;
   Push ("when ",
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
   --              Push ("is" & ASCII.CR);
   Push (Is_Label(True));
   --              Push ("begin" & ASCII.CR,
   Push (Make_Proxy_For_Block & Make_Var_For_Block & Begin_Label,
         Not_In_A_List,
         Count (Asis.Declarations.Body_Declarative_Items (Element, True)));
   Indent;
   L := Count (Asis.Declarations.Body_Statements (Element));
   M := Count (Asis.Declarations.Body_Exception_Handlers (Element));
   if M = 0
   then
      --                  Push ("end " &
      Push (End_Label &
```

```
         To_String (Asis.Declarations.Defining_Name_Image
         (First_Element (Asis.Declarations.Names (Element)))) &
         ";" & ASCII.CR ,
       Not_In_A_List,
       L);
      Indent;
   else
      --                  Push ("exception" & ASCII.CR,
      Push (Exception_Label,
            Not_In_A_List,
            L);
      Indent;
      Push ("end " &
          To_String (Asis.Declarations.Defining_Name_Image
          (First_Element (Asis.Declarations.Names (Element)))) &
          ";" & ASCII.CR ,
        Not_In_A_List,
        M);
      Indent;
   end if;
when An_Entry_Index_Specification =>               -- 9.5.2
   Send ("for ");
   Push ("in ");
   Push;
when A_Procedure_Body_Stub =>                      -- 10.1.3
   Send ("procedure ");
   Push;
   Push ("is separate;" & ASCII.CR,
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
when A_Function_Body_Stub =>                       -- 10.1.3
   Send ("function ");
   Push ("");
   Push ("return ",
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
   Push ("is separate;" & ASCII.CR);
when A_Package_Body_Stub =>                        -- 10.1.3
   Send ("package body ");
   Push ("is separate;" & ASCII.CR);
when A_Task_Body_Stub =>                           -- 10.1.3
   Send ("task body ");
   Push ("is separate;" & ASCII.CR);
when A_Protected_Body_Stub =>                        -- 10.1.3
   Send ("protected body ");
   Push ("is separate;" & ASCII.CR);
when An_Exception_Declaration =>                   -- 11.1
   Push (": exception;" & ASCII.CR,
         Is_Comma_No_Parenthesis_List,
         Count (Asis.Declarations.Names (Element)));
when A_Choice_Parameter_Specification =>           -- 11.2
   Push (": "); -- in exception handler ...
when A_Generic_Procedure_Declaration =>            -- 12.1
   if Is_Private_Unit (Element) then
      Send ("private" & ASCII.CR);
   end if;
   Send ("generic" & ASCII.CR);
   Push ("procedure ",
         Not_In_A_List,
         Count (Asis.Declarations.Generic_Formal_Part (Element, true)));
   Indent;
   Push;
   Push (";" & ASCII.CR ,
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
when A_Generic_Function_Declaration =>             -- 12.1
   if Is_Private_Unit (Element) then
      Send ("private" & ASCII.CR);
   end if;
   Send ("generic" & ASCII.CR);
   Push ("function ",
         Not_In_A_List,
         Count (Asis.Declarations.Generic_Formal_Part (Element, true)));
   Indent;
   Push;
   Push ("return ",
         Is_Semi_Colon_List,
         Count (Asis.Declarations.Parameter_Profile (Element)));
   Push (";" & ASCII.CR);
when A_Generic_Package_Declaration =>              -- 12.1
   if Is_Private_Unit (Element) then
      Send ("private" & ASCII.CR);
   end if;
   Send ("generic" & ASCII.CR);
   Push ("package ",
         Not_In_A_List,
         Count (Asis.Declarations.Generic_Formal_Part (Element, true)));
   Indent;
   --               Push ("is" & ASCII.CR);
   Push (Is_Label);
   L := Count (Asis.Declarations.Visible_Part_Declarative_Items (Element, True));
   if Asis.Declarations.Is_Private_Present (Element)
   then
      Push ("private" & ASCII.CR,
            Not_In_A_List,
            L);
      Indent;
      Push ("end " &
          To_String (Asis.Declarations.Defining_Name_Image
          (First_Element (Asis.Declarations.Names (Element)))) &
          ";" & ASCII.CR ,
        Not_In_A_List,
        Count (Asis.Declarations.Private_Part_Declarative_Items (Element, True)));
   else
      Push ("end " &
          To_String (Asis.Declarations.Defining_Name_Image
          (First_Element (Asis.Declarations.Names (Element)))) &
          ";" & ASCII.CR ,
        Not_In_A_List,
```

```
         L);
      end if;
      Indent;

   when A_Package_Instantiation =>                  -- 12.3
      Send ("package ");
      Push ("is new ");
      Push;
      Push (";" & ASCII.CR,
            Is_Comma_List,
            Count (Asis.Declarations.Generic_Actual_Part (Element, False)));
   when A_Procedure_Instantiation =>                -- 12.3
      Send ("procedure ");
      Push ("is new ");
      Push;
      Push (";" & ASCII.CR,
            Is_Comma_List,
            Count (Asis.Declarations.Generic_Actual_Part (Element, False)));
   when A_Function_Instantiation =>                 -- 12.3
      Send ("function ");
      Push ("is new ");
      Push;
      Push (";" & ASCII.CR,
            Is_Comma_List,
            Count (Asis.Declarations.Generic_Actual_Part (Element, False)));
   when A_Formal_Object_Declaration =>              -- 12.4  -> Mode_Kinds
      case Asis.Elements.Mode_Kind (Element) is
         when Not_A_Mode =>
            Push ("<<Not_A_Mode !!!>>");
         when A_Default_In_Mode =>
            -- it is the only mode that can be access...
            Push (": " & Trait_String,
                  Is_Comma_No_Parenthesis_List,
                  Count (Asis.Declarations.Names (Element)));
         when An_In_Mode =>
            Push (": in ",
                  Is_Comma_No_Parenthesis_List,
                  Count (Asis.Declarations.Names (Element)));
         when An_Out_Mode =>
            Push (": out ",
                  Is_Comma_No_Parenthesis_List,
                  Count (Asis.Declarations.Names (Element)));
         when An_In_Out_Mode =>
            Push (": in out ",
                  Is_Comma_No_Parenthesis_List,
                  Count (Asis.Declarations.Names (Element)));
      end case;

      if (Is_Here (Asis.Declarations.Initialization_Expression (Element))) then
         Push (":= ");
         Push (";" & ASCII.CR);
      else
         Push (";" & ASCII.CR);
      end if;
   when A_Formal_Type_Declaration =>                -- 12.5
      Send ("type ");
      if Is_Here (Asis.Declarations.Discriminant_Part (Element)) then
         Push;
      end if;
      Push ("is ");
      Push (";" & ASCII.CR);
      --  --|A2012 start
   when A_Formal_Incomplete_Type_Declaration =>
      null;
      --  --|A2012 end
   when A_Formal_Procedure_Declaration =>           -- 12.6  -> Default_Kinds
      Send ("with procedure ");
      Push;
      case (Asis.Elements.Default_Kind (Element)) is
         when Not_A_Default =>
            Ada.Text_IO.Put ("<<Node Not_A_Default>>");
         when A_Name_Default =>
            Push ("is " ,
                  Is_Semi_Colon_List,
                  Count (Asis.Declarations.Parameter_Profile (Element)));
            Push (";" & ASCII.CR);
         when A_Box_Default =>
            Push ("is <>;" & ASCII.CR,
                  Is_Semi_Colon_List,
                  Count (Asis.Declarations.Parameter_Profile (Element)));
            -- --|Ada2005 start
         when A_Null_Default =>
            null;
            -- --|Ada2005 end
         when A_Nil_Default =>
            Push (";" & ASCII.CR,
                  Is_Semi_Colon_List,
                  Count (Asis.Declarations.Parameter_Profile (Element)));
      end case;
   when A_Formal_Function_Declaration =>            -- 12.6  -> Default_Kinds
      Send ("with function ");
      Push;
      Push ("return ",
            Is_Semi_Colon_List,
            Count (Asis.Declarations.Parameter_Profile (Element)));
      case (Asis.Elements.Default_Kind (Element)) is
         when Not_A_Default =>
            Ada.Text_IO.Put ("<<Node Not_A_Default>>");
         when A_Name_Default =>
            Push ("is ");
            Push (";" & ASCII.CR);
         when A_Box_Default =>
            Push ("is <>;" & ASCII.CR);
            -- --|Ada2005 start
         when A_Null_Default =>
            null;
            -- --|Ada2005 end

         when A_Nil_Default =>
```

```
               Push (";" & ASCII.CR);
            end case;

         when A_Formal_Package_Declaration =>             -- 12.7
            Send ("with package ");
            Push ("is new ");
            Push;
            Push (";" & ASCII.CR,
                  Is_Comma_List,
                  Count (Asis.Declarations.Generic_Actual_Part (Element, false)));
         when A_Formal_Package_Declaration_With_Box =>   -- 12.7
            Send ("with package ");
            Push ("is new ");
            Push ("(<>);" & ASCII.CR);
      end case;
   when A_Definition =>
      case Asis.Elements.Definition_Kind (Element) is
         when Not_A_Definition =>                -- An unexpected element
            Ada.Text_IO.Put ("<<Node Not_A_Definition>>");
         when A_Type_Definition =>               -- 3.2.1   -> Type_Kinds
            case Asis.Elements.Type_Kind (Element) is
               when Not_A_Type_Definition =>
                  Ada.Text_IO.Put ("<<Node Not_A_Type_Definition>>");
               when A_Derived_Type_Definition =>
                  Send ("new ");
                  Push;
               when A_Derived_Record_Extension_Definition =>
                  Send (Trait_String & "new ");
                  if  Asis.Elements.Definition_Kind (
                                                    Asis.Definitions.Record_Definition (Element)) =
                    A_Null_Record_Definition  then
                     Push ("with ");
                     Push;
                  else
                     Push ("with record" & ASCII.CR);
                     Push ("end record ");
                     Indent;
                  end if;
               when An_Enumeration_Type_Definition =>
                  Push ("",
                        Is_Comma_List,
                        Count (Asis.Definitions.Enumeration_Literal_Declarations (Element)));
                  Check_If_Return_Separator
                    (Asis.Definitions.Enumeration_Literal_Declarations (Element));
               when A_Signed_Integer_Type_Definition =>
                  Send ("range ");
                  Push;
               when A_Modular_Type_Definition =>
                  Send ("mod ");
                  Push;
               when A_Root_Type_Definition =>
                  Ada.Text_IO.Put ("<<Node A_Root_Type_Definition>>");
               when A_Floating_Point_Definition =>
                  Send ("digits ");
                  if Is_Here (Asis.Definitions.Real_Range_Constraint (Element))
                  then
                     Push ("range ");
                     Push;
                  else
                     Push;
                  end if;
               when An_Ordinary_Fixed_Point_Definition =>
                  Send ("delta ");
                  Push ("range ");
                  Push;
               when A_Decimal_Fixed_Point_Definition =>
                  Send ("delta ");
                  Push ("digits ");
                  if Is_Here (Asis.Definitions.Real_Range_Constraint (Element))
                  then
                     Push ("range ");
                     Push;
                  else
                     Push;
                  end if;
               when An_Unconstrained_Array_Definition =>
                  Send ("array ");
                  Push ("of ",
                        Is_Comma_Range_List,
                        Count (Asis.Definitions.Index_Subtype_Definitions (Element)));
                  Push;
               when A_Constrained_Array_Definition =>
                  Send ("array ");
                  Push ("of ",
                        Is_Comma_List,
                        Count (Asis.Definitions.Discrete_Subtype_Definitions (Element)));
                  Push;
               when A_Record_Type_Definition =>
                  if  Asis.Elements.Definition_Kind (
                                                    Asis.Definitions.Record_Definition (Element)) =
                    A_Null_Record_Definition  then
                     Send (Trait_String);
                     Push;
                  else
                     Send (Trait_String & "record" & ASCII.CR);
                     Push ("end record ");
                     Indent;
                  end if;
               when A_Tagged_Record_Type_Definition =>
                  if  Asis.Elements.Definition_Kind
                    (Asis.Definitions.Record_Definition (Element)) =
                    A_Null_Record_Definition
                  then
                     Send (Trait_String ("tagged "));
                     Push;
                  else
                     Send (Trait_String ("tagged ") & "record" & ASCII.CR);
                     Push ("end record ");
                     Indent;
```

```
         end if;
      when An_Access_Type_Definition =>
         case Asis.Elements.Access_Type_Kind (Element) is
            when Not_An_Access_Type_Definition =>
               Ada.Text_IO.Put ("<<Node Not_An_Access_Type_Definition>>");
            when A_Pool_Specific_Access_To_Variable =>
               Send ("access ");
               Push;
            when An_Access_To_Constant =>
               Send ("access constant ");
               Push;
            when An_Access_To_Variable =>
               Send ("access all ");
               Push;
            when An_Access_To_Procedure =>
               Send ("access procedure ");
               Push ("",
                     Is_Semi_Colon_List,
                     Count (Asis.Definitions.Access_To_Subprogram_Parameter_Profile (Element)));
            when An_Access_To_Protected_Procedure =>
               Send ("access protected procedure ");
               Push ("",
                     Is_Semi_Colon_List,
                     Count (Asis.Definitions.Access_To_Subprogram_Parameter_Profile (Element)));
            when An_Access_To_Function =>
               Send ("access function ");
               Push ("return ",
                     Is_Semi_Colon_List,
                     Count (Asis.Definitions.Access_To_Subprogram_Parameter_Profile (Element)));
               Push;
            when An_Access_To_Protected_Function =>
               Send ("access protected function ");
               Push ("return ",
                     Is_Semi_Colon_List,
                     Count (Asis.Definitions.Access_To_Subprogram_Parameter_Profile (Element)));
               Push;
         end case;
         --  --|A2005 start
      when An_Interface_Type_Definition =>          -- 3.9.4      -> Interface_Kinds
         null;
         --  --|A2005 end

   end case;
when A_Subtype_Indication =>            -- 3.2.2
   declare
      M : Asis.Element := Asis.Definitions.Subtype_Constraint (Element);
   begin
      if Is_Here (M)
      then
         case Asis.Elements.Constraint_Kind (M) is
            when A_Range_Attribute_Reference |
                 A_Simple_Expression_Range =>
               Push ("range ");
               Push;
            when others =>
               Push;
               Push;
         end case;
      else
         Push;
      end if;
   end;
when A_Constraint =>                    -- 3.2.2   -> Constraint_Kinds
   case Asis.Elements.Constraint_Kind (Element) is
      when Not_A_Constraint =>
         Ada.Text_IO.Put ("<<Node Not_A_Constraint>>");
      when A_Range_Attribute_Reference =>
         -- Send ("range "); -- No range to Ada.Text_IO.Put ...
         Push;
      when A_Simple_Expression_Range =>
         -- Send ("range "); -- Ada.Text_IO.Put a range here might fail...
         Push (".. ");
         Push;
      when A_Digits_Constraint =>
         Send ("digits ");
         if (Asis.Elements.Constraint_Kind
             (Asis.Definitions.Real_Range_Constraint (Element)) /=
               Not_A_Constraint) then
            Push ("range ");
            Push;
         else
            Push;
         end if;
      when A_Delta_Constraint =>
         Send ("delta ");
         if (Asis.Elements.Constraint_Kind
             (Asis.Definitions.Real_Range_Constraint (Element)) /=
               Not_A_Constraint) then
            Push ("range ");
            Push;
         else
            Push;
         end if;
      when An_Index_Constraint =>
         Push ("",
               Is_Comma_List,
               Count (Asis.Definitions.Discrete_Ranges (Element)));
      when A_Discriminant_Constraint =>
         Push ("",
               Is_Comma_List,
               Count (Asis.Definitions.Discriminant_Associations (Element, False)));
   end case;
when A_Component_Definition =>          -- 3.6       -> Trait_Kinds
   Send (Trait_String);
   Push;
when A_Discrete_Subtype_Definition |   -- 3.6     -> Discrete_Range_Kinds
     A_Discrete_Range =>                -- 3.6.1   -> Discrete_Range_Kinds
   case Asis.Elements.Discrete_Range_Kind (Element) is
      when Not_A_Discrete_Range =>
```

```
         Ada.Text_IO.Put ("<<Node Not_A_Discrete_Range>>");
      when A_Discrete_Subtype_Indication =>
         declare
            C : Asis.Element := Asis.Definitions.Subtype_Constraint (Element);
         begin
            if Is_Here (C)
            then
               case Asis.Elements.Constraint_Kind (C) is
                  when A_Range_Attribute_Reference |
                       A_Simple_Expression_Range =>
                     Push ("range ");
                  when others =>
                     Push;
               end case;
            end if;
            Push;
         end;
      when A_Discrete_Range_Attribute_Reference =>
         Push;
      when A_Discrete_Simple_Expression_Range =>
         Push (".. ");
         Push;
   end case;
when An_Unknown_Discriminant_Part =>    -- 3.7
   Send ("(<>) ");
when A_Known_Discriminant_Part =>       -- 3.7
   Push ("",
         Is_Semi_Colon_List,
         Count (Asis.Definitions.Discriminants (Element)));
when A_Record_Definition =>             -- 3.8
   Push ("",
         Not_In_A_List,
         Count (Asis.Definitions.Record_Components (Element)));
when A_Null_Record_Definition =>        -- 3.8
   Send ("null record ");
when A_Null_Component =>                -- 3.8
   Send ("null;" & ASCII.CR);
when A_Variant_Part =>                  -- 3.8
   Send ("case ");
   Push ("is" & ASCII.CR);
   Push ("end case;" & ASCII.CR,
         Not_In_A_List,
         Count (Asis.Definitions.Variants (Element, true)));
   Indent;
when A_Variant =>                       -- 3.8
   Send ("when ");
   Push ("=>" & ASCII.CR,
         Is_Vertical_Line_List,
         Count (Asis.Definitions.Variant_Choices (Element)));
   Push ("",
         Not_In_A_List,
         Count (Asis.Definitions.Record_Components (Element, true)));
   Indent;
when An_Others_Choice =>                -- 3.8.1, 4.3.1, 4.3.3, 11.2
   Send ("others ");
   --  --|A2005 start
when An_Access_Definition =>             -- 3.10(6/2)   -> Access_Definition_Kinds
   null;
   --  --|A2005 end
when A_Private_Type_Definition =>       -- 7.3     -> Trait_Kinds
   Send (Trait_String);
when A_Tagged_Private_Type_Definition =>-- 7.3     -> Trait_Kinds
   Send (Trait_String ("tagged "));
when A_Private_Extension_Definition =>  -- 7.3     -> Trait_Kinds
   Push ("with private ");
when A_Task_Definition |                  -- 9.1
     A_Protected_Definition =>              -- 9.4
   L := Count (Asis.Definitions.Private_Part_Items (Element, True));
   if L /= 0
   then
      Push ("private" & ASCII.CR,
            Not_In_A_List,
            Count (Asis.Definitions.Visible_Part_Items (Element, True)));
      Indent;
      Push ("end ",
            Not_In_A_List,
            L);
      Indent;
   else
      Push ("end ",
            Not_In_A_List,
            Count (Asis.Definitions.Visible_Part_Items (Element, True)));
      Indent;
   end if;

when A_Formal_Type_Definition =>       -- 12.5    -> Formal_Type_Kinds
   case Asis.Elements.Formal_Type_Kind (Element) is
      when Not_A_Formal_Type_Definition =>
         Ada.Text_IO.Put ("<<Node Not_A_Formal_Type_Definition>>");
      when A_Formal_Private_Type_Definition =>
         Send (Trait_String);
      when A_Formal_Tagged_Private_Type_Definition =>
         Send (Trait_String ("tagged "));
      when A_Formal_Discrete_Type_Definition =>
         Send ("(<>) ");
      when A_Formal_Signed_Integer_Type_Definition =>
         Send ("range <> ");
      when A_Formal_Modular_Type_Definition =>
         Send ("mod <> ");
      when A_Formal_Floating_Point_Definition =>
         Send ("digits <> ");
      when A_Formal_Ordinary_Fixed_Point_Definition =>
         Send ("delta <> ");
      when A_Formal_Decimal_Fixed_Point_Definition =>
         Send ("delta <> digits <> ");
         --  --|A2005 start
      when A_Formal_Interface_Type_Definition =>      -- 12.5.5(2) -> Interface_Kinds
         null;
         --  --|D2005 start
```

```
               --  Do we really need this value in Formal_Type_Kinds? There is no
               --  difference between it and An_Interface_Type_Definition. The only
               --  reason to have it is not to break the ASIS 95 idea tp have separate
               --  values representing definitions of formal kinds
               --  --|D2005 end

               --  --|A2005 end

            when A_Formal_Derived_Type_Definition =>
               case Asis.Elements.Trait_Kind (Element) is
                  when An_Abstract_Private_Trait =>
                     Send ("abstract new ");
                     Push ("with private ");
                  when An_Abstract_Trait =>
                     Send ("abstract new ");
                     Push;
                  when A_Private_Trait =>
                     Send ("new ");
                     Push ("with private ");
                  when others =>
                     Send ("new ");
                     Push;
               end case;
            when A_Formal_Unconstrained_Array_Definition =>
               Send ("array ");
               Push ("of ",
                     Is_Comma_Range_List,
                     Count (Asis.Definitions.Index_Subtype_Definitions (Element)));
               Push;
            when A_Formal_Constrained_Array_Definition =>
               Send ("array ");
               Push ("of ",
                     Is_Comma_List,
                     Count (Asis.Definitions.Discrete_Subtype_Definitions (Element)));
               Push;
            when A_Formal_Access_Type_Definition =>
               case Asis.Elements.Access_Type_Kind (Element) is
                  when Not_An_Access_Type_Definition =>
                     Ada.Text_IO.Put ("<<Node Not_An_Access_Type_Definition>>");
                  when A_Pool_Specific_Access_To_Variable =>
                     Send ("access ");
                     Push;

                  when An_Access_To_Constant =>
                     Send ("access constant ");
                     Push;
                  when An_Access_To_Variable =>
                     Send ("access all ");
                     Push;
                  when An_Access_To_Procedure =>
                     Send ("access procedure ");
                     Push ("",
                           Is_Semi_Colon_List,
                           Count (Asis.Definitions.Access_To_Subprogram_Parameter_Profile (Element)));
                  when An_Access_To_Protected_Procedure =>
                     Send ("access protected procedure ");
                     Push ("",
                           Is_Semi_Colon_List,
                           Count (Asis.Definitions.Access_To_Subprogram_Parameter_Profile (Element)));
                  when An_Access_To_Function =>
                     Send ("access function ");
                     Push ("return ",
                           Is_Semi_Colon_List,
                           Count (Asis.Definitions.Access_To_Subprogram_Parameter_Profile (Element)));
                     Push;
                  when An_Access_To_Protected_Function =>
                     Send ("access protected function ");
                     Push ("return ",
                           Is_Semi_Colon_List,
                           Count (Asis.Definitions.Access_To_Subprogram_Parameter_Profile (Element)));
                     Push;
               end case;
         end case;
         --  --|A2012 start
      when An_Aspect_Specification =>         -- 13.3.1
         null;
         --  --|A2012 end

   end case;
when An_Expression =>
   if (not Gela_Ids.Is_Nil(Conversion_ID)) and
     Gela_Ids.Is_Equal(Conversion_ID, Gela_Ids.Create_ID(Element)) then
      Send(To_S(Conversion_Name));
      --    Put_Line(To_String(Asis.Elements.Debug_Image(Element)));
      Commit;
      Is_Send_Lock := True;
   end if;
   if Gela_Ids.Is_Equal(Assignment_ID,
                        Gela_Ids.Create_ID(Element)) then
      Is_Send_Lock := True;
   end if;

   if not Is_Send_Lock then

      case Asis.Elements.Expression_Kind (Element) is
      when Not_An_Expression =>                       -- An unexpected element
         Ada.Text_IO.Put ("<<Node Not_An_Expression>>");
      when An_Integer_Literal |                       -- 2.4
           A_Real_Literal |                           -- 2.4.1
           A_String_Literal =>                        -- 2.6

         Send (To_String (Asis.Expressions.Value_Image (Element)) & " ");

      when An_Identifier =>

         declare
            use Variable_Analyzer;
            Tmp_Use_Variable : Use_Variable_Link;
         begin
```

```
      begin
         Tmp_Use_Variable
           := Use_Variable_List.Search_Node
             (Current_Unit.Used_Variables,
              Gela_Ids.Create_ID
                (Asis.Expressions.Corresponding_Name_Definition
                   (Element)));
      exception
         when Asis.Exceptions.Asis_Failed =>
            Tmp_Use_Variable := null;
         when ASIS.EXCEPTIONS.ASIS_INAPPROPRIATE_ELEMENT =>
            Tmp_Use_Variable := null;
      end;

      if Tmp_Use_Variable /= null
        and then Tmp_Use_Variable.Is_Refer then
         Send(To_S(Tmp_Use_Variable.Proxy_Name & ".Mutex.Get"));
      else
         if Is_Protected_Type_Identifier then
            Send (To_String (Asis.Expressions.Name_Image (Element)) & Protected_Identifier & " ");
            Is_Protected_Type_Identifier := False;
         else
            Send (Protected_Call_Ident & To_String (Asis.Expressions.Name_Image (Element)) & " ");
         end if;
      end if;
   end;
   --      put_Line("####");
   case To_Ada12_Identifier(Asis.Expressions.Name_Image(Element)) is
   when Synchronized_Queue_Interfaces =>
      --                  Send(To_S(Tmp_Use_Variable.Proxy_Name) &"*****");
      Q.Queue_Entry("Q1");

   when Bounded_Synchronized_Queues =>
      null;
   when others =>
      null;
   end case;
   Q.Queue_Entry("Q1");
   declare
      subtype AAA_Type is Asis.Name_List;
      AAA : AAA_Type := Nil_Element;
      begin
      Asis.Expressions.References(Element, Element, False);
   end;
when A_Character_Literal |
     An_Enumeration_Literal =>
   Send (To_String (Asis.Expressions.Name_Image (Element)) & " ");
when An_Operator_Symbol =>
   if Is_Infix
   then
      case Asis.Elements.Operator_Kind (Element) is
      when Not_An_Operator =>
         Ada.Text_IO.Put ("<<Node Not_An_Operator>>");

      when A_Unary_Plus_Operator |
           A_Unary_Minus_Operator =>
         Send (Function_Call_Operator);

      when An_Abs_Operator |
           A_Not_Operator =>
         Send (Function_Call_Operator & " ");
      when others =>
         Push (Function_Call_Operator & " ");
      end case;
   else
      Send (To_String (Asis.Expressions.Name_Image (Element)) & " ");
   end if;

when An_Explicit_Dereference =>                 -- 4.1
   Push (".all ");
   No_Space;
when A_Function_Call =>                         -- 4.1
   --       Put_Line(To_String(Asis.Elements.Debug_Image(Asis.Expressions.Prefix(Element))));
   --       Put_Line(To_String(Asis.Text.Element_Image(Asis.Expressions.Prefix(Element))));
   --       Put_Line(To_String(Gela_Ids.Debug_Image(Gela_Ids.Create_Id(Element))));
   -- If it is an operator, we print it here, so the element Operator
   -- won't have to do it.
   if Asis.Expressions.Is_Prefix_Call (Element)
   then
      Push;
      Push ("",
            Is_Comma_List,
            Count (Asis.Expressions.Function_Call_Parameters (Element)));
      begin
         declare
            use Function_Analyzer;
            Tmp_Use_Function : Use_Function_Link
              := Use_Function_List.Search_Node
                (Current_Unit.Used_Functions,
                 Gela_Ids.Create_ID
                   (Asis_Utils.Corresponding_Called_Function(Element)));
         begin
            if Tmp_Use_Function /= null then
               Conversion_ID := Gela_Ids.Create_ID
                 (Asis.Expressions.Prefix(Element));
               Conversion_Name := Tmp_Use_Function.Proxy_Name;
            end if;
         end;
      exception
         when ASIS.EXCEPTIONS.ASIS_INAPPROPRIATE_ELEMENT =>
            null;

      end;
   else
      case Asis.Elements.Operator_Kind
        (Asis.Expressions.Prefix (Element)) is
      when Not_An_Operator =>
         Ada.Text_IO.Put ("<<Node Not_An_Operator>>");
      when An_And_Operator |
```

```
            An_Or_Operator |
            An_Xor_Operator |
            An_Equal_Operator |
            A_Not_Equal_Operator |
            A_Less_Than_Operator |
            A_Less_Than_Or_Equal_Operator |
            A_Greater_Than_Operator |
            A_Greater_Than_Or_Equal_Operator |
            A_Plus_Operator |
            A_Minus_Operator |
            An_Exponentiate_Operator |
            A_Multiply_Operator |
            A_Divide_Operator |
            A_Mod_Operator |
            A_Concatenate_Operator |
            A_Rem_Operator =>
          Push;
          Infix;
          Push;

       when A_Unary_Plus_Operator |
            A_Unary_Minus_Operator |
            An_Abs_Operator |
            A_Not_Operator =>
          Push;
          Infix;
          Push;
       end case;
   end if;
when An_Indexed_Component =>                      -- 4.1.1
   Push;
   Push ("",
         Is_Comma_List,
         Count (Asis.Expressions.Index_Expressions (Element)));
when A_Slice =>                                   -- 4.1.2
   Push ("(");
   Push (") ");
when A_Selected_Component =>                      -- 4.1.3
   declare
      use Variable_Analyzer;
      Tmp_Use_Variable : Use_Variable_Link := null;
      Ident : Asis.Expression := Asis_Utils.Get_Identifier
        (Element);
   begin
      if not Asis.Elements.Is_Nil(Ident) then
         Tmp_Use_Variable :=
           Use_Variable_List.Search_Node
             (Current_Unit.Used_Variables,
              Gela_Ids.Create_ID
                (Asis.Expressions.Corresponding_Name_Definition
                   (Ident)));
      end if;
      if Tmp_Use_Variable /= null
        and then Tmp_Use_Variable.Is_Refer then
         Send(To_S(Tmp_Use_Variable.Proxy_Name & ".Get"));
         Commit;
         Is_Send_Lock := True;
         Refered_ID := Gela_Ids.Create_ID(Element);
      else
         Push (".");
         No_Space;
         Push;
      end if;
   end;
when An_Attribute_Reference =>                    -- 4.1.4  -> Attribute_Kinds
   case Asis.Elements.Attribute_Kind (Element) is
   when Not_An_Attribute =>
      Ada.Text_IO.Put ("<<Node Not_An_Attribute>>");
   when A_First_Attribute |
        A_Last_Attribute |
        A_Length_Attribute |
        A_Range_Attribute |
        An_Implementation_Defined_Attribute |
        An_Unknown_Attribute =>
      Push ("'");
      No_Space;
      Push;
      Push ("",
            Is_Comma_List,
            Count (Asis.Expressions.Attribute_Designator_Expressions (Element)));
   when others =>
      Push ("'");
      No_Space;
      Push;
   end case;
when A_Record_Aggregate =>                        -- 4.3
   if Count (Asis.Expressions.Record_Component_Associations (Element, False)) = 0
   then
      Send ("(null record)");
   else
      Push ("",
            Is_Comma_List,
            Count (Asis.Expressions.Record_Component_Associations (Element, False)));
   end if;
when An_Extension_Aggregate =>                    -- 4.3
   if Count (Asis.Expressions.Record_Component_Associations (Element, False)) = 0
   then
      Send ("(");
      Push ("with null record)");
   else
      Send ("(");
      Push ("with ");
      Push (")",
            Is_Comma_No_Parenthesis_List,
            Count (Asis.Expressions.Record_Component_Associations (Element, False)));
   end if;
when A_Positional_Array_Aggregate |               -- 4.3 -- corrected in ASIS-GNAT
     A_Named_Array_Aggregate =>                   -- 4.3 -- corrected in ASIS-GNAT
   Push ("",
```

```
               Is_Comma_List,
               Count (Asis.Expressions.Array_Component_Associations (Element)));
      when An_And_Then_Short_Circuit =>                -- 4.4
         Push ("and then ");
         Push;
      when An_Or_Else_Short_Circuit =>                 -- 4.4
         Push ("or else ");
         Push;
      when An_In_Membership_Test =>              -- 4.4
         Push ("in ");
         Push;
      when A_Not_In_Membership_Test =>           -- 4.4
         Push ("not in ");
         Push;
      when A_Null_Literal =>                           -- 4.4
         Send ("null ");
      when A_Parenthesized_Expression =>               -- 4.4
         Send ("(");
         Push (") ");
      when A_Type_Conversion =>                        -- 4.6
         Push ("(");
         Push  (") ");
      when A_Qualified_Expression =>                   -- 4.7
         Push("'");
         No_Space;
         Push;
      when An_Allocation_From_Subtype =>               -- 4.8
         Send ("new ");
         Push;
      when An_Allocation_From_Qualified_Expression => -- 4.8
         Send ("new ");
         Push;
         -- --|A2012 start
      when A_Case_Expression =>                        -- Ada 2012
         null;
      when An_If_Expression =>                         -- Ada 2012
         null;
      when A_For_All_Quantified_Expression =>          -- Ada 2012
         null;
      when A_For_Some_Quantified_Expression =>         -- Ada 2012
         null;
         -- --|A2012 end
      end case;
   end if;
when An_Association =>
   case Asis.Elements.Association_Kind (Element) is
      when Not_An_Association =>                   -- An unexpected element
         Ada.Text_IO.Put ("<<Node Not_An_Association>>");
      when A_Discriminant_Association =>           -- 3.7.1
         L := Count (Asis.Expressions.Discriminant_Selector_Names (Element));
         if L /= 0
         then
            Push ("=> ",
                  Is_Vertical_Line_List,
                  L);
         end if;
         Push;
      when A_Record_Component_Association =>       -- 4.3.1
         L := Count (Asis.Expressions.Record_Component_Choices (Element));
         if L /= 0
         then
            Push ("=> ",
                  Is_Vertical_Line_List,
                  L);
         end if;
         Push;
      when An_Array_Component_Association =>       -- 4.3.3
         L := Count (Asis.Expressions.Array_Component_Choices (Element));
         if L /= 0
         then
            Push ("=> ",
                  Is_Vertical_Line_List,
                  L);
         end if;
         Push;
      when A_Parameter_Association |              -- 6.4
           A_Pragma_Argument_Association |          -- 2.8
           A_Generic_Association =>                 -- 12.3
         declare
            Pname : V_String := Convert_List.Search_Node
              (Aconv_List,
               Gela_Ids.Create_ID(Element));
         begin
            if Pname = Null_Str then
               if Is_Here (Asis.Expressions.Formal_Parameter (Element))
               then
                  Push ("=> ");
               end if;
               Push;
            else
               Send(To_S(Pname));
               Commit;
               Is_Send_Lock := True;
            end if;
         end;
   end case;
when A_Statement =>
   if Current_Unit.Is_In_MH then
      Control := Abandon_Children;
      Send (To_S(Null_Str));
   else

      case Asis.Elements.Statement_Kind (Element) is
      when Not_A_Statement =>                      -- An unexpected element
         Ada.Text_IO.Put ("<<Node Not_A_Statement>>");
      when A_Null_Statement =>                     -- 5.1
         Send_Label ("null;" & ASCII.CR);
      when An_Assignment_Statement =>              -- 5.2
         declare
```

```
      use Variable_Analyzer;
      Ident : Asis.Expression := Asis_Utils.Get_Identifier
        (Asis.Statements.Assignment_Variable_Name(Element)
        );
      Tmp_Use_Variable : Use_Variable_Link := null;
   begin
      if not Asis.Elements.Is_Nil(Ident) then
         Tmp_Use_Variable := Use_Variable_List.Search_Node
           (Current_Unit.Used_Variables,
            Gela_Ids.Create_ID
              (Asis.Expressions.Corresponding_Name_Definition
                 (Ident)));
      end if;
      if Tmp_Use_Variable /= null
        and then Tmp_Use_Variable.Is_Assign then
         Send_Label(To_S(Tmp_Use_Variable.Proxy_Name & ".Mutex.Set"));
         Push;
         Push(";" & ASCII.CR, Is_Comma_List, 1);
         Assignment_ID :=
           Gela_Ids.Create_ID
             (Asis.Statements.Assignment_Variable_Name(Element));
      else
         Send_Label ("");
         Push (":= ");
         Push (";" &  ASCII.CR);
      end if;
   end;
when An_If_Statement =>                     -- 5.3
   Send_Label ("");
   Push ("end if;" & ASCII.CR,
         Not_In_A_List,
         Count (Asis.Statements.Statement_Paths (Element)));
when A_Case_Statement =>                    -- 5.4
   Send_Label ("case ");
   Push ("is" & ASCII.CR);
   Push ("end case;" & ASCII.CR,
         Not_In_A_List,
         Count (Asis.Statements.Statement_Paths (Element)));
   Indent;
when A_Loop_Statement =>             -- 5.55
   if Is_Here (Asis.Statements.Statement_Identifier (Element))
   then
      Send_Label (Statement_Identifier_Label);
      Push (":" & ASCII.CR & "loop" & ASCII.CR);
      No_Space;
      Push ("end loop " &
          To_String (Asis.Declarations.Defining_Name_Image
          (Asis.Statements.Statement_Identifier (Element))) &
          ";" & ASCII.CR & Statement_Completion_Label,
        Not_In_A_List,
        Count (Asis.Statements.Loop_Statements (Element, True)));
      Indent;
   else
      Send_Label ("loop" & ASCII.CR);
      Push ("end loop;" & ASCII.CR,
            Not_In_A_List,
            Count (Asis.Statements.Loop_Statements (Element, True)));
      Indent;
   end if;
when A_While_Loop_Statement =>         -- 5.5
   if Is_Here (Asis.Statements.Statement_Identifier (Element))
   then
      Send_Label (Statement_Identifier_Label);
      Push (":" & ASCII.CR & "while ");
      No_Space;
      Push ("loop" & ASCII.CR);
      Push ("end loop " &
          To_String (Asis.Declarations.Defining_Name_Image
          (Asis.Statements.Statement_Identifier (Element))) &
          ";" & ASCII.CR & Statement_Completion_Label,
        Not_In_A_List,
        Count (Asis.Statements.Loop_Statements (Element, True)));
      Indent;
   else
      Send_Label ("while ");
      Push ("loop" & ASCII.CR);
      Push ("end loop;" & ASCII.CR,
            Not_In_A_List,
            Count (Asis.Statements.Loop_Statements (Element, True)));
      Indent;
   end if;
when A_For_Loop_Statement =>         -- 5.5
   if Is_Here (Asis.Statements.Statement_Identifier (Element))
   then
      Send_Label (Statement_Identifier_Label);
      Push (":" & ASCII.CR & "for ");
      No_Space;
      Push ("loop" & ASCII.CR);
      Push ("end loop " &
          To_String (Asis.Declarations.Defining_Name_Image
          (Asis.Statements.Statement_Identifier (Element))) &
          ";" & ASCII.CR & Statement_Completion_Label,
        Not_In_A_List,
        Count (Asis.Statements.Loop_Statements (Element, True)));
      Indent;
   else
      Send_Label ("for ");
      Push ("loop" & ASCII.CR);
      Push ("end loop;" & ASCII.CR,
            Not_In_A_List,
            Count (Asis.Statements.Loop_Statements (Element, True)));
      Indent;
   end if;

when A_Block_Statement =>                   -- 5.6
   if Is_Here (Asis.Statements.Statement_Identifier (Element)) then
      -- Name_Handler
      Enter_Unit_Spec(To_V(Asis.Declarations.Defining_Name_Image
        (Asis.Statements.Statement_Identifier (Element))),
```

```
        Element,
        Not_A_Declaration, BLOCKS);
      --  Regist_Name(Gela_Ids.Create_Id(Asis.Declarations.Corresponding_Declaration(Element)));
      -- Name_Handler
      Send_Label (Statement_Identifier_Label);
      if Asis.Statements.Is_Declare_Block (Element) then
         Push (":" & ASCII.CR & Declare_Label & ASCII.CR);
         No_Space;
         --                      Push ("begin" & ASCII.CR ,
         Push (Make_Proxy_For_Block & Make_Var_For_Block & Begin_Label,
               Not_In_A_List,
               Count (Asis.Statements.Block_Declarative_Items (Element , True)));
         Indent;
      else
         --                      Push (":" & ASCII.CR & "begin" & ASCII.CR);
         Push (":" & ASCII.CR & Begin_Label(False));
         No_Space;
      end if;
   else
      declare

         Block_Label : String := Unique_Identifier;
      begin
         -- Name_Handler
         Enter_Unit_Spec(To_V(Block_Label),
                         Element,
                         Not_A_Declaration, BLOCKS);
         --  Regist_Name(Gela_Ids.Create_Id(Asis.Declarations.Corresponding_Declaration(Element)));
         -- Name_Handler
         if Asis.Statements.Is_Declare_Block (Element) then
            --                      Send_Label ("declare" & ASCII.CR);
            Send_Label (Block_Label & " : " & Declare_Label & ASCII.CR);
            --                      Push ("begin" & ASCII.CR ,
            Push (Begin_Label,
                  Not_In_A_List,
                  Count (Asis.Statements.Block_Declarative_Items (Element, True)));
            Indent;
         else
            --                      Send_Label ("begin" & ASCII.CR);
            Send_Label (Block_Label & ": " & Begin_Label(False));
         end if;
      end;
   end if;
   if (Count (Asis.Statements.Block_Exception_Handlers (Element, True)) /= 0) then
      --                  Push ("exception" & ASCII.CR ,
      Push (Exception_Label,
            Not_In_A_List,
            Count (Asis.Statements.Block_Statements (Element, True)));
      Indent;
      if (Is_Here (Asis.Statements.Statement_Identifier (Element))) then
         Push ("end "
           & To_String (Asis.Declarations.Defining_Name_Image (
             Asis.Statements.Statement_Identifier (Element)))
           & ";" & ASCII.CR & Statement_Completion_Label,
           Not_In_A_List,
           Count (Asis.Statements.Block_Exception_Handlers (Element, True)));
      else
         Push ("end " & To_S(Current_Unit.Name) & ";" & ASCII.CR,
               Not_In_A_List,
               Count (Asis.Statements.Block_Exception_Handlers (Element, True)));
      end if;
      Indent;
   else
      if (Is_Here (Asis.Statements.Statement_Identifier (Element))) then
         --                      Push ("end "
         Push (End_Label
           & To_String (Asis.Declarations.Defining_Name_Image (
             Asis.Statements.Statement_Identifier (Element)))
           & ";" & ASCII.CR  & Statement_Completion_Label,
           Not_In_A_List,
           Count (Asis.Statements.Block_Statements (Element, True)));
      else
         --                      Push ("end;" & ASCII.CR,
         Push (End_Label & " " & To_S(Current_Unit.Name) & ";" & ASCII.CR,
               Not_In_A_List,
               Count (Asis.Statements.Block_Statements (Element, True)));
      end if;
      Indent;
   end if;
when An_Exit_Statement =>                  -- 5.7
   if Is_Here (Asis.Statements.Exit_Loop_Name (Element))
   then
      Send_Label ("exit ");
      if Is_Here (Asis.Statements.Exit_Condition (Element))
      then
         Push ("when ");
      end if;
      Push (";" & ASCII.CR);
   else
      if Is_Here (Asis.Statements.Exit_Condition (Element))
      then
         Send_Label ("exit when ");
         Push (";" & ASCII.CR);
      else
         Send_Label ("exit;" & ASCII.CR);
      end if;
   end if;

when A_Goto_Statement =>                   -- 5.8
   Send_Label ("goto ");
   Push (";" & ASCII.CR);
when An_Entry_Call_Statement =>        -- 9.5.3
   --              Send_Label ("");
   if Is_Procedure(Element) then
      Process_Procedure;
   else
      if Is_Protected(Asis.Statements.Corresponding_Called_Entity(Element)) then
         if not Timed_Entry_Mode then
            Send_Label (To_S(Designated_Probes(PROTECTED_ENTRY_CALL, To_V(Current_Unit_Name), False)));
```

```
         else
            Send_Label ("");
            Timed_Entry_Mode := False;
         end if;
         Push;
         Push (";" & ASCII.CR & To_S(Designated_Probes(PROTECTED_ENTRY_CALL_COMPLETION, To_V(Current_Unit_Name))),
               Is_Comma_List,
               Count (Asis.Statements.Call_Statement_Parameters (Element, False)));
         Set_Conv_Flag;
      else
         if not Timed_Entry_Mode then
            Send_Label (To_S(Designated_Probes(SIMPLE_ENTRY_CALL, To_V(Current_Unit_Name), False)));
         else
            Send_Label ("");
            Timed_Entry_Mode := False;
         end if;
         Push;
         Push (";" & ASCII.CR & To_S(Designated_Probes(SIMPLE_ENTRY_CALL_COMPLETION, To_V(Current_Unit_Name))),
               Is_Comma_List,
               Count (Asis.Statements.Call_Statement_Parameters (Element, False)));
      end if;
   end if;
when A_Procedure_Call_Statement =>
   Process_Procedure;
when A_Return_Statement =>                -- 6.5
   if Is_Here (Asis.Statements.Return_Expression (Element))
   then
      --                  Send_Label ("return ");
      Send_Label (Return_Label);
      Push (";" & ASCII.CR);
   else
      --                  Send_Label ("return;" & ASCII.CR);
      Send_Label (Return_Label & ";" & ASCII.CR);
   end if;
   --  --|A2005 start
when An_Extended_Return_Statement =>        -- 6.5
   null;
   --  --|A2005 end
when An_Accept_Statement =>               -- 9.5.2
   declare
      Is_Selective : Boolean
        := Selective_Accept_List.Search_Node(Selective_Accept_Flags, Gela_Ids.Create_Id(Element));
   begin
      if not Is_Selective then
         Send_Label (To_S(Designated_Probes(ACCEPT_START, To_V(Current_Unit_Name), False)) & "accept ");
      else
         Send_Label ("accept ");
      end if;
   end;
   if Is_Here (Asis.Statements.Accept_Entry_Index (Element))
   then
      Push ("(");
      Push (") ");
   else
      Push;
   end if;
   L := Count (Asis.Statements.Accept_Body_Statements (Element, True));
   M := Count (Asis.Statements.Accept_Body_Exception_Handlers (Element, True));
   if L = 0 -- if L = 0 then M = 0 too ..
   then
      --                  Push (";" & ASCII.CR ,
      Push (Do_Or_No_Label ,
            Is_Semi_Colon_List,
            --                        Is_Comma_List,
            Count (Asis.Statements.Accept_Parameters (Element)));
   else
      --                  Push ("do" & ASCII.CR,
      Push (Do_Label,
            Is_Semi_Colon_List,
            --                        Is_Comma_List,
            Count (Asis.Statements.Accept_Parameters (Element)));
      if M = 0
      then
         --                     Push ("end " &
         Push (End_Label_In_Accept &
             To_String (Asis.Expressions.Name_Image
             (Asis.Statements.Accept_Entry_Direct_Name (Element))) &
             ";" & ASCII.CR,
           Not_In_A_List,
           L);
         Indent;
      else
         --                     Push ("exception" & ASCII.CR,
         Push (Exception_Label_In_Accept & ASCII.CR,
               Not_In_A_List,
               L);
         Indent;
         Push ("end " &
           --                         Asis.Declarations.Defining_Name_Image
             To_String (Asis.Expressions.Name_Image
             (Asis.Statements.Accept_Entry_Direct_Name (Element))) &
             ";" & ASCII.CR,
           Not_In_A_List,
           M);
         Indent;
      end if;
   end if;
when A_Requeue_Statement =>          -- 9.5.4
   if Is_Protected(Asis.Statements.Corresponding_Called_Entity
                   (Element)) then
      Send_Label(To_S(Designated_Probes
        (PROTECTED_REQUEUE_START,
           To_V(Current_Unit_Name), False)) &
         "requeue ");
   else
      Send_Label(To_S(Designated_Probes
        (REQUEUE_START,
           To_V(Current_Unit_Name), False)) &
         "requeue ");
```

```
   end if;
   --              Send_Label ("requeue ");
   Push (";" & ASCII.CR);
when A_Requeue_Statement_With_Abort =>     -- 9.5.4
   if Is_Protected(Asis.Statements.Corresponding_Called_Entity
                   (Element)) then
      Send_Label(To_S(Designated_Probes
        (PROTECTED_REQUEUE_WITH_ABORT_START,
           To_V(Current_Unit_Name), False)) &
         "requeue ");
   else
      Send_Label(To_S(Designated_Probes
        (REQUEUE_WITH_ABORT_START,
           To_V(Current_Unit_Name), False)) &
         "requeue ");
   end if;
   Send_Label ("requeue ");
   Push ("with abort;" & ASCII.CR);
when A_Delay_Until_Statement =>           -- 9.6
   Send_Label ("delay until ");
   Push (";" & ASCII.CR);
when A_Delay_Relative_Statement =>        -- 9.6
   Send_Label ("delay ");
   Push (";" & ASCII.CR);
when A_Terminate_Alternative_Statement =>  -- 9.7.1
   Send_Label ("terminate;" & ASCII.CR);
when A_Selective_Accept_Statement =>     -- 9.7.2
   declare
      -- terminate
      Terminate_Flag : Boolean := False;
      Plist : Asis.Path_List := Asis.Statements.Statement_Paths(Element);
      Tmp_Stmt : Asis.Statement;
      Tmp_Label : V_String := Null_Str;
   begin

      for I in Plist'RANGE loop
         if Asis.Elements.Statement_Kind(Asis.Statements.Sequence_Of_Statements(Plist(I))(1)) = A_Terminate_Alternative_Statement then
            Terminate_Flag := True;
         end if;
      end loop;
      -- accept statement
      for I in Plist'RANGE loop
         Tmp_Stmt := Asis.Statements.Sequence_Of_Statements(Plist(I))(1);
         if Asis.Elements.Statement_Kind(Tmp_Stmt) = An_Accept_Statement then
            -- selective accept
            Selective_Accept_List.Set_Node(Selective_Accept_Flags,
                                           Gela_Ids.Create_Id(Tmp_Stmt),
                                           True);


            Tmp_Guard := Asis.Statements.Guard(Plist(I));
            -- terminate
            if Terminate_Flag then
               Tmp_Label := Tmp_Label & Designated_Probes(ACCEPT_START_WITH_TERMINATE, To_V(Current_Unit_Name), False, Tmp_Stmt);
            else
               Tmp_Label := Tmp_Label & Designated_Probes(ACCEPT_START, To_V(Current_Unit_Name), False, Tmp_Stmt);
            end if;

            -- accpet に対して、selective フラグをたてる
         end if;
      end loop;
      Send_Label (To_S(Tmp_Label) & "select" & ASCII.CR);
   end ;
   Push ("end select;" & ASCII.CR,
         Not_In_A_List,
         Count (Asis.Statements.Statement_Paths (Element)));
   Indent;
   --       Selective_Accept_Mode := True;
when A_Conditional_Entry_Call_Statement => -- 9.7.3
   --              Send_Label ("select" & ASCII.CR);
   if Is_Protected(Asis.Statements.Sequence_Of_Statements(Asis.Statements.Statement_Paths(Element)(1))(1)) then
      Send_Label (To_S(Designated_Probes(CONDITIONAL_PROTECTED_ENTRY_CALL, To_V(Current_Unit_Name), False)) & "select" & ASCII.CR);
   else
      Send_Label (To_S(Designated_Probes(CONDITIONAL_ENTRY_CALL, To_V(Current_Unit_Name), False)) & "select" & ASCII.CR);
   end if;

   Push ("end select;" & ASCII.CR,
         Not_In_A_List,
         Count (Asis.Statements.Statement_Paths (Element)));
   Indent;
when A_Timed_Entry_Call_Statement =>        -- 9.7.3
   if Is_Protected(Asis.Statements.Sequence_Of_Statements(Asis.Statements.Statement_Paths(Element)(1))(1)) then
      Send_Label (To_S(Designated_Probes(TIMED_PROTECTED_ENTRY_CALL, To_V(Current_Unit_Name), False)) & "select" & ASCII.CR);
   else
      Send_Label (To_S(Designated_Probes(TIMED_ENTRY_CALL, To_V(Current_Unit_Name), False)) & "select" & ASCII.CR);
   end if;

   Push;
   Push ("end select;" & ASCII.CR);

   Timed_Entry_Mode := True;
   -- True
when An_Asynchronous_Select_Statement =>     -- 9.7.4
   --               Send_Label ("select" & ASCII.CR);
   declare
      Trigger_Statement : Asis.Statement
        := Asis.Statements.Sequence_Of_Statements
          (Asis.Statements.Statement_Paths(Element)(1))(1);
   begin
      case Asis.Elements.Statement_Kind(Trigger_Statement) is
      when A_Delay_Until_Statement |
           A_Delay_Relative_Statement =>
         Send_Label (To_S(Designated_Probes
           (ASYNCHRONOUS_DELAY,
              To_V(Current_Unit_Name),
              False)) & "select" & ASCII.CR);
      when An_Entry_Call_Statement =>
         if Is_Protected (Trigger_Statement) then
            Send_Label (To_S(Designated_Probes
```

```
                  (ASYNCHRONOUS_PROTECTED_ENTRY_CALL,
                     To_V(Current_Unit_Name),
                     False)) & "select" & ASCII.CR);
               else
                  Send_Label (To_S(Designated_Probes
                    (ASYNCHRONOUS_ENTRY_CALL,
                     To_V(Current_Unit_Name), False))
                    & "select" & ASCII.CR);
               end if;
            when others =>
               Put_Line("invalid trigger?");
               raise Program_Error;
            end case;
         end ;
         Push;
         Push ("end select;" & ASCII.CR &
             To_S(Designated_Probes(ASYNCHRONOUS_SELECT_END,
             To_V(Current_Unit_Name),
             False)));
      when An_Abort_Statement =>                -- 9.8
         --              Send_label ("abort ");
         Send_label (Abort_Label);
         Push (";" & ASCII.CR,
               Is_Comma_No_Parenthesis_List,
               Count (Asis.Statements.Aborted_Tasks (Element)));
      when A_Raise_Statement =>                 -- 11.3
         if Is_Here (Asis.Statements.Raised_Exception (Element))
         then
            Send_Label ("raise ");
            Push (";" & ASCII.CR);
         else
            Send_Label ("raise;" & ASCII.CR);
         end if;
      when A_Code_Statement =>                  -- 13.8
         Push (";" & ASCII.CR);
      end case;

   end if;

when A_Path =>
   case Asis.Elements.Path_Kind (Element) is
      when Not_A_Path =>                 -- An unexpected element
         Ada.Text_IO.Put ("<<Node Not_A_Path>>");
      when An_If_Path =>                 -- 5.3:
         Send ("if ");
         Push (ASCII.CR & "then" & ASCII.CR);
         Push ("",
               Not_In_A_List,
               Count (Asis.Statements.Sequence_Of_Statements (Element , True)));
         Indent;
      when An_Elsif_Path =>              -- 5.3:
         Send ("elsif ");
         Push (ASCII.CR & "then" & ASCII.CR);
         Push ("",
               Not_In_A_List,
               Count (Asis.Statements.Sequence_Of_Statements (Element , True)));
         Indent;
      when An_Else_Path =>               -- 5.3, 9.7.1, 9.7.3:
         Send ("else" & ASCII.CR);
         Push ("",
               Not_In_A_List,
               Count (Asis.Statements.Sequence_Of_Statements (Element , True)));
         Indent;
      when A_Case_Path =>                -- 5.4:
         Send ("when ");
         Push ("=>" & ASCII.CR,
               Is_Vertical_Line_List,
               Count (Asis.Statements.Case_Statement_Alternative_Choices (Element)));
         Indent (5);
         Check_If_Return_Separator
           (Asis.Statements.Case_Statement_Alternative_Choices (Element));
         Push ("",
               Not_In_A_List,
               Count (Asis.Statements.Sequence_Of_Statements (Element , True)));
         Indent;
      when A_Select_Path =>              -- 9.7.1:
         if Is_Here (Asis.Statements.Guard (Element))
         then
            Send ("when ");
            Push ("=>" & ASCII.CR);
            --                Push;
            --                Indent;
         end if;
         Push ("",
               Not_In_A_List,
               Count (Asis.Statements.Sequence_Of_Statements (Element, True)));
         Indent;
      when An_Or_Path =>                 -- 9.7.1:
         Send ("or" & ASCII.CR);
         if Is_Here (Asis.Statements.Guard (Element))
         then
            Send ("when ");
            Push ("=>" & ASCII.CR);
         end if;
         Push ("",
               Not_In_A_List,
               Count (Asis.Statements.Sequence_Of_Statements (Element, True)));
         Indent;
      when A_Then_Abort_Path =>          -- 9.7.4
         Send ("then abort" & ASCII.CR);
         Push ("",
               Not_In_A_List,
               Count (Asis.Statements.Sequence_Of_Statements (Element , True)));
         Indent;
         --  --|A2012 start
         --  Expression paths:
      when A_Case_Expression_Path =>
         null;
         --  ??? (RM 2012)
```

```
            --  when expression => expression

         when An_If_Expression_Path =>
            null;
            --  ??? (RM 2012)
            --  if condition then expression

         when An_Elsif_Expression_Path =>
            null;
            --  ??? (RM 2012)
            --  elsif condition then expression

         when An_Else_Expression_Path =>
            null;
            --  ??? (RM 2012)
            --  else expression
            --  --|A2012 end
      end case;
   when A_Clause =>
      case Asis.Elements.Clause_Kind (Element) is
         when Not_A_Clause =>                   -- An unexpected element
            Ada.Text_IO.Put ("<<Node Not_A_Clause>>");
         when A_Use_Package_Clause =>           -- 8.4
            Send ("use ");
            Push (";" & ASCII.CR,
                  Is_Comma_No_Parenthesis_List,
                  Count (Asis.Clauses.Clause_Names (Element)));
         when A_Use_Type_Clause =>              -- 8.4
            Send ("use type ");
            Push (";" & ASCII.CR,
                  Is_Comma_No_Parenthesis_List,
                  Count (Asis.Clauses.Clause_Names (Element)));
         when A_Use_All_Type_Clause =>        -- 8.4, Ada 2012
            null;
         when A_With_Clause =>                  -- 10.1.2
            Send ("with ");
            Push (";" & ASCII.CR,
                  Is_Comma_No_Parenthesis_List,
                  Count (Asis.Clauses.Clause_Names (Element)));
         when A_Representation_Clause =>       -- 13.1     -> Representation_Clause_Kinds
            case Asis.Elements.Representation_Clause_Kind (Element) is
               when Not_A_Representation_Clause =>             -- An unexpected element
                  Ada.Text_IO.Put ("<<Node Not_A_Representation_Clause>>");
               when An_Attribute_Definition_Clause =>          -- 13.3
                  Send ("for ");
                  Push ("use ");
                  Push (";" & ASCII.CR);
               when An_Enumeration_Representation_Clause =>    -- 13.4
                  Send ("for ");
                  Push ("use ");
                  Push (";" & ASCII.CR);
               when A_Record_Representation_Clause =>          -- 13.5.1
                  Send ("for ");
                  Push ("use record" & ASCII.CR);
                  Push ("end record;" & ASCII.CR,
                        Not_In_A_List,
                        Count (Asis.Clauses.Component_Clauses (Element, true)));
                  Indent;
               when An_At_Clause =>                            -- J.7
                  Send ("for ");
                  Push ("use at ");
                  Push (";" & ASCII.CR);
            end case;
         when A_Component_Clause =>            -- 13.5.1
            Push ("at ");
            Push ("range ");
            Push (";" & ASCII.CR);
      end case;
   when An_Exception_Handler =>
      Send ("when ");
      if Is_Here (Asis.Statements.Choice_Parameter_Specification (Element))  then
         Push;
      end if;
      Push ("=>" & ASCII.CR,
            Is_Vertical_Line_List,
            Count (Asis.Statements.Exception_Choices (Element)));
      Push ("",
            Not_In_A_List,
            Count (Asis.Statements.Handler_Statements (Element, True)));
      Indent;
   end case;
   ------< end of the case >------
   -- pours the Tmp_Stack in the Lexical_Stack
   -- and reverses the order of the pushed elements.
   if not Is_Send_Lock then
      Commit;
   end if;
exception
   when others =>
      Put_Line("Exception in pre_source");
      --       Put_Line(To_String(Asis.Text.Element_Image(Element)));
      Put_Line(To_String(Asis.Elements.Debug_Image(Element)));
      raise ;
end Pre_Source;

-- 3 procedures that must be in the package.

procedure Initiate_Source
  (Unit    : in    Asis.Compilation_Unit;
   Name    : in    String;
   Control : in out Asis.Traverse_Control;
   State   : in out Info_Source) is
begin
   --     case Asis.Compilation_Units.Unit_Kind (Unit) is
   --        when A_Procedure_Body_Subunit |
   --             A_Function_Body_Subunit  |
   --             A_Package_Body_Subunit   |
   --             A_Task_Body_Subunit      |
   --             A_Protected_Body_Subunit
```

```
   --          =>
   --         declare
   --            I : Natural := 0;
   --         begin
   --            for Index in Name'First .. Name'Last - 4
   --            loop
   --               if Name (Index) = '.'
   --               then
   --                  I := Index;
   --               end if;
   --            end loop;
   --            if I = 0
   --            then
   --               Ada.Text_IO.Put_Line ("Subunit has no complex name.");
   --               return;
   --            else
   --
   --               Ada.Text_IO.New_Line;
   --               Ada.Text_IO.Put_line ("separate (" &
   --                         Name (Name'First .. I - 1) &
   --                         ")");
   --            end if;
   --         end;
   --      when others =>
   --         null;
   --     end case;
   null;
   --  !!???? REQUIRES REVISING
end Initiate_Source;

procedure Terminate_Source
  (Control : in out Asis.Traverse_Control;
   State   : in out Info_Source) is
begin
   State.Finishing_Traversal := True;
   -- normaly there remain only one node on stack ....
   while not Node_Stack.Is_Empty (State.Lexical_Stack)
   loop
      Pre_Source (Asis.Nil_Element, Control, State);
   end loop;
end Terminate_Source;

procedure Post_Source
  (Element : in     Asis.Element;
   Control : in out Asis.Traverse_Control;
   State   : in out Info_Source) is
begin
   case Asis.Elements.Element_Kind(Element) is
   when An_Expression =>
      if (not Gela_Ids.Is_Nil(Conversion_ID)) and
        Gela_Ids.Is_Equal(Conversion_ID, Gela_Ids.Create_ID(Element)) then
         Is_Send_Lock := False;
      end if;
      if Gela_Ids.Is_Equal(Gela_Ids.Create_ID(Element),
                           Assignment_ID) then
         Assignment_ID := Gela_Ids.Nil_ID;
         Is_Send_Lock := False;
      end if;
      if Gela_Ids.Is_Equal(Gela_Ids.Create_ID(Element),
                           Refered_ID) then
         Refered_ID := Gela_Ids.Nil_ID;
         Is_Send_Lock := False;
      end if;
   when An_Association =>
      case Asis.Elements.Association_Kind (Element) is
      when A_Parameter_Association |              -- 6.4
           A_Pragma_Argument_Association |          -- 2.8
           A_Generic_Association =>                 -- 12.3
         declare
            Pname : V_String := Convert_List.Search_Node
              (Aconv_List,
               Gela_Ids.Create_ID(Element));
         begin
            if Pname /= Null_Str then
               Is_Send_Lock := False;
               Convert_List.Delete_Node(Aconv_List,
                                        Gela_Ids.Create_ID(Element));
            end if;
         end;
      when others =>
         null;
      end case;
   when A_Declaration =>
      case Asis.Elements.Declaration_Kind (Element) is
      when A_Procedure_Body_Declaration | A_Function_Body_Declaration
         | A_Package_Body_Declaration | A_Task_Body_Declaration
         | A_Protected_Body_Declaration | An_Entry_Body_Declaration
         | A_Task_Type_Declaration | A_Protected_Type_Declaration
         | A_Single_Task_Declaration | A_Single_Protected_Declaration
         | A_Procedure_Declaration | A_Function_Declaration
         | A_Package_Declaration | An_Entry_Declaration
         =>
         -- Name_Handler
         Exit_Unit;
         -- Name_Handler
      when others =>
         null;
      end case;
   when A_Statement =>
      case Asis.Elements.Statement_Kind (Element) is
      when A_Block_Statement =>
         -- Name_Handler
         Exit_Unit;
         -- Name_Handler
         --    when A_Selective_Accept_Statement =>
         --       Selective_Accept_Mode := False;
      when others =>
         null;
      end case;
```

```
      when others =>
         null;
      end case;
   exception
      when others =>
         Put_Line("Exception in post_source");
         Put_Line(To_String(Asis.Text.Element_Image(Element)));
         Put_Line(To_String(Asis.Elements.Debug_Image(Element)));
         raise ;
   end Post_Source;

   procedure Make_CDT_Of_Top(Decl : Asis.Declaration) is
   begin
      if (Asis.Elements.Declaration_Kind(Decl) /= A_Package_Declaration and
            Asis.Elements.Declaration_Kind(Decl) /= A_Package_Body_Declaration)
        and then
          Component_List."/="(Spec_Reader.Spec_Data(GET_CDT), null) then

         Call_Analyzer.Find_Calls(Decl);

      end if;
   end Make_CDT_Of_Top;

end Source_Trav;
```

C.1.17 Spec Package

```
with List;
with V_Strings; use V_Strings;
with Asis;
with Designations; use Designations;
with Measure_Types; use Measure_Types;


package Spec is


   package Parameter_List is new List(Parameter_T);


   type Selection_Kind is (ALL_PLACE,
   SPECIFIED_PLACE);

   type Place_Kind is (IN_DECLARATION,
       IN_BODY);

   Place_Kind_Of_T : array(Measurement_T) of Place_Kind
     := (IN_DECLARATION, IN_DECLARATION, IN_BODY, IN_BODY, IN_BODY,
 IN_BODY, IN_BODY, IN_BODY, IN_BODY, IN_BODY, IN_BODY, IN_BODY,
 IN_BODY, IN_BODY, IN_DECLARATION, IN_DECLARATION, IN_BODY,
 IN_BODY, IN_BODY, IN_BODY, IN_BODY, IN_BODY,
 IN_BODY, IN_BODY, IN_BODY, IN_DECLARATION, IN_DECLARATION,
 IN_BODY, IN_BODY, IN_BODY, IN_BODY, IN_BODY, IN_BODY,
 IN_BODY, IN_BODY, IN_BODY, IN_BODY, IN_DECLARATION, IN_DECLARATION,
 IN_BODY, IN_BODY, IN_BODY, IN_BODY, IN_BODY);


   Prot_Conversion : array(Measurement_T) of Boolean
     := (False, False, False, False, False, True, False, True, False,
 False, True, True, False, True, False, False, False, False, False,
 True, False, False, False, False, True, True, False, False,
 False, False, True, False, False, False, False, False, True, True,
 False, False, False, False, False, False, False );


   type Measurement_Status;
   type Measurement_Status_Link is access Measurement_Status'CLASS;

   package Status_List is new List(Measurement_Status_Link);


   type Measurement_Condition;
   type Measurement_Condition_Link is access Measurement_Condition;

   package Condition_List is new List(Measurement_Condition_Link);


   subtype Query_ID is Integer range 0..9999;


   type Measurement_Component is
      record

 Parameters : Parameter_List.Node_Link := null;

 Status : Status_List.Node_Link := null;

 Condition : Condition_List.Node_Link := null;

 QID : Query_ID := 0;

 Variable_Name : V_String := Null_Str;
 Call_Name : V_String := Null_Str;
 Conversion_Type : V_String := Null_Str;

      end record;
   type Measurement_Component_Link is access Measurement_Component;
   package Component_List is new List(Measurement_Component_Link);

--   type Spec_Data_T is array(Measurement_T) of Status_List.Node_Link;
   type Spec_Data_T is array(Measurement_T) of Component_List.Node_Link;


   type Measurement_Status is tagged
      record
```

```
 Selection : Selection_Kind := ALL_PLACE;
-- Specified_Block_Name : V_String := Null_Str;
 Specified_Block_Name : Designation;
      end record;


   type Measurement_Condition is tagged
      record

 Selection : Selection_Kind := ALL_PLACE;
 Specified_Block_Name : Designation;
 Condition_Expression : V_String := Null_Str;
      end record;

   function Special_Condition(Status : Measurement_Status; Element : Asis.Element; Current_Unit_Name : V_string) return Boolean;

--    type Block_Status is new Measurement_Status with
--       record
--       end record;


   type Specified_Target is
      record
 Selection : Selection_Kind := ALL_PLACE;

-- Name : V_String := Null_Str;
 Name : Designation;
      end record;
   type Specified_Target_Link is access Specified_Target;
   package Specified_Target_List is new List(Specified_Target_Link);


   type Entry_Call_Status is new Measurement_Status with
      record
 Entries : Specified_Target_List.Node_Link := null;
      end record;

   function Special_Condition(Status : Entry_Call_Status; Element : Asis.Element; Current_Unit_Name : V_string) return Boolean;

   type Subprogram_Call_Status is new Measurement_Status with
      record
 Subprograms : Specified_Target_List.Node_Link := null;
      end record;

   function Special_Condition(Status : Subprogram_Call_Status; Element : Asis.Element; Current_Unit_Name : V_string) return Boolean;

   Prot_Convert_List : Specified_Target_List.Node_Link := null;


end Spec;

with Asis; use Asis;
with Gela_Ids;
with Asis.Elements;
with Asis.Statements;
with Asis.Expressions;
with Asis.Text;
with Ada.Text_IO; use Ada.Text_IO;

with Name_Handler; use Name_Handler;
with Ada.Characters.Handling; use Ada.Characters.Handling;

with Asis_Utils; use Asis_Utils;
with Designations;
use Designations;

package body Spec is

   -- Measurement_Component

   function Special_Condition(Status : Measurement_Status; Element : Asis.Element; Current_Unit_Name : V_string) return Boolean is
   begin
      return True;
   end Special_Condition;

   function Special_Condition(Status : Entry_Call_Status; Element : Asis.Element; Current_Unit_Name : V_string) return Boolean is
      use Specified_Target_List;
      Iter : Iterator_T;
      Callee : Asis.Declaration;
-- := Asis.Statements.Corresponding_Called_Entity(Element);
      Tmp_Name : V_String := Null_Str;
      Entry_Name : Asis.Expression := Get_Called_Name(Element);
      Tmp_Entry : Specified_Target_Link;
      Result : Boolean := False;
   begin
      case Asis.Elements.Statement_Kind(Element) is
 -- timed entry call
 when A_Timed_Entry_Call_Statement |
   A_Conditional_Entry_Call_Statement =>
    Callee := Asis.Statements.Corresponding_Called_Entity(Asis.Statements.Sequence_Of_Statements(Asis.Statements.Statement_Paths(Element)(1))(1));
 when An_Accept_Statement =>
    Tmp_Name := Current_Unit_Name & ".";
 when others =>
    Callee := Asis.Statements.Corresponding_Called_Entity(Element);
      end case;

      if Tmp_Name = Null_Str then
 declare
    Tmp_Entry_Decl : V_String;
 begin
    Tmp_Entry_Decl := Unit_Name_List.Search_Node(Unit_Names, Gela_Ids.Create_Id(Asis.Elements.Enclosing_Element(Callee)));
    if Tmp_Entry_Decl /= Null_Str then
       Tmp_Name := Tmp_Entry_Decl & ".";
    else
       Tmp_Name := Unit_Name_List.Search_Node(Unit_Names, Gela_Ids.Create_Id(Asis.Elements.Enclosing_Element(Asis.Elements.Enclosing_Element(Callee)))) & ".";
    end if;
-- Put_Line(To_String(Asis.Text.Element_Image(Asis.Elements.Enclosing_Element(Asis.Elements.Enclosing_Element(Callee)))));
 end;
      end if;
```

```
      case Asis.Elements.Expression_Kind(Entry_Name) is
 when An_Identifier =>

    Tmp_Name := Tmp_Name & To_String(Asis.Expressions.Name_Image(Entry_Name));
 when A_Selected_Component =>
    Tmp_Name := Tmp_Name & To_String(Asis.Expressions.Name_Image(Asis.Expressions.Selector(Entry_Name)));
 when An_Indexed_Component =>
    null;

 when others =>
    Put_Line("invalid entry name.");
    raise Program_Error;
      end case;

      Set_List(Iter, Status.Entries);
      while not Is_End(Iter) loop
 Tmp_Entry := Get_Now(Iter);
-- Put_Line("entry match: " & To_S(Tmp_Entry.Name.Name.all) & To_S(Tmp_Name));

 if Tmp_Entry.Selection = ALL_PLACE then
    Result := True;

 elsif Match_Designation(Tmp_Entry.Name, Tmp_Name) then
    Result := True;

 elsif Match_Designation(Tmp_Entry.Name, "-" & Tmp_Name) then
    Result := False;
 end if;
 Set_Next(Iter);
      end loop;

      return Result;
   end Special_Condition;

   function Special_Condition(Status : Subprogram_Call_Status; Element : Asis.Element; Current_Unit_Name : V_string) return Boolean is
      use Specified_Target_List;
      Iter : Iterator_T;
      Callee : Asis.Declaration;
-- := Asis.Statements.Corresponding_Called_Entity(Element);
      Tmp_Name : V_String;
--      Subprogram_Name : Asis.Expression := Asis.Statements.Called_Name(Element);
      Tmp_Target : Specified_Target_Link;
      Result : Boolean := False;
   begin
      -- function なら Corresponding_Called_Function
      case Asis.Elements.Element_Kind(Element) is
 when An_Expression =>
    case Asis.Elements.Expression_Kind(Element) is
       when A_Function_Call =>
  Callee := Asis_Utils.Corresponding_Called_Function(Element);
       when others =>
  Ada.Text_IO.Put_Line("not supported.");
  raise Program_Error;
    end case;

 when A_Statement =>
    -- timed entry call
    case Asis.Elements.Statement_Kind(Element) is
       when A_Timed_Entry_Call_Statement |
 A_Conditional_Entry_Call_Statement =>
  Callee := Asis.Statements.Corresponding_Called_Entity(Asis.Statements.Sequence_Of_Statements(Asis.Statements.Statement_Paths(Element)(1))(1));
       when others =>
  Callee := Asis.Statements.Corresponding_Called_Entity(Element);
    end case;

 when others =>
    Ada.Text_IO.Put_Line("not supported.");
    raise Program_Error;
      end case;

--      Tmp_Name := Unit_Name_List.Search_Node(Unit_Names, Gela_Ids.Create_Id(Callee));
      Tmp_Name := To_V(Declaration_Full_Name(Callee));
      Set_List(Iter, Status.Subprograms);
      while not Is_End(Iter) loop
 Tmp_Target := Get_Now(Iter);
-- Put_Line("subp match: " & To_S(Tmp_Target.Name.Name.all) & To_S(Tmp_Name));

 if Tmp_Target.Selection = ALL_PLACE then
    Result := True;

 elsif Match_Designation(Tmp_Target.Name, Tmp_Name) then
    Result := True;

 elsif Match_Designation(Tmp_Target.Name, "-" & Tmp_Name) then
    Result := False;
 end if;
 Set_Next(Iter);
      end loop;

      return Result;
   end Special_Condition;

end Spec;
```

C.1.18 Spec_Reader Package

```
with Asis;
--with Asis.Elements;
with Asis.Iterator;
with Spec; use Spec;
with Measure_Types; use Measure_Types;
with V_Strings; use V_Strings;

package Spec_Reader is
```

```
   Has_Spec : Boolean := False;

   type Info_Source is new Integer;

   procedure Pre_Source
     (Element : in     Asis.Element;
      Control : in out Asis.Traverse_Control;
      State   : in out Info_Source);

   procedure Post_Source
     (Element : in     Asis.Element;
      Control : in out Asis.Traverse_Control;
      State   : in out Info_Source);

   procedure Traverse_Spec is new Asis.Iterator.Traverse_Element
     (Info_Source, Pre_Source, Post_Source);

   Spec_Data : Spec_Data_T;
   Spec_Name : V_String := Null_Str;

end Spec_Reader;



------------------------------------------------------------------------------
--                                                                          --
--                      DISPLAY_SOURCE COMPONENTS                           --
--                                                                          --
--                         S O U R C E _ T R A V                            --
--                                                                          --
--                                  B o d y                                 --
--                                                                          --
--           Copyright (c) 1995-1998, Free Software Foundation, Inc.        --
--                                                                          --
-- Display_Source is free software; you can redistribute it and/or modify it--
-- under terms of the  GNU General Public License  as published by the Free --
-- Software Foundation;  either version 2,  or  (at your option)  any later --
-- version. Display_Source is distributed in the hope  that it will be use- --
-- ful, but WITHOUT ANY WARRANTY; without even the implied warranty of MER- --
-- CHANTABILITY or  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General  --
-- Public License for more details. You should have received a copy of the  --
-- GNU General Public License distributed with GNAT; see file COPYING. If   --
-- not, write to the Free Software Foundation, 59 Temple Place Suite 330,   --
-- Boston, MA 02111-1307, USA.                                              --
--                                                                          --
-- Display_Source is distributed as a part of the ASIS implementation for   --
-- GNAT (ASIS-for-GNAT).                                                     --
--                                                                          --
-- The   original   version   of  Display_Source  has  been  developed  by  --
-- Jean-Charles  Marteau and Serge Reboul,  ENSIMAG  High School Graduates  --
-- (Computer sciences)  Grenoble,  France  in  Sema Group Grenoble, France. --
--                                                                          --
-- Display_Source is now maintained by Ada Core Technologies Inc            --
-- (http://www.gnat.com).                                                   --
------------------------------------------------------------------------------

---------------------------------------------------------------------
-- This package is part of the ASIS application display_source --
---------------------------------------------------------------------
with Ada;
with Ada.Wide_Text_IO; use Ada.Wide_Text_IO;
with Ada.Text_IO;

with Ada.Characters.Handling; use Ada.Characters.Handling; --  ???
with String_Handler; use String_Handler;
with V_Strings; use V_Strings;
with Designations; use Designations;
with Gnat.Regexp; use Gnat.Regexp;

with Asis;

with Asis.Compilation_Units;
with Asis.Clauses;
with Asis.Declarations;
with Asis.Definitions;
with Asis.Elements;
with Asis.Expressions;
with Asis.Statements;
with Asis.Text;

package body Spec_Reader is

   use Asis;
   --  to make all the literals from Element classification hierarchy
   --  directly visible

   -- 計測対象ブロック指定プラグマ
   Designate_Subject_Pragma : constant Wide_String := "designate_subject";
   -- 計測対象ブロック指定プラグマに対する最大引数
   Max_Designate_Argument : constant Integer := 5;
   -- 現在の計測対象指定
   Current_Designation : array(1..Max_Designate_Argument) of V_String;
   Current_Designation_Count : Integer range 0..Max_Designate_Argument;

   -- 計測対象エントリ指定プラグマ
   Designate_Entry_Pragma : constant Wide_String := "designate_entry";
   -- 計測対象エントリ指定プラグマに対する最大引数
   Max_Designate_Entry : constant Integer := 5;
   -- 現在の計測対象指定
   Current_Designate_Entry : array(1..Max_Designate_Entry) of V_String;
   Current_Designate_Entry_Count : Integer range 0..Max_Designate_Entry;

   -- 計測対象サブプログラム指定プラグマ
   Designate_Subprogram_Pragma : constant Wide_String
     := "designate_subprogram";
   -- 計測対象サブプログラム指定プラグマに対する最大引数
   Max_Designate_Subprogram : constant Integer := 5;
   -- 現在の計測対象指定
```

```
Current_Designate_Subprogram : array(1..Max_Designate_Subprogram) of V_String;
Current_Designate_Subprogram_Count : Integer range 0..Max_Designate_Subprogram;

-- 計測条件指定プラグマ
Designate_Condition_Pragma : constant Wide_String
  := "designate_condition";
-- 計測条件指定プラグマに対する最大引数
Max_Designate_Condition : constant Integer := 10;
-- 現在の計測条件指定
Current_Condition : array(1..Max_Designate_Condition) of V_String;
Current_Condition_Count : Integer range 0..Max_Designate_Condition;
-- 現在の計測条件指定 (リスト化)
Temp_Condition_List : Condition_List.Node_Link := null;


-- 手続き宣言を読み取ったときの動作モード
-- NORMAL : 通常
-- REFER : 変数参照を受け取る手続きとみなす
-- ASSIGN : 変数定義を受け取る手続きとみなす
type Reading_Mode_T is (NORMAL, REFER, ASSIGN);
Reading_Mode : Reading_Mode_T := NORMAL;

-- 変数計測のための手続き登録 (参照)
Variable_Refer_Pragma : constant Wide_String
  := "variable_refer";
-- pragma Variable_Refer("計測対象変数名"
-- [, "計測対象ブロック [, ...]]);
-- Target : Target_Type : 特殊な引数 (計測対象変数の値)
-- 計測対象に対する最大引数
-- 計測対象変数名 フルネーム block.var

Max_Variable_Block : constant Integer := 5;
-- 現在の計測対象変数名
Current_Variable_Name : V_String;
-- 現在の計測対象指定
Current_Variable_Block : array(1..Max_Variable_Block) of V_String;
Current_Variable_Block_Count : Integer range 0..Max_Variable_Block;

-- 変数計測のための手続き登録 (定義)
Variable_Assign_Pragma : constant Wide_String
  := "variable_assign";
-- pragma Variable_Assign("計測対象変数名"
-- [, "計測対象ブロック [, ...]]);
-- Target : Target_Type : 特殊な引数 (計測対象変数の値)
-- 計測対象に対する最大引数
--Max_Variable_Assign : constant Integer := 5;
-- 現在の計測対象変数名 (参照と共通)
-- Current_Variable_Name : V_String := Null_Str;
-- 現在の計測対象指定 (参照と共通)
--Current_Variable_Block : array(1..Max_Variable_Assign) of V_String;
--Current_Variable_Block_Count : Integer range 0..Max_Variable_Assign;

-- probe の場所によって期待される宣言の種類
Expected_Declaration : array(Place_Kind) of Asis.Declaration_Kinds
  := (A_Function_Declaration, A_Procedure_Declaration);

-- 仕様読み取りモード
-- compilation_unit となっているパッケージの仕様部からのみ
-- 計測仕様を読み込む
Spec_Mode : Boolean := False;

-- クエリー名の判定のための正規表現
Query_Regexp : array(Measurement_T) of Designation;

-- クエリー名の判定のための正規表現 Query_Regexp を作成
procedure Make_Query_Regexp is
begin
   for I in Measurement_T loop
      Set_Designation(Query_Regexp(I), "#"
                      & Measurement_T'IMAGE(I)
                      & "[0-9]*");
   end loop;
end Make_Query_Regexp;

--   package Measurement_T_IO is new Enumeration_IO(Measurement_T);
-------------------------------------------
--                                       --
--  Here is the pre procedure to provide  --
--  to Traverse_Element to make a source  --
--  display.                             --
--                                       --
-------------------------------------------

procedure Pre_Source
  (Element : in     Asis.Element;
   Control : in out Asis.Traverse_Control;
   State   : in out Info_source) is

   procedure Set_Conversion(Target : in Specified_Target_List.Node_Link) is
      use Specified_Target_List;
      Iter_Prot, Iter_Target : Iterator_T;
      Tmp_Target : Specified_Target_Link;
   begin
      Set_List(Iter_Prot, Prot_Convert_List);
      if Is_End(Iter_Prot) or else
        Get_Now(Iter_Prot).Selection /= ALL_PLACE then
         Set_List(Iter_Target, Target);
         while not Is_End(Iter_Target) loop
            Tmp_Target := Get_Now(Iter_Target);
            if Tmp_Target.Selection = ALL_PLACE then
               Prot_Convert_List := null;
               Add_Contents(Prot_Convert_List,
                 new Specified_Target'(ALL_PLACE,
                   Null_Designation));
               exit;
            else
               Add_Contents(Prot_Convert_List, Tmp_Target);
            end if;
            Set_Next(Iter_Target);
```

```
         end loop;
      end if;
   end Set_Conversion;


begin

   ------< begining of the case >------
   case Asis.Elements.Element_Kind (Element) is
   when Not_An_Element =>
      null;
   when A_Pragma =>
      case Asis.Elements.Pragma_Kind (Element) is
         when Not_A_Pragma =>
            null;
         when others =>
            --       Put_Line(Asis.Elements.Pragma_Name_Image(Element));
            -- プラグマの種類が、計測場所指定のプラグマならば
            if Is_Equal(Asis.Elements.Pragma_Name_Image(Element),
                        Designate_Subject_Pragma) then
               -- 計測箇所数のリセット
               Current_Designation_Count := 0;
               declare
                  -- プラグマの引数
                  Elist : Asis.Association_List
                    := Asis.Elements.Pragma_Argument_Associations(Element);
               begin
                  -- 各引数について
                  for I in Elist'RANGE loop
                     declare
                        Exp : Asis.Expression
                          := Asis.Expressions.Actual_Parameter(Elist(I));
                     begin
                        if I > Max_Designate_Argument then
                           Put_Line("too many arguments in pragma " &
                                    Designate_Subject_Pragma);
                           raise Program_Error;
                        end if;
                        -- Put_Line(Asis.Elements.Debug_Image(Exp));
                        if Asis.Elements.Expression_Kind(Exp)
                          = A_String_Literal then
                           -- Put_Line(Asis.Expressions.Value_Image(Exp));
                           declare
                              Tmp_Str : Wide_String := Asis.Expressions.Value_Image(Exp);
                           begin
                              if Tmp_Str'LENGTH > 2 then
                                 Current_Designation(I)
                                   := To_V
                                     (Tmp_Str(Tmp_Str'FIRST+1..Tmp_Str'LAST-1)
                                     );
                              else
                                 Current_Designation(I) := Null_Str;
                              end if;
                           end;
                           Current_Designation_Count := I;
                        else
                           Put_Line("invalid parameter(1).");
                           raise Program_Error;
                        end if;
                     end;
                  end loop;
               end;
            end if;

            -- エントリ指定のプラグマ
            if Is_Equal(Asis.Elements.Pragma_Name_Image(Element),
                        Designate_Entry_Pragma) then
               Current_Designate_Entry_Count := 0;
               declare
                  Elist : Asis.Association_List
                    := Asis.Elements.Pragma_Argument_Associations(Element);
               begin
                  for I in Elist'RANGE loop
                     declare
                        Exp : Asis.Expression
                          := Asis.Expressions.Actual_Parameter(Elist(I));
                     begin
                        if I > Max_Designate_Entry then
                           Put_Line("too many arguments in pragma " &
                                    Designate_Entry_Pragma);
                           raise Program_Error;
                        end if;
                        -- Put_Line(Asis.Elements.Debug_Image(Exp));
                        if Asis.Elements.Expression_Kind(Exp)
                          = A_String_Literal then
                           -- Put_Line(Asis.Expressions.Value_Image(Exp));
                           declare
                              Tmp_Str : Wide_String := Asis.Expressions.Value_Image(Exp);
                           begin
                              if Tmp_Str'LENGTH > 2 then
                                 Current_Designate_Entry(I) :=
                                   To_V(Tmp_Str(Tmp_Str'FIRST+1..Tmp_Str'LAST-1));
                              else
                                 Current_Designate_Entry(I) := Null_Str;
                              end if;
                           end;
                           Current_Designate_Entry_Count := I;
                        else
                           Put_Line("invalid parameter(2).");
                           raise Program_Error;
                        end if;
                     end;
                  end loop;
               end;
            end if;

            -- サブプログラム指定のプラグマ
            if Is_Equal(Asis.Elements.Pragma_Name_Image(Element),
```

```
            Designate_Subprogram_Pragma) then
   Current_Designate_Entry_Count := 0;
   declare
      Elist : Asis.Association_List
        := Asis.Elements.Pragma_Argument_Associations(Element);
   begin
      for I in Elist'RANGE loop
         declare
            Exp : Asis.Expression
              := Asis.Expressions.Actual_Parameter(Elist(I));
         begin
            if I > Max_Designate_Subprogram then
               Put_Line("too many arguments in pragma " &
                        Designate_Subprogram_Pragma);
               raise Program_Error;
            end if;
            -- Put_Line(Asis.Elements.Debug_Image(Exp));
            if Asis.Elements.Expression_Kind(Exp)
              = A_String_Literal then
               -- Put_Line(Asis.Expressions.Value_Image(Exp));
               declare
                  Tmp_Str : Wide_String := Asis.Expressions.Value_Image(Exp);
               begin
                  if Tmp_Str'LENGTH > 2 then
                     Current_Designate_Subprogram(I) :=
                       To_V(Tmp_Str(Tmp_Str'FIRST+1..Tmp_Str'LAST-1));
                  else
                     Current_Designate_Subprogram(I) := Null_Str;
                  end if;
               end;
               Current_Designate_Subprogram_Count := I;
            else
               Put_Line("invalid parameter(3).");
               raise Program_Error;
            end if;
         end;
      end loop;
   end;
end if;

-- 変数指定のプラグマ
if Is_Equal(Asis.Elements.Pragma_Name_Image(Element),
            Variable_Refer_Pragma) or
  Is_Equal(Asis.Elements.Pragma_Name_Image(Element),
           Variable_Assign_Pragma) then
   if Is_Equal(Asis.Elements.Pragma_Name_Image(Element),
               Variable_Refer_Pragma) then
      Reading_Mode := REFER;
   else
      Reading_Mode := ASSIGN;
   end if;

   Current_Variable_Block_Count := 0;
   declare
      Elist : Asis.Association_List
        := Asis.Elements.Pragma_Argument_Associations(Element);
      Exp : Asis.Expression;
   begin
      if Elist'LENGTH /= 1 then
         Put_Line("One parameter is required in Variable pragma.");
         raise Program_Error;
      end if;
      --     Current_Variable_Block_Count := 0;
      --     for I in Elist'RANGE loop
      --  if I > Max_Variable_Block + 1 then
      --     Put_Line("too many arguments in pragma variable");
      --     raise Program_Error;
      --  end if;
      Exp := Asis.Expressions.Actual_Parameter(Elist(1));
      if Asis.Elements.Expression_Kind(Exp)
        = A_String_Literal then
         declare
            Tmp_Str : Wide_String := Asis.Expressions.Value_Image(Exp);
         begin
            if Tmp_Str'LENGTH > 2 then
               --       if I = Elist'FIRST then
               Current_Variable_Name :=
                 To_V(Tmp_Str(Tmp_Str'FIRST+1..Tmp_Str'LAST-1));
               --       else
               --       Current_Variable_Block(I-1) :=
               --   To_V(Tmp_Str(Tmp_Str'FIRST+1..Tmp_Str'LAST-1));
               --    Current_Variable_Block_Count := I - 1:
               -- end if;

            else
               Put_Line("invalid parameter(4).");
               raise Program_Error;
            end if;
         end;
      else
         Put_Line("invalid parameter(5).");
         raise Program_Error;
      end if;
      --     end loop;
   end;
end if;

-- 計測条件指定のプラグマならば
if Is_Equal(Asis.Elements.Pragma_Name_Image(Element),
            Designate_Condition_Pragma) then
   -- 計測箇所数のリセット
   Current_Designation_Count := 0;
   declare
      -- プラグマの引数
      Elist : Asis.Association_List
        := Asis.Elements.Pragma_Argument_Associations(Element);
      -- 以下の 2 つは、ある一つの場所指定
      Specified_Block_Name : Designation;
      Selection : Selection_Kind;
```

```
            begin
               Temp_Condition_List := null;
               -- 各引数について
               for I in Elist'RANGE loop
                  declare
                     Exp : Asis.Expression
                       := Asis.Expressions.Actual_Parameter(Elist(I));
                  begin
                     if I > Max_Designate_Condition then
                        Put_Line("too many arguments in pragma " &
                                 Designate_Condition_Pragma);
                        raise Program_Error;
                     end if;
                     if Asis.Elements.Expression_Kind(Exp)
                       = A_String_Literal then
                        declare
                           Tmp_Str : Wide_String := Asis.Expressions.Value_Image(Exp);
                        begin
                           if Tmp_Str'LENGTH > 2 then
                              Current_Condition(I)
                                := To_V
                                  (Tmp_Str(Tmp_Str'FIRST+1..Tmp_Str'LAST-1)
                                  );
                           else
                              Current_Condition(I) := Null_Str;
                           end if;
                        end;
                        Current_Condition_Count := I;


                     else
                        Put_Line("invalid parameter(1).");
                        raise Program_Error;
                     end if;
                  end;
               end loop;

               -- ただちに各ハンドラの Measurement_Component に
               -- 格納できるように、データ構造をリストに変換
               for I in 1..Current_Condition_Count loop
                  -- 奇数番目の引数 : 条件式
                  -- 偶数           : 場所
                  if I mod 2 = 1 then
                     if I = Current_Condition_Count
                       or else Is_Equal(To_S(Current_Condition(I+1)), String'("all")) then
                        Selection := All_Place;
                     else
                        Selection := Specified_Place;
                        Set_Designation(Specified_Block_Name,
                                        To_S(Current_Condition(I+1)));
                     end if;
                     Condition_List.Add_Contents
                       (Temp_Condition_List,
                        new Measurement_Condition'
                          (Selection,
                           Specified_Block_Name,
                           Current_Condition(I)));
                  end if;
               end loop;
            end;
         end if;



   end case;
when A_Defining_Name =>
   case Asis.Elements.Defining_Name_Kind (Element) is
      when Not_A_Defining_Name =>
         null;
      when A_Defining_Identifier |
           A_Defining_Character_Literal |
           A_Defining_Enumeration_Literal =>
         null;
      when A_Defining_Operator_Symbol =>
         null;
      when A_Defining_Expanded_Name =>
         null;
   end case;
when A_Declaration =>
   case Asis.Elements.Declaration_Kind (Element) is
      when Not_A_Declaration =>
         null;
      when An_Ordinary_Type_Declaration =>
         null;
      when A_Task_Type_Declaration =>
         null;
      when A_Protected_Type_Declaration =>
         null;
      when An_Incomplete_Type_Declaration =>
         null;
         --  --|A2005 start
      when A_Tagged_Incomplete_Type_Declaration =>     --  3.10.1(2)
         null;
         --  --|A2005 end
      when A_Private_Type_Declaration =>
         null;
      when A_Private_Extension_Declaration =>
         null;
      when A_Subtype_Declaration =>
         null;
      when A_Variable_Declaration =>
         null;
      when A_Constant_Declaration =>
         null;
      when A_Deferred_Constant_Declaration =>
         null;
      when A_Single_Task_Declaration =>
         null;
      when A_Single_Protected_Declaration =>
```

```
               null;
            when An_Integer_Number_Declaration =>
               null;
            when A_Real_Number_Declaration =>
               null;
            when An_Enumeration_Literal_Specification =>
               null;
            when A_Discriminant_Specification =>
               null;
            when A_Component_Declaration =>
               null;
            when A_Loop_Parameter_Specification =>
               null;
               --  --|A2012 start
            when A_Generalized_Iterator_Specification =>     -- 5.5.2    -> Trait_Kinds
               null;
            when An_Element_Iterator_Specification =>        -- 5.5.2    -> Trait_Kinds
               null;
               --  --|A2012 end
            when A_Procedure_Body_Declaration =>
               null;
            when A_Function_Body_Declaration =>
               null;
--  --|A2005 start
            when A_Return_Variable_Specification =>          -- 6.5
               null;
            when A_Return_Constant_Specification =>          -- 6.5
               null;
            when A_Null_Procedure_Declaration =>             -- 6.7
               null;
--  --|A2005 end
--  --|A2012 start
            when An_Expression_Function_Declaration =>       -- 6.8
               null;
--  --|A2012 end
            when A_Parameter_Specification =>
               null;
            when A_Procedure_Declaration | A_Function_Declaration =>
               declare
                  Subprogram_Name : V_String := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names(Element)(1)));
                  --  Subprogram_Name : Wide_String := Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names(Element)(1));
                  --  Selection : Selection_Kind := ALL_PLACE;
               begin
                  --  Put_Line(Asis.Text.Element_Image(Element));
                  --  if Current_Designation_Count > 0 and then
                  --    (not Is_Equal(To_S(Current_Designation(1)), String'("all"))) then
                  --     Ada.Text_IO.Put_Line(To_S(Current_Designation(1)));
                  --     Selection := Specified_Place;
                  --  end if;

                  -- Reading_Mode に基づいて、通常時と変数関係の時とを
                  -- 処理をわける
                  case Reading_Mode is
                  when NORMAL =>

                     -- 各計測タイプについて
                     for I in Measurement_T loop
                        -- 計測タイプ名とサブプログラム名が同じか、
                        -- もしくはサブプログラム名の末尾に数字がついている
                        -- だけならば
                        if Match_Designation(Query_Regexp(I), Subprogram_Name) then
                           --      if Is_Equal(Request_Name, Subprogram_Name) then
                           -- 関数か手続きが一致しなければならない
                           if Asis.Elements.Declaration_Kind(Element) /=
                             Expected_Declaration(Place_Kind_Of_T(I)) then
                              Put(To_S(Subprogram_Name) & " must be ");
                              Ada.Text_IO.Put_Line(Declaration_Kinds'IMAGE(Expected_Declaration(Place_Kind_Of_T(I))));
                              raise Program_Error;
                           end if;
                           declare
                              Params : Parameter_List.Node_Link := null;
                              States : Status_List.Node_Link := null;
                              P_List : Asis.Parameter_Specification_List :=
                                Asis.Declarations.Parameter_Profile(Element);
                              Qid : Query_ID := 0;
                              Measurement_T_Name : String
                                := String'(Measurement_T'IMAGE(I));
                           begin
                              -- クエリ ID の取得
                              -- クエリ名よりサブプログラム名が長い
                              --   = 数値によって ID が明示されている
                              if Length(Subprogram_Name) >
                                Measurement_T_Name'LENGTH then
                                 declare
                                    Subp_Str : String
                                      := To_S(Subprogram_Name);
                                 begin
                                    Qid := Query_ID'VALUE
                                      (Subp_Str(Subp_Str'FIRST
                                       + Measurement_T_Name'LENGTH
                                       .. Subp_Str'LAST));
                                    if not (Qid in 1..9999) then
                                       Put_Line("Query ID "
                                         & To_Wide_String(Integer'IMAGE
                                           (Integer(Qid)))
                                         & To_Wide_String(" is out of range."));
                                       raise Program_Error;
                                    end if;
                                 end;
                              end if;

                              -- 引数情報の取得
                              for J in P_List'RANGE loop
                                 declare
                                    P_Name : V_String := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names(P_List(J)) (1)));
                                    P_Type_Str : V_String;
                                    P_Type : Asis.Expression := Asis.Declarations.Declaration_Subtype_Mark(P_List(J));
                                    Hit_Flag : Boolean := False;
                                 begin
```

```
      if Asis.Elements.Expression_Kind(P_Type) /= An_Identifier then
         Put_Line("invalid parameter subtype.");
         raise Program_Error;
      end if;
      P_Type_Str := To_V(Asis.Expressions.Name_Image(P_Type));
      for K in Parameter_T loop
         if Equal_Insensitive(To_V(Parameter_T'IMAGE(Parameter_T'(K))), P_Name) and then Equal_Insensitive(Type_Of_Parameter(K), P_Type_Str) then
            Hit_Flag := True;
            Parameter_List.Add_Contents(Params,
                                        K);
            exit;
         end if;
      end loop;

      if not Hit_Flag then
         Put_Line("invalid parameter(4).");
         raise Program_Error;
      end if;
   end;
end loop;

declare
   --       Specified_Block_Name : V_String := Null_Str;
   Specified_Block_Name : Designation;
   Selection : Selection_Kind;
   Target_List : Specified_Target_List.Node_Link := null;
   Target_Selection : Selection_Kind;
   --       Target_Name : V_String := Null_Str;
   Target_Name : Designation;
begin
   -- 各指定固有の処理
   case I is
   when SIMPLE_ENTRY_CALL |
        SIMPLE_ENTRY_CALL_COMPLETION |
        TIMED_ENTRY_CALL |
        ACCEPT_START |
        ACCEPT_START_WITH_TERMINATE |
        CONDITIONAL_ENTRY_CALL |
        RENDEZVOUS_START |
        RENDEZVOUS_END |
        ASYNCHRONOUS_ENTRY_CALL |
        ASYNCHRONOUS_PROTECTED_ENTRY_CALL |
        REQUEUE_START |
        REQUEUE_WITH_ABORT_START |
        PROTECTED_REQUEUE_START |
        PROTECTED_REQUEUE_WITH_ABORT_START =>

      -- (エントリ指定リストの作成)
      if Current_Designate_Entry_Count = 0 then
         Specified_Target_List.Add_Contents
           (Target_List,
            new Specified_Target'(All_Place,
              Null_Designation));
      else
         for J in 1..Current_Designate_Entry_Count loop
            if Is_Equal(To_S(Current_Designate_Entry(J)), String'("all")) then
               Target_Selection := All_Place;
               --       Target_Name := Null_Str;
            else
               Target_Selection := Specified_Place;
               --       Target_Name := Current_Designate_Entry(J);
               Set_Designation
                 (Target_Name,
                  To_S(Current_Designate_Entry(J)));
            end if;
            Specified_Target_List.Add_Contents
              (Target_List,
               new Specified_Target'(Target_Selection,
                 Target_Name));
         end loop;
      end if;
      if Prot_Conversion(I) then
         Set_Conversion(Target_List);
      end if;

   when PROTECTED_PROCEDURE_CALL |
        PROCEDURE_CALL |
        PROTECTED_PROCEDURE_CALL_COMPLETION |
        PROCEDURE_CALL_COMPLETION |
        PROTECTED_ENTRY_CALL |
        PROTECTED_ENTRY_CALL_COMPLETION |
        TIMED_PROTECTED_ENTRY_CALL |
        CONDITIONAL_PROTECTED_ENTRY_CALL |
        FUNCTION_CALL |
        FUNCTION_CALL_COMPLETION |
        PROTECTED_FUNCTION_CALL |
        PROTECTED_FUNCTION_CALL_COMPLETION =>
      -- (手続き指定リストの作成)
      if Current_Designate_Subprogram_Count = 0 then
         Specified_Target_List.Add_Contents
           (Target_List,
            new Specified_Target'(All_Place,
              Null_Designation));
      else
         for J in 1..Current_Designate_Subprogram_Count loop
            if Is_Equal(To_S(Current_Designate_Subprogram(J)), String'("all")) then
               Target_Selection := All_Place;
               --       Target_Name := Null_Str;
            else
               Target_Selection := Specified_Place;
               --       Target_Name := Current_Designate_Subprogram(J);
               Set_Designation
                 (Target_Name,
                  To_S(Current_Designate_Subprogram(J)));
            end if;
            Specified_Target_List.Add_Contents
              (Target_List,
               new Specified_Target'(Target_Selection,
                 Target_Name));
```

```
      end loop;
   end if;
   if Prot_Conversion(I) then
      Set_Conversion(Target_List);
   end if;

when others =>
   null;
end case;
if Current_Designation_Count = 0 then
   case I is
   when BLOCK_ELABORATION_START |
        BLOCK_ELABORATION_COMPLETION |
        BLOCK_EXECUTION_START |
        BLOCK_EXECUTION_COMPLETION |
        TASK_ACTIVATION_START |
        TASK_ACTIVATION_COMPLETION |
        LIBRARY_PACKAGE_ELABORATION_START |
        LIBRARY_PACKAGE_ELABORATION_COMPLETION |
        ABORT_START |
        ASYNCHRONOUS_DELAY |
        ASYNCHRONOUS_TRIGGER_END |
        ASYNCHRONOUS_SELECT_END |
        GET_CDT |
        GET_UNIT_ID |
        LABEL_PASSAGE |
        STATEMENT_EXECUTION_START |
        STATEMENT_EXECUTION_COMPLETION =>
      Status_List.Add_Contents
        (States,
         new Measurement_Status'(All_Place,
           Null_Designation));
   when SIMPLE_ENTRY_CALL |
        SIMPLE_ENTRY_CALL_COMPLETION |
        TIMED_ENTRY_CALL |
        ACCEPT_START |
        ACCEPT_START_WITH_TERMINATE |
        CONDITIONAL_ENTRY_CALL |
        RENDEZVOUS_START |
        RENDEZVOUS_END |
        ASYNCHRONOUS_ENTRY_CALL |
        ASYNCHRONOUS_PROTECTED_ENTRY_CALL |
        REQUEUE_START |
        REQUEUE_WITH_ABORT_START |
        PROTECTED_REQUEUE_START |
        PROTECTED_REQUEUE_WITH_ABORT_START =>
      Status_List.Add_Contents
        (States,
         new Entry_Call_Status'(All_Place,
           Null_Designation,
           Target_list));
   when PROTECTED_PROCEDURE_CALL |
        PROCEDURE_CALL |
        PROTECTED_PROCEDURE_CALL_COMPLETION |
        PROCEDURE_CALL_COMPLETION |
        PROTECTED_ENTRY_CALL |
        PROTECTED_ENTRY_CALL_COMPLETION|
        TIMED_PROTECTED_ENTRY_CALL |
        CONDITIONAL_PROTECTED_ENTRY_CALL |
        FUNCTION_CALL |
        FUNCTION_CALL_COMPLETION |
        PROTECTED_FUNCTION_CALL |
        PROTECTED_FUNCTION_CALL_COMPLETION =>
      Status_List.Add_Contents
        (States,
         new Subprogram_Call_Status'
           (All_Place,
            Null_Designation,
            Target_list));
   when others =>
      null;
      --       when others =>
      --  null;
   end case;

else
   for J in 1..Current_Designation_Count loop
      if Is_Equal(To_S(Current_Designation(J)), String'("all")) then
         Selection := All_Place;
         --  Specified_Block_Name
         --    := Null_Str;
      else
         Selection := Specified_Place;
         --  Specified_Block_Name
         --    := Current_Designation(J);
         Set_Designation(Specified_Block_Name, To_S(Current_Designation(J)));
      end if;
      case I is
      when BLOCK_ELABORATION_START |
           BLOCK_ELABORATION_COMPLETION |
           BLOCK_EXECUTION_START |
           BLOCK_EXECUTION_COMPLETION |
           TASK_ACTIVATION_START |
           TASK_ACTIVATION_COMPLETION |
           LIBRARY_PACKAGE_ELABORATION_START |
           LIBRARY_PACKAGE_ELABORATION_COMPLETION |
           ABORT_START |
           ASYNCHRONOUS_DELAY |
           ASYNCHRONOUS_TRIGGER_END |
           ASYNCHRONOUS_SELECT_END |
           GET_CDT |
           GET_UNIT_ID |
           LABEL_PASSAGE |
           STATEMENT_EXECUTION_START |
           STATEMENT_EXECUTION_COMPLETION
         =>
         Status_List.Add_Contents
           (States,
            new Measurement_Status'(Selection,
```

```
                        Specified_Block_Name));
                  when SIMPLE_ENTRY_CALL |
                       SIMPLE_ENTRY_CALL_COMPLETION |
                       TIMED_ENTRY_CALL |
                       ACCEPT_START |
                       ACCEPT_START_WITH_TERMINATE |
                       CONDITIONAL_ENTRY_CALL |
                       RENDEZVOUS_START |
                       RENDEZVOUS_END |
                       ASYNCHRONOUS_ENTRY_CALL |
                       ASYNCHRONOUS_PROTECTED_ENTRY_CALL |
                       REQUEUE_START |
                       REQUEUE_WITH_ABORT_START |
                       PROTECTED_REQUEUE_START |
                       PROTECTED_REQUEUE_WITH_ABORT_START =>

                     Status_List.Add_Contents
                       (States,
                        new Entry_Call_Status'(Selection,
                          Specified_Block_Name,
                          Target_List));
                  when PROTECTED_PROCEDURE_CALL |
                       PROCEDURE_CALL |
                       PROTECTED_PROCEDURE_CALL_COMPLETION |
                       PROCEDURE_CALL_COMPLETION |
                       PROTECTED_ENTRY_CALL |
                       PROTECTED_ENTRY_CALL_COMPLETION |
                       TIMED_PROTECTED_ENTRY_CALL |
                       CONDITIONAL_PROTECTED_ENTRY_CALL |
                       FUNCTION_CALL |
                       FUNCTION_CALL_COMPLETION |
                       PROTECTED_FUNCTION_CALL |
                       PROTECTED_FUNCTION_CALL_COMPLETION =>
                     Status_List.Add_Contents
                       (States,
                        new Subprogram_Call_Status'(Selection,
                          Specified_Block_Name,
                          Target_List));

                  when others =>
                     null;

                  end case;
               end loop;
            end if;
         end;
         Component_List.Add_Contents
           (Spec_Data(I),
            new Measurement_Component'(Params, States,
              Temp_Condition_List,
              Qid,
              Null_Str,
              Null_Str,
              Null_Str));
      end;
    end if;
  end loop;
when REFER|ASSIGN =>
  -- 変数指定は、必ず手続き
  if Asis.Elements.Declaration_Kind(Element) /=
    A_Procedure_Declaration then
    Put_Line(To_S(Subprogram_Name) & " must be a procedure.");
    raise Program_Error;
  end if;
  declare
    Params : Parameter_List.Node_Link := null;
    States : Status_List.Node_Link := null;
    P_List : Asis.Parameter_Specification_List :=
      Asis.Declarations.Parameter_Profile(Element);
    M_Type : Measurement_T;
    -- 型変換をする型名
    C_Type : V_String := Null_Str;
  begin
    if Reading_Mode = REFER then
      M_Type := REFERED_VARIABLE;
    else
      M_Type := ASSIGNED_VARIABLE;
    end if;

    for J in P_List'RANGE loop
      declare
        P_Name : V_String := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names(P_List(J)) (1)));
        P_Type_Str : V_String;
        P_Type : Asis.Expression := Asis.Declarations.Declaration_Subtype_Mark(P_List(J));
        Hit_Flag : Boolean := False;
      begin
        if Asis.Elements.Expression_Kind(P_Type) /= An_Identifier then
          Put_Line("invalid parameter subtype.");
          raise Program_Error;
        end if;
        P_Type_Str := To_V(Asis.Expressions.Name_Image(P_Type));
        -- Clone の型名を得るための仮の処置
        --C_Type := P_Type_Str;
        for K in Parameter_T loop
          if Equal_Insensitive(To_V(Parameter_T'IMAGE(Parameter_T'(K))), P_Name) and then Equal_Insensitive(Type_Of_Parameter(K), P_Type_Str) then
            Hit_Flag := True;
            Parameter_List.Add_Contents(Params,
                                        K);
            exit;
          elsif Equal_Insensitive
            (To_V(String'("target")), P_Name) then
            Hit_Flag := True;
            Parameter_List.Add_Contents(Params,
                                        TARGET);
            C_Type := P_Type_Str;
            exit;

          end if;
        end loop;
```

```
                  if not Hit_Flag then
                     Put_Line("invalid parameter(4).");
                     raise Program_Error;
                  end if;
               end;
            end loop;
            declare
               --      Specified_Block_Name : V_String := Null_Str;
               Specified_Block_Name : Designation;
               Selection : Selection_Kind;
            begin
               if Current_Designation_Count = 0 then
                  Status_List.Add_Contents
                    (States,
                     new Measurement_Status'(All_Place,
                       Null_Designation
                      ));
               else
                  for J in 1..Current_Designation_Count loop
                     if Is_Equal(To_S(Current_Designation(J)), String'("all")) then
                        Selection := All_Place;
                        --         Specified_Block_Name
                        --   := Null_Str;
                     else
                        Selection := Specified_Place;
                        --         Specified_Block_Name
                        --   := Current_Designation(J);
                        Set_Designation
                          (Specified_Block_Name,
                           To_S(Current_Designation(J)));
                     end if;
                     Status_List.Add_Contents
                       (States,
                        new Measurement_Status'
                          (Selection,
                           Specified_Block_Name));
                  end loop;
               end if;
            end;
            Component_List.Add_Contents
              (Spec_Data(M_Type),
               new Measurement_Component'(Params, States,
                 Temp_Condition_List,
                 0,
                 Current_Variable_Name,
                 Subprogram_Name,
                 C_Type));
         end;
         Reading_Mode := NORMAL;
      when others =>
         null;
      end case;
   end;
   --       Asis.Elements.Declaration_Kind (Element)

when A_Package_Declaration =>
   if Spec_Name = Null_Str then
      Spec_Name := To_V(Asis.Declarations.Defining_Name_Image(Asis.Declarations.Names(Element) (1)));
   end if;
when A_Package_Body_Declaration =>
   null;
when An_Object_Renaming_Declaration =>
   null;
when An_Exception_Renaming_Declaration =>
   null;
when A_Package_Renaming_Declaration =>
   null;
when A_Procedure_Renaming_Declaration =>
   null;
when A_Function_Renaming_Declaration =>
   null;
when A_Generic_Package_Renaming_Declaration =>
   null;
when A_Generic_Procedure_Renaming_Declaration =>
   null;
when A_Generic_Function_Renaming_Declaration =>
   null;
when A_Task_Body_Declaration =>
   null;
when A_Protected_Body_Declaration =>
   null;
when An_Entry_Declaration =>
   null;
when An_Entry_Body_Declaration =>
   null;
when An_Entry_Index_Specification =>
   null;
when A_Procedure_Body_Stub =>
   null;
when A_Function_Body_Stub =>
   null;
when A_Package_Body_Stub =>
   null;
when A_Task_Body_Stub =>
   null;
when A_Protected_Body_Stub =>
   null;
when An_Exception_Declaration =>
   null;
when A_Choice_Parameter_Specification =>
   null;
when A_Generic_Procedure_Declaration =>
   null;
when A_Generic_Function_Declaration =>
   null;
when A_Generic_Package_Declaration =>
   null;
when A_Package_Instantiation =>
```

```
               null;
            when A_Procedure_Instantiation =>
               null;
            when A_Function_Instantiation =>
               null;
            when A_Formal_Object_Declaration =>
               null;
            when A_Formal_Type_Declaration =>
               null;
--  --|A2012 start
            when A_Formal_Incomplete_Type_Declaration =>
               null;
--  --|A2012 end
            when A_Formal_Procedure_Declaration =>
               null;
            when A_Formal_Function_Declaration =>
               null;
            when A_Formal_Package_Declaration =>
               null;
            when A_Formal_Package_Declaration_With_Box =>
               null;
         end case;
      when A_Definition =>
         case Asis.Elements.Definition_Kind (Element) is
            when Not_A_Definition =>
               null;
            when A_Type_Definition =>
               case Asis.Elements.Type_Kind (Element) is
                  when Not_A_Type_Definition =>
                     null;
                  when A_Derived_Type_Definition =>
                     null;
                  when A_Derived_Record_Extension_Definition =>
                     null;
                  when An_Enumeration_Type_Definition =>
                     null;
                  when A_Signed_Integer_Type_Definition =>
                     null;
                  when A_Modular_Type_Definition =>
                     null;
                  when A_Root_Type_Definition =>
                     null;
                  when A_Floating_Point_Definition =>
                     null;
                  when An_Ordinary_Fixed_Point_Definition =>
                     null;
                  when A_Decimal_Fixed_Point_Definition =>
                     null;
                  when An_Unconstrained_Array_Definition =>
                     null;
                  when A_Constrained_Array_Definition =>
                     null;
                  when A_Record_Type_Definition =>
                     null;
                  when A_Tagged_Record_Type_Definition =>
                     null;
                     --  --|A2005 start
                  when An_Interface_Type_Definition =>          -- 3.9.4      -> Interface_Kinds
                     null;
                     --  --|A2005 end
                  when An_Access_Type_Definition =>
                     case Asis.Elements.Access_Type_Kind (Element) is
                        when Not_An_Access_Type_Definition =>
                           null;
                        when A_Pool_Specific_Access_To_Variable =>
                           null;
                        when An_Access_To_Constant =>
                           null;
                        when An_Access_To_Variable =>
                           null;
                        when An_Access_To_Procedure =>
                           null;
                        when An_Access_To_Protected_Procedure =>
                           null;
                        when An_Access_To_Function =>
                           null;
                        when An_Access_To_Protected_Function =>
                           null;
                     end case;
               end case;
            when A_Subtype_Indication =>
               null;
            when A_Constraint =>
               case Asis.Elements.Constraint_Kind (Element) is
                  when Not_A_Constraint =>
                     null;
                  when A_Range_Attribute_Reference =>
                     null;
                  when A_Simple_Expression_Range =>
                     null;
                  when A_Digits_Constraint =>
                     null;
                  when A_Delta_Constraint =>
                     null;
                  when An_Index_Constraint =>
                     null;
                  when A_Discriminant_Constraint =>
                     null;
               end case;
            when A_Component_Definition =>
               null;
            when A_Discrete_Subtype_Definition |
                 A_Discrete_Range =>
               case Asis.Elements.Discrete_Range_Kind (Element) is
                  when Not_A_Discrete_Range =>
                     null;
                  when A_Discrete_Subtype_Indication =>
                     null;
                  when A_Discrete_Range_Attribute_Reference =>
```

```
               null;
            when A_Discrete_Simple_Expression_Range =>
               null;
         end case;
      when An_Unknown_Discriminant_Part =>
         null;
      when A_Known_Discriminant_Part =>
         null;
      when A_Record_Definition =>
         null;
      when A_Null_Record_Definition =>
         null;
      when A_Null_Component =>
         null;
      when A_Variant_Part =>
         null;
      when A_Variant =>
         null;
      when An_Others_Choice =>
         null;
         --  --|A2005 start
      when An_Access_Definition =>             -- 3.10(6/2)   -> Access_Definition_Kinds
         null;
         --  --|A2005 end
      when A_Private_Type_Definition =>
         null;
      when A_Tagged_Private_Type_Definition =>
         null;
      when A_Private_Extension_Definition =>
         null;
      when A_Task_Definition |
           A_Protected_Definition =>
         null;
      when A_Formal_Type_Definition =>
         case Asis.Elements.Formal_Type_Kind (Element) is
            when Not_A_Formal_Type_Definition =>
               null;
            when A_Formal_Private_Type_Definition =>
               null;
            when A_Formal_Tagged_Private_Type_Definition =>
               null;
            when A_Formal_Discrete_Type_Definition =>
               null;
            when A_Formal_Signed_Integer_Type_Definition =>
               null;
            when A_Formal_Modular_Type_Definition =>
               null;
            when A_Formal_Floating_Point_Definition =>
               null;
            when A_Formal_Ordinary_Fixed_Point_Definition =>
               null;
            when A_Formal_Decimal_Fixed_Point_Definition =>
               null;
               --  --|A2005 start
            when A_Formal_Interface_Type_Definition =>       -- 12.5.5(2) -> Interface_Kinds
               null;
               --  --|D2005 start
               --  Do we really need this value in Formal_Type_Kinds? There is no
               --  difference between it and An_Interface_Type_Definition. The only
               --  reason to have it is not to break the ASIS 95 idea tp have separate
               --  values representing definitions of formal kinds
               --  --|D2005 end

               --  --|A2005 end

            when A_Formal_Derived_Type_Definition =>
               null;
            when A_Formal_Unconstrained_Array_Definition =>
               null;
            when A_Formal_Constrained_Array_Definition =>
               null;
            when A_Formal_Access_Type_Definition =>
               case Asis.Elements.Access_Type_Kind (Element) is
                  when Not_An_Access_Type_Definition =>
                     null;
                  when A_Pool_Specific_Access_To_Variable =>
                     null;
                  when An_Access_To_Constant =>
                     null;
                  when An_Access_To_Variable =>
                     null;
                  when An_Access_To_Procedure =>
                     null;
                  when An_Access_To_Protected_Procedure =>
                     null;
                  when An_Access_To_Function =>
                     null;
                  when An_Access_To_Protected_Function =>
                     null;
               end case;
         end case;
         --  --|A2012 start
      when An_Aspect_Specification =>          -- 13.3.1
         null;
         --  --|A2012 end
   end case;
when An_Expression =>
   case Asis.Elements.Expression_Kind (Element) is
      when Not_An_Expression =>
         null;
      when An_Integer_Literal |
           A_Real_Literal |
           A_String_Literal =>
         null;
      when An_Identifier |
           A_Character_Literal |
           An_Enumeration_Literal =>
         null;
      when An_Operator_Symbol =>
```

```
            null;
         when An_Explicit_Dereference =>
            null;
         when A_Function_Call =>
            null;
         when An_Indexed_Component =>
            null;
         when A_Slice =>
            null;
         when A_Selected_Component =>
            null;
         when An_Attribute_Reference =>
            case Asis.Elements.Attribute_Kind (Element) is
               when Not_An_Attribute =>
                  null;
               when A_First_Attribute |
                    A_Last_Attribute |
                    A_Length_Attribute |
                    A_Range_Attribute |
                    An_Implementation_Defined_Attribute |
                    An_Unknown_Attribute =>
                  null;
               when others =>
                  null;
            end case;
         when A_Record_Aggregate =>
            null;
         when An_Extension_Aggregate =>
            null;
         when A_Positional_Array_Aggregate |
              A_Named_Array_Aggregate =>
            null;
         when An_And_Then_Short_Circuit =>
            null;
         when An_Or_Else_Short_Circuit =>
            null;
         when An_In_Membership_Test =>       --|Ada 2012
            null;
         when A_Not_In_Membership_Test =>        --|Ada 2012
            null;
         when A_Null_Literal =>
            null;
         when A_Parenthesized_Expression =>
            null;
         when A_Type_Conversion =>
            null;
         when A_Qualified_Expression =>
            null;
         when An_Allocation_From_Subtype =>
            null;
         when An_Allocation_From_Qualified_Expression =>
            null;
         when A_Case_Expression =>                       -- Ada 2012
            null;
         when An_If_Expression =>                        -- Ada 2012
            null;
         when A_For_All_Quantified_Expression =>          -- Ada 2012
            null;
         when A_For_Some_Quantified_Expression =>        -- Ada 2012
            null;
      end case;
   when An_Association =>
      case Asis.Elements.Association_Kind (Element) is
         when Not_An_Association =>
            null;
         when A_Discriminant_Association =>
            null;
         when A_Record_Component_Association =>
            null;
         when An_Array_Component_Association =>
            null;
         when A_Parameter_Association |
              A_Pragma_Argument_Association |
              A_Generic_Association =>
            null;
      end case;
   when A_Statement =>
      case Asis.Elements.Statement_Kind (Element) is
         when Not_A_Statement =>
            null;
         when A_Null_Statement =>
            null;
         when An_Assignment_Statement =>
            null;
         when An_If_Statement =>
            null;
         when A_Case_Statement =>
            null;
         when A_Loop_Statement =>
            null;
         when A_While_Loop_Statement =>
            null;
         when A_For_Loop_Statement =>
            null;
         when A_Block_Statement =>
            null;
         when An_Exit_Statement =>
            null;
         when A_Goto_Statement =>
            null;
         when A_Procedure_Call_Statement |
              An_Entry_Call_Statement =>
            null;
         when A_Return_Statement =>
            null;
            --  --|A2005 start
         when An_Extended_Return_Statement =>        -- 6.5
            null;
            --  --|A2005 end
```

```
         when An_Accept_Statement =>
            null;
         when A_Requeue_Statement =>
            null;
         when A_Requeue_Statement_With_Abort =>
            null;
         when A_Delay_Until_Statement =>
            null;
         when A_Delay_Relative_Statement =>
            null;
         when A_Terminate_Alternative_Statement =>
            null;
         when A_Selective_Accept_Statement |
              A_Conditional_Entry_Call_Statement =>
            null;
         when A_Timed_Entry_Call_Statement |
              An_Asynchronous_Select_Statement =>
            null;
         when An_Abort_Statement =>
            null;
         when A_Raise_Statement =>
            null;
         when A_Code_Statement =>
            null;
      end case;
   when A_Path =>
      case Asis.Elements.Path_Kind (Element) is
         when Not_A_Path =>
            null;
         when An_If_Path =>
            null;
         when An_Elsif_Path =>
            null;
         when An_Else_Path =>
            null;
         when A_Case_Path =>
            null;
         when A_Select_Path =>
            null;
         when An_Or_Path =>
            null;
         when A_Then_Abort_Path =>
            null;
            --  --|A2012 start
            --  Expression paths:
         when A_Case_Expression_Path =>
            null;
            --  ??? (RM 2012)
            --  when expression => expression

         when An_If_Expression_Path =>
            null;
            --  ??? (RM 2012)
            --  if condition then expression

         when An_Elsif_Expression_Path =>
            null;
            --  ??? (RM 2012)
            --  elsif condition then expression

         when An_Else_Expression_Path =>
            null;
            --  ??? (RM 2012)
            --  else expression
            --  --|A2012 end
      end case;
   when A_Clause =>
      case Asis.Elements.Clause_Kind (Element) is
         when Not_A_Clause =>
            null;
         when A_Use_Package_Clause =>
            null;
         when A_Use_Type_Clause =>
            null;
         when A_Use_All_Type_Clause =>        -- 8.4, Ada 2012
            null;
         when A_With_Clause =>
            null;
         when A_Representation_Clause =>
            case Asis.Elements.Representation_Clause_Kind (Element) is
               when Not_A_Representation_Clause =>
                  null;
               when An_Attribute_Definition_Clause =>
                  null;
               when An_Enumeration_Representation_Clause =>
                  null;
               when A_Record_Representation_Clause =>
                  null;
               when An_At_Clause =>
                  null;
            end case;
         when A_Component_Clause =>
            null;
      end case;
   when An_Exception_Handler =>
      null;
   end case;
   ------< end of the case >------
   -- pours the Tmp_Stack in the Lexical_Stack
   -- and reverses the order of the pushed elements.
   --   Commit;
end Pre_Source;

procedure Post_Source
  (Element : in     Asis.Element;
   Control : in out Asis.Traverse_Control;
   State   : in out Info_source) is
begin
   null;
```

```
   end Post_Source;

begin
   Make_Query_Regexp;
end Spec_Reader;
```

C.1.19 Stacks Package

```
------------------------------------------------------------------------------
--                                                                          --
--                     DISPLAY_SOURCE COMPONENTS                            --
--                                                                          --
--                             S T A C K S                                  --
--                                                                          --
--                                S p e c                                   --
--                                                                          --
--           Copyright (c) 1995-1998, Free Software Foundation, Inc.        --
--                                                                          --
-- Display_Source is free software; you can redistribute it and/or modify it--
-- under terms of the  GNU General Public License  as published by the Free --
-- Software Foundation;  either version 2,  or  (at your option)  any later --
-- version. Display_Source is distributed in the hope  that it will be use- --
-- ful, but WITHOUT ANY WARRANTY; without even the implied warranty of MER- --
-- CHANTABILITY or  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General  --
-- Public License for more details. You should have received a copy of the  --
-- GNU General Public License distributed with GNAT; see file COPYING. If   --
-- not, write to the Free Software Foundation, 59 Temple Place Suite 330,   --
-- Boston, MA 02111-1307, USA.                                              --
--                                                                          --
-- Display_Source is distributed as a part of the ASIS implementation for   --
-- GNAT (ASIS-for-GNAT).                                                     --
--                                                                          --
-- The   original   version   of  Display_Source  has  been  developed  by  --
-- Jean-Charles  Marteau and Serge Reboul,  ENSIMAG  High School Graduates  --
-- (Computer sciences)  Grenoble,  France  in  Sema Group Grenoble, France. --
--                                                                          --
-- Display_Source is now maintained by Ada Core Technologies Inc            --
-- (http://www.gnat.com).                                                   --
------------------------------------------------------------------------------

-------------------------------------------------------------
-- This package is part of the ASIS application display_source --
-------------------------------------------------------------
-- It provides an implementation of stacks ...                --
-- It is used in Source_Trav package .                        --
-------------------------------------------------------------
generic

   type T_Elem is private ;

   type A_Elem is access all T_Elem ;

   Initial_Number : Natural := 50 ;

package Stacks is

   type Stack is private ;
   Empty_Stack : constant Stack ;

   Stack_Error : exception ;

   -- standard functions for a stack :
   --  Push puts an Elem on the top of the stack
   --  Pop gets the last pushed element of the stack
   procedure Push ( St : in out Stack ; Elem : in T_Elem ) ;
   procedure Pop ( St : in out Stack ; Elem : out T_Elem ) ;
   -- Upper lets you have access to the first Elem
   -- on the top of the stack.
   function Upper ( St : in Stack ) return A_Elem ;
   function Size_Of_Stack ( St : in Stack ) return Natural ;
   function Is_Empty ( St : in Stack ) return Boolean ;

private

   type T_Node ;

   type Stack is access all T_Node ;

   Empty_Stack : constant Stack := null ;

end Stacks ;



------------------------------------------------------------------------------
--                                                                          --
--                     DISPLAY_SOURCE COMPONENTS                            --
--                                                                          --
--                             S T A C K S                                  --
--                                                                          --
--                                B o d y                                   --
--                                                                          --
--           Copyright (c) 1995-1998, Free Software Foundation, Inc.        --
--                                                                          --
-- Display_Source is free software; you can redistribute it and/or modify it--
-- under terms of the  GNU General Public License  as published by the Free --
-- Software Foundation;  either version 2,  or  (at your option)  any later --
-- version. Display_Source is distributed in the hope  that it will be use- --
-- ful, but WITHOUT ANY WARRANTY; without even the implied warranty of MER- --
-- CHANTABILITY or  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General  --
-- Public License for more details. You should have received a copy of the  --
-- GNU General Public License distributed with GNAT; see file COPYING. If   --
-- not, write to the Free Software Foundation, 59 Temple Place Suite 330,   --
-- Boston, MA 02111-1307, USA.                                              --
--                                                                          --
-- Display_Source is distributed as a part of the ASIS implementation for   --
```

```
-- GNAT (ASIS-for-GNAT).                                                    --
--                                                                          --
-- The   original   version   of  Display_Source  has  been  developed  by  --
-- Jean-Charles  Marteau and Serge Reboul,  ENSIMAG  High School Graduates  --
-- (Computer sciences)  Grenoble,  France  in  Sema Group Grenoble, France. --
--                                                                          --
-- Display_Source is now maintained by Ada Core Technologies Inc            --
-- (http://www.gnat.com).                                                   --
------------------------------------------------------------------------------

-------------------------------------------------------------
-- This package is part of the ASIS application display_source --
-------------------------------------------------------------
package body Stacks is

   type T_Node is
      record
 Elem : aliased T_Elem ;
 Next : Stack ;
      end record ;

   -- I don't like allocating new records to get rid
   -- of them in the next minute, so there is a record
   -- provider here that gets the old ones and give
   -- them back when needed. That means unemployment for
   -- garbage collectors ...  :)
   Stock : array (1 .. Initial_Number) of aliased T_Node ;
   -- He he this is a stock of stack ...  :)

   Is_Provider_Initialized : Boolean := False ;

   Provider : Stack := Stock (1)'access ;

   procedure Init_Provider is
   begin
      for I in 2 .. Initial_Number loop
 Stock ( I - 1 ).Next := Stock (I)'Access ;
      end loop ;
      Stock ( Initial_Number ).Next := Empty_Stack ;
   end ;

   -- standard functions for a stack :
   --  Push puts an Elem on the top of the stack
   --  Pop gets the last pushed element of the stack
   procedure Push ( St : in out Stack ; Elem : in T_Elem ) is
      Neu : Stack := Provider ;
      -- Well as new is a reserved word, i use neu which is
      -- new in valencian, a language spoken in a spanish
      -- city on the east coast (east coast of spain ... )
      -- Isn't that exotic ?
   begin
      if Neu = Empty_Stack
      then
 -- Arg ! We got to do it !!!
 Neu := new T_Node ;
      else
 Provider := Provider.Next ;
      end if ;
      Neu.Elem := Elem ;
      Neu.Next := St ;
      St := Neu ;
   end Push ;

   procedure Pop ( St : in out Stack ; Elem : out T_Elem ) is
      Old : Stack := St ;
   begin
      if St = Empty_Stack
      then
 raise Stack_Error ;
      end if ;
      Elem := St.Elem ;
      St := St.Next ;
      Old.Next := Provider ;
      Provider := Old ;
   end Pop ;

   -- Upper lets you have access to the first Elem
   -- on the top of the stack.
   function Upper ( St : in Stack ) return A_Elem is
   begin
      if St = Empty_Stack
      then
 return null ;
      else
 return St.Elem'Access ;
      end if ;
   end Upper ;

   function Size_Of_Stack ( St : in Stack ) return Natural is
      Result : Natural := 0 ;
      Current : Stack := St ;
   begin
      while Current /= Empty_Stack loop
 Result := Result + 1 ;
 Current := Current.Next ;
      end loop ;
      return Result ;
   end Size_Of_Stack ;

   function Is_Empty ( St : in Stack ) return Boolean is
   begin
      return St = Empty_Stack ;
   end Is_Empty ;

begin
   Init_Provider ;
end Stacks ;
```

C.1.20 String_Handler Package

```
package String_Handler is

   -- string comparation(case insensitive)
   function Is_Equal(Left, Right : Wide_String) return Boolean;
   function Is_Equal(Left, Right : String) return Boolean;

   function Eliminate_Space (Source : String) return String;
   function Eliminate_Top_Space (Source : String) return String;

end String_Handler;



with Ada.Characters.Handling;
use Ada.Characters.Handling;
with Ada.Text_IO;
use Ada.Text_IO;

package body String_Handler is

   function Is_Equal (Left, Right : Wide_String) return Boolean is
   begin
      return (To_Lower(To_String(Left)) = To_Lower(To_String(Right)));
   end Is_Equal;

   function Is_Equal (Left, Right : String) return Boolean is
   begin
      return (To_Lower(Left) = To_Lower(Right));
   end Is_Equal;

   function Eliminate_Space (Source : String) return String is
      Dest : String(Source'FIRST..Source'LAST);
      Counter : Integer := Source'FIRST;
   begin
      for I in Source'RANGE loop
 if Source(I) /= ' ' and Source(I) /= Character'VAL(9) then
    Dest(Counter) := Source(I);
    Counter := Counter + 1;
 end if;
      end loop;
      if Counter <= Source'FIRST then
 Put_Line("is_equal : No character except spaces are included.");
 raise Program_Error;
      end if;
      return Dest(Source'FIRST..Counter-1);
   end Eliminate_Space;

   function Eliminate_Top_Space (Source : String) return String is
      Dest : String(Source'FIRST..Source'LAST);
      Counter : Integer := Source'FIRST;
      Is_Top : Boolean := True;
   begin
      for I in Source'RANGE loop
 if (Source(I) /= ' ' and Source(I) /= Character'VAL(9))
   or (not Is_Top) then
    Dest(Counter) := Source(I);
    Is_Top := False;
    Counter := Counter + 1;
 end if;
      end loop;
      if Counter <= Source'FIRST then
 Put_Line("is_equal : No character except spaces are included.");
 raise Program_Error;
      end if;
      return Dest(Source'FIRST..Counter-1);
   end Eliminate_Top_Space;

end String_Handler;
```

C.1.21 Task_Indexed_List Package

```
with Ada.Task_Identification;
use Ada.Task_Identification;

generic
type ITEM is private;
Null_ITEM : ITEM;

package Task_Indexed_List is
   type NODE;
   type A_NODE is access NODE;

   type NODE is
      record
 Index : Task_ID := Null_Task_ID;
 Object : ITEM;
 Next : A_NODE := null;
      end record;

   Null_Node : NODE := (Null_Task_ID, Null_ITEM, null);

   function SEARCH_NODE(TOP : A_NODE; CHILD : Task_ID) return ITEM;
   procedure SET_NODE(TOP : in out A_NODE ; CHILD : in Task_ID ;
    PARENT : in ITEM);
   function GET_NEXT_NODE(TOP : NODE) return NODE;

   procedure DELETE_NODE(TOP : in out A_NODE; CHILD : in Task_ID);
   function IsNULL(TOP : A_NODE) return Boolean;
end Task_Indexed_List;



with Ada.Task_Identification;
use Ada.Task_Identification;
with Ada.Text_IO;
```

```
use Ada.Text_IO;

package body Task_Indexed_List is

--   procedure FREE_NODE is new UNCHECKED_DEALLOCATION(NODE, A_NODE);

   function SEARCH_NODE(TOP : A_NODE; CHILD : Task_ID) return ITEM is
       function SEARCH_ITER(CURRENT_NODE : A_NODE) return ITEM is
       begin
--  New_Line;
--  New_Line;
--  Put_Line(Image(CHILD));
--  New_Line;
  if CURRENT_NODE = null then
--     Put_Line("null!");
     return Null_ITEM;
  end if;
  if CURRENT_NODE.Index = CHILD then
--     Put_Line("GET!");
--     New_Line;
     return CURRENT_NODE.Object;
  end if;
  return SEARCH_ITER(CURRENT_NODE.Next);
       end SEARCH_ITER;

    begin
       return SEARCH_ITER(TOP);
    end SEARCH_NODE;


   function GET_NEXT_NODE(TOP : NODE) return NODE is
   begin
      if TOP.Next = null then
 return Null_NODE;
      end if;
      return TOP.Next.all;
   end GET_NEXT_NODE;

   procedure SET_NODE(TOP : in out A_NODE ; CHILD : in Task_ID ; PARENT : in ITEM) is
       procedure SET_ITER(CURRENT_NODE : in out A_NODE) is
       begin
  if CURRENT_NODE = null then
     CURRENT_NODE := new NODE;
     CURRENT_NODE.Index := CHILD;
     CURRENT_NODE.Object := PARENT;
     CURRENT_NODE.Next := null;
  elsif CURRENT_NODE.Index = CHILD then
     CURRENT_NODE.Object := PARENT;
  else
     SET_ITER(CURRENT_NODE.Next);
  end if;
       end SET_ITER;
    begin
       SET_ITER(TOP);
    end SET_NODE;

   procedure DELETE_NODE(TOP : in out A_NODE ; CHILD : in Task_ID) is
      procedure DELETE_ITER(CURRENT_NODE : in out A_NODE) is
      begin
 if CURRENT_NODE.Index = CHILD then
--    FREE_NODE(CURRENT_NODE);
    CURRENT_NODE := CURRENT_NODE.Next;
 elsif CURRENT_NODE /= null then
    DELETE_ITER(CURRENT_NODE.Next);
 end if;
      end DELETE_ITER;
   begin
      DELETE_ITER(TOP);
   end DELETE_NODE;

   function IsNULL(TOP : A_NODE) return Boolean is
   begin
     return (TOP = null);
   end IsNull;

end Task_Indexed_List;
```

C.1.22 V_Strings Package

```
-- $Id: v_strings.ads,v 1.2 2000/04/07 05:21:20 yusuke Exp $
--
-- Variable-length strings handler package specification.
--
--  by Yoshiaki Kasahara, 1992, 1993.

-- Modified by Yusuke Nonaka, 1999, 2000.

package V_Strings is

type v_string is private;

null_str: constant v_string;

function to_v(s: in string) return v_string;
function to_s(s: in v_string) return string;
function to_v(s: in Wide_string) return v_string;
function to_s(s: in v_string) return Wide_string;

function equal(s1, s2: v_string) return boolean;
function Equal_Insensitive(s1, s2: v_string) return Boolean;
function "&"(s1: v_string;  s2: v_string ) return v_string;
function "&"(s1: v_string;  s2: string   ) return v_string;
function "&"(s1: string;    s2: v_string ) return v_string;
function "&"(s1: v_string;  s2: character) return v_string;
function "&"(s1: character; s2: v_string ) return v_string;
```

```
function Match_Wildcard(S1, S2 : V_String) return Boolean;
function Length(S1 : V_String) return Natural;

private

MAX_LENGTH: constant natural := 1024;

type v_string is
record
str: string(1 .. MAX_LENGTH)
 := (1 .. MAX_LENGTH => ascii.nul);
len: natural range 0 .. MAX_LENGTH:= 0;
end record;

null_str: constant v_string:=( (1 .. MAX_LENGTH => ascii.nul), 0);

end V_Strings;



-- $Id: v_strings.adb,v 1.2 2000/04/07 05:21:29 yusuke Exp $
--
-- Variable-length strings handler package body.
--
--  by Yoshiaki Kasahara, 1992, 1993.
--  modified by Yusuke Nonaka, 1998, 1999.
with Ada.Characters.Handling;
use Ada.Characters.Handling;
package body V_Strings is

function to_v(s: in string) return v_string is
buffer: v_string;
begin
buffer.len := s'Last - s'First + 1;
buffer.str(1..buffer.len) := s;
return buffer;
end to_v;

function to_v(s: in Wide_string) return v_string is
buffer: v_string;
begin
buffer.len := s'Last - s'First + 1;
buffer.str(1..buffer.len) := To_String(S);
return buffer;
end to_v;

function to_s(s: in v_string) return string is
begin
return s.str(1..s.len);
end to_s;

function to_s(s: in v_string) return Wide_string is
begin
return To_Wide_String(s.str(1..s.len));
end to_s;

function equal(s1, s2: v_string) return boolean is
begin
return ((s1.str = s2.str) and (s1.len = s2.len));
end equal;

function Equal_Insensitive(s1, s2: v_string) return boolean is
begin
return ((To_Lower(s1.Str) = To_Lower(s2.Str)) and (s1.len = s2.len));
end Equal_Insensitive;

function "&"(s1: v_string; s2: v_string) return v_string is
buffer: v_string;
begin
buffer.len:= s1.len + s2.len;
buffer.str(1..buffer.len)
:= s1.str(1..s1.len)&s2.str(1..s2.len);
return buffer;
end "&";

function "&"(s1: v_string; s2: string) return v_string is
buffer: v_string;
begin
buffer.len:= s1.len + s2'length;
buffer.str(1..buffer.len):= s1.str(1..s1.len)&s2;
return buffer;
end "&";

function "&"(s1: string; s2: v_string) return v_string is
buffer: v_string;
begin
buffer.len:= s1'length + s2.len;
buffer.str(1..buffer.len):= s1&s2.str(1..s2.len);
return buffer;
end "&";

function "&"(s1: v_string; s2: character) return v_string is
buffer: v_string;
begin
buffer.len:= s1.len + 1;
buffer.str(1..buffer.len):= s1.str(1..s1.len)&s2;
return buffer;
end "&";

function "&"(s1: character; s2: v_string) return v_string is
buffer: v_string;
begin
buffer.len:= 1 + s2.len;
buffer.str(1..buffer.len):= s1&s2.str(1..s2.len);
return buffer;
end "&";

-- ワイルドカード一致判定（第一引数のみ）
```

```
-- * -- '.' を含まない
-- + -- '.' を含む

-- とりあえず、最後にのみワイルドカード文字が使えるシンプルな
-- やつを (;_;)  S1 -- ワイルドカード使用可
function Match_Wildcard(S1, S2 : V_String) return Boolean is
   S1_L : String := To_Lower(S1.Str);
   S2_L : String := To_Lower(S2.Str);
begin
--     if S2.Len < S1.Len then
--        return False;
--     end if;
   for I in 1..S1.Len loop
      if I > S2.Len or else S1_L(I) /= S2_L(I) then
 case S1_L(I) is
    when '*' =>
       for J in I..S2.Len loop
  if S2_L(J) = '.' then
     return False;
  end if;
       end loop;
       return True;
    when '+' =>
       return True;
    when others =>
       return False;
 end case;
      end if;
   end loop;

   return (S1.Len = S2.Len);
end Match_Wildcard;

function Length(S1 : V_String) return Natural is
begin
   return S1.Len;
end Length;


end V_Strings;
```

C.1.23 Variable_Analyzer Package

```
with Asis;
with Asis.Ids;
with Id_List;
with V_Strings; use V_Strings;
with Spec; use Spec;
with Measure_Types; use Measure_Types;
with List;

package Variable_Analyzer is

   type Info_Source is new Boolean;

--   package Variable_Probe_List is new List(Parameter_List.Node_Link);
--   Component_List を利用

   subtype Variable_Measurement_T is Measurement_T
     range ASSIGNED_VARIABLE..REFERED_VARIABLE;

   type Probes_Array is array(Variable_Measurement_T)
   of Component_List.Node_Link;

   -- ブロック内で使用されている変数
   type Use_Variable is
      record
 -- 呼び出し先の ID
-- Callee_ID : Asis.Ids.Id;
 -- (本当は ID の方がいいけど、ID -> Element の変換が a4g で
 --  実装されていないので渋々 Element)
 Using_Element : Asis.Element;
 -- (本当は ID の方がいいけど、ID -> Element の変換が a4g で
 --  実装されていないので渋々 Element)
 Used_Element : Asis.Element;
 -- 中継関数の名前
 Proxy_Name : V_String := Null_Str;
 -- 型名
 Type_Name : V_String := Null_Str;
 -- 本物の変数の名前
 Genuine_Name : V_String := Null_Str;
 -- 計測種類毎の probe のリスト
 Probes : Probes_Array;
 -- 最初に使用されているのが宣言部であるか否か
 Is_In_Decl : Boolean := False;
 -- Get 変換を必要としているか
 Is_Refer : Boolean := False;
 -- Put 変換を必要としているか
 Is_Assign : Boolean := False;
      end record;
--   Null_Function : Use_Function := (Asis.Ids.Nil_ID, Null_Str);

   type Use_Variable_Link is access Use_Variable;

   package Use_Variable_List is new Id_List(Use_Variable_Link, null);

   -- ID を Index としないリスト (集合)
   package Variable_Set is new List(Use_Variable_Link);
   -- Function_Set の、ID を Index とするリスト
   package Variable_Set_List is new Id_List(Variable_Set.Node_Link, null);


   -- 同じブロック内にある同じ関数への呼び出しをカウントしないため
   package Already_Checked_List is new Id_List(Boolean, False);
   procedure Analyze(Elem : Asis.Element; Current_Block : V_String);
```

```
end Variable_Analyzer;



with Ada.Text_IO;
with Ada.Characters.Handling; use Ada.Characters.Handling;

with Asis;
with Asis.Declarations;
with Asis.Elements;
with Asis.Expressions;
with Asis.Iterator;
with Asis.Statements;
with Asis.Text;
with Gela_Ids;
with Asis_Utils;
with Asis.Exceptions;
with Spec; use Spec;
with Measure_Types; use Measure_Types;
with Spec_Reader;
with V_Strings; use V_Strings;
with Name_Handler; use Name_Handler;
with String_Handler; use String_Handler;
with Designations; use Designations;

package body Variable_Analyzer is

   use Asis;

   -- 中継関数/手続きの識別子のためのカウンタ
   Identlast : Integer := 0;

   -- 宣言の中での変数使用か否か
   Global_Is_Decl : Boolean;

   -- 現在処理している宣言文

   -- 変数を使用している宣言文の ID
   -- (命令文から呼び出している場合は Nil_ID)
   Decl_ID : Gela_Ids.Id := Gela_Ids.Nil_ID;

   procedure Analyze(Elem : Asis.Element; Current_Block : V_String) is

--      Tmp_Use_Functions : Use_Function_Link := null;

   -- 同じブロック内にある同じ変数の使用をカウントしないため
      Already_Checked : Already_Checked_List.A_NODE;

      -- 唯一な識別子を作る
      function Unique_Identifier return String is
 package Int_IO is new Ada.Text_IO.Integer_IO(Integer);
 use Int_IO;
 Result : String(1..5);
      begin
 Put(Result, IdentLast);
 IdentLast := IdentLast + 1;
 for I in Result'Range loop
    if Result(I) = ' ' then
       Result(I) := '0';
    end if;
 end loop;
 return "VAR_" & Result;
      end Unique_Identifier;

      procedure Pre_Source
(Element : in     Asis.Element;
 Control : in out Asis.Traverse_Control;
 State   : in out Info_Source) is

 procedure Set_Probes is
    Tmp_Component : Measurement_Component_Link;
    Tmp_Use_Variable : Use_Variable_Link;

    function Is_Designated return Boolean is
       use Status_List;
       Iter : Iterator_T;
       Tmp_Status : Measurement_Status_Link;
       Result : Boolean := False;
    begin
       Set_List(Iter, Tmp_Component.Status);
       while not Is_End(Iter) loop
  Tmp_Status := Get_Now(Iter);
  if Tmp_Status.Selection = All_Place then
     Result := True;
  elsif Match_Designation(Tmp_Status.Specified_Block_Name, Current_Block) then
     Result := True;
  elsif Match_Designation(Tmp_Status.Specified_Block_Name, "- " & Current_Block) then
     Result := False;
  end if;
  Set_Next(Iter);
       end loop;
       return Result;
    end Is_Designated;

    procedure Condition_Traverse(Kind : Variable_Measurement_T) is
       use Status_List;
       Iter : Iterator_T;
    begin
--       Set_List(Iter, Tmp_Component.Status);
--       while not Is_End(Iter) loop
       if Tmp_Use_Variable = null then
-- (本当は ID の方がいいけど、ID -> Element の変換が a4g で
--  実装されていないのでとりあえず Element)

  Tmp_Use_Variable := new Use_Variable'
    (Element,
     Asis.Expressions.Corresponding_Name_Definition
```

```
     (Element),
     To_V(Unique_Identifier),
     --Tmp_Component.Conversion_Type,
     To_V(Asis_Utils.Var_Type_Name(Element)),
     To_V(Asis_Utils.Variable_Name(Element)),
     (others => null),
     Global_Is_Decl, False, False);

       end if;
       -- use_variable に対する操作
       Component_List.Add_Contents
 (Tmp_Use_Variable.Probes(Kind),
  Tmp_Component);
       if Kind = REFERED_VARIABLE then
  Tmp_Use_Variable.Is_Refer := True;
       else
  Tmp_Use_Variable.Is_Assign := True;
       end if;
       --exit;
       --Set_Next(Iter);
       --end loop;
    end Condition_Traverse;

    use Component_List;
    Iter : Iterator_T;

    function Is_Here (Element : Asis.Element) return Boolean is
    begin
       return (Asis.Elements.Element_Kind (Element) /= Not_An_Element);
    end Is_Here;

    -- 計測対象変数のフルネーム
    Full_Name : V_String := Unit_Name_List.Search_Node
      (Variable_Names, Gela_Ids.Create_Id
       (Asis.Expressions.Corresponding_Name_Definition(Element)));

 begin -- Set_Probes
    -- すでにチェックされている変数か、計測対象になっていない
    -- 変数なら何もしない
    if (not Already_Checked_List.Search_Node
      (Already_Checked, Gela_Ids.Create_ID(Asis.Expressions.Corresponding_Name_Definition(Element)))) or Full_Name = Null_Str then
       -- function に関連した Measurement_T 全部に対して
       for Kind in Variable_Measurement_T loop
--         if Is_Prot_Measure(Kind) and Is_Protected then
--    Ada.Text_IO.Put_Line("prot のほうがヒット");
--         end if;
--         if (not Is_Prot_Measure(Kind)) and (not Is_Protected) then
--    Ada.Text_IO.Put_Line("prot じゃないほうがヒット");
--         end if;

  Set_List(Iter, Spec_Reader.Spec_Data(Kind));
  while not Is_End(Iter) loop
     Tmp_Component := Get_Now(Iter);
     if Equal_Insensitive(Full_Name,
  Tmp_Component.Variable_Name)
       and then
       Is_Designated then
Condition_Traverse(Kind);
     end if;
     Set_Next(Iter);
  end loop;
       end loop;
       if Tmp_Use_Variable /= null then
  Use_Variable_List.Set_Node
    (Current_Unit.Used_Variables,
     Gela_Ids.Create_ID
     (Asis.Expressions.Corresponding_Name_Definition
      (Element)),
     Tmp_Use_Variable);
  if Global_Is_Decl then
     declare
Tmp_Variable_Set : Variable_Set.Node_Link
  := Variable_Set_List.Search_Node
  (Current_Unit.Decl_Used_Variables, Decl_ID);
     begin
Variable_Set.Add_Contents
  (Tmp_Variable_Set, Tmp_Use_Variable);
Variable_Set_List.Set_Node
  (Current_Unit.Decl_Used_Variables, Decl_ID,
   Tmp_Variable_Set);
     end;

  end if;

  Already_Checked_List.Set_Node
    (Already_Checked,
     Gela_Ids.Create_ID
     (Asis.Expressions.Corresponding_Name_Definition(Element)),
     True);
       end if;
    end if;
--    Ada.Text_IO.Put_Line(To_S(Current_Unit.Used_Functions.Object.Proxy_Name));
 end Set_Probes;

 -- ある変数名 (full name) が、計測対象になっているかどうか
 function Is_Needed_To_Register(Name : V_String) return Boolean is
    use Component_List;
    Iter : Iterator_T;
 begin
    for Kind in Variable_Measurement_T loop
       Set_List(Iter, Spec_Reader.Spec_Data(Kind));
       while not Is_End(Iter) loop
  if Equal_Insensitive(Get_Now(Iter).Variable_Name, Name) then
     return True;
  end if;
  Set_Next(Iter);
       end loop;
    end loop;
    return False;
```

```
 end Is_Needed_To_Register;

      begin
 -- 最上位の Element については、何もしない
-- if not State then
    case Asis.Elements.Element_Kind (Element) is
       when Not_An_Element =>
  null;
       when A_Declaration =>
  case Asis.Elements.Declaration_Kind(Element) is
     when A_Task_Type_Declaration |
       A_Protected_Type_Declaration |
       A_Single_Task_Declaration |
       A_Single_Protected_Declaration |
       A_Procedure_Body_Declaration |
       A_Function_Body_Declaration |
       A_Procedure_Declaration |
       A_Function_Declaration |
       A_Package_Declaration |
       A_Package_Body_Declaration |
       A_Task_Body_Declaration |
       A_Protected_Body_Declaration |
       An_Entry_Declaration |
       An_Entry_Body_Declaration |
       A_Generic_Procedure_Declaration |
       A_Generic_Function_Declaration |
       A_Generic_Package_Declaration |
       A_Formal_Procedure_Declaration |
       A_Formal_Function_Declaration |
       A_Formal_Package_Declaration |
       A_Formal_Package_Declaration_With_Box =>
Control := Abandon_Children;
      when A_Variable_Declaration =>
        -- 計測の必要がある変数であれば、フルネームを登録しておく
declare
   D_Names : Asis.Defining_Name_List
     := Asis.Declarations.Names(Element);
   Full_Name : V_String;
begin
   for I in D_Names'RANGE loop
     Full_Name := To_V(Current_Unit_Name) & "."
& To_String
(Asis.Declarations.Defining_Name_Image
(D_Names(I)));
--      Ada.Text_IO.Put_Line(To_S(Full_Name));
      if Is_Needed_To_Register(Full_Name) then
 Unit_Name_List.Set_Node
   (Variable_Names,
    Gela_Ids.Create_Id(D_Names(I)),
    Full_Name);
      end if;
   end loop;
end;

     when others =>
null;
  end case;
       when A_Statement =>
  case Asis.Elements.Statement_Kind(Element) is
     when A_Block_Statement =>
if Asis.Statements.Is_Declare_Block (Element) then
   Control := Abandon_Children;
end if;
      when others =>
null;
  end case;
       when An_Expression =>
  case Asis.Elements.Expression_Kind(Element) is
     when An_Identifier =>
begin
   if Asis_Utils.Is_Variable(Element) then
      Set_Probes;
   end if;
exception
   when ASIS.EXCEPTIONS.ASIS_INAPPROPRIATE_ELEMENT =>
      null;
      -- a4g のバグ対策
      --  attribute reference が function call と
      -- 誤認識されてしまう
end;
     when others =>
null;
  end case;
       when others =>
  null;
    end case;
-- else
--    State := False;
-- end if;

      exception
 when others =>
    Ada.Text_IO.Put_Line(To_String(Asis.Elements.Debug_Image(Element)));
    raise;
      end Pre_Source;

      procedure Post_Source
(Element : in     Asis.Element;
 Control : in out Asis.Traverse_Control;
 State   : in out Info_Source) is
      begin
--   case Asis.Elements.Element_Kind (Element) is
--      when An_Expression =>
--         case Asis.Elements.Expression_Kind(Element) is
--    when A_Function_Call =>
 null;
      end Post_Source;

      procedure Traverse_Block is new Asis.Iterator.Traverse_Element
```

```
(Info_Source, Pre_Source, Post_Source);

      procedure Run_Traverse(Element : in Asis.Element) is
 The_Control : Asis.Traverse_Control := Asis.Continue ;
 The_Spec_Information : Info_Source := True;
      begin
--   if Is_Decl then
--      Decl_ID := Gela_Ids.Create_ID(Element);
--   else
--      Decl_ID := Gela_Ids.Nil_ID;
--   end if;
 Traverse_Block(Element, The_Control, The_Spec_Information);
      end Run_Traverse;

      function Decl_List return Asis.Element_List is
      begin
 case Asis.Elements.Element_Kind(Elem) is
    when A_Statement =>
       return Asis.Statements.Block_Declarative_Items(Elem);
    when A_Declaration =>
       return Asis.Declarations.Body_Declarative_Items(Elem);
    when others =>
       return Nil_Element_List;
 end case;
      end Decl_List;

      function Stmt_List return Asis.Element_List is
      begin
 case Asis.Elements.Element_Kind(Elem) is
    when A_Statement =>
       return Asis.Statements.Block_Statements(Elem);
    when A_Declaration =>
       return Asis.Declarations.Body_Statements(Elem);
    when others =>
       return Nil_Element_List;
 end case;
      end Stmt_List;

      Decls : Asis.Element_List := Decl_List;
      Stmts : Asis.Element_List := Stmt_List;

   begin
      Global_Is_Decl := True;
      for I in Decls'RANGE loop
 Decl_ID := Gela_Ids.Create_ID(Decls(I));
 case Asis.Elements.Declaration_Kind(Decls(I)) is
    when A_Task_Type_Declaration |
      A_Protected_Type_Declaration =>
       declare
  Disc_Part : Asis.Element := Asis.Declarations.Discriminant_Part(Decls(I));
       begin
  if not Asis.Elements.Is_Nil(Disc_Part) then
     Run_Traverse(Disc_Part);
  end if;
       end;

    when A_Procedure_Body_Declaration |
      A_Function_Body_Declaration |
      A_Procedure_Declaration |
      A_Function_Declaration |
      An_Entry_Declaration |
      An_Entry_Body_Declaration |
      A_Generic_Procedure_Declaration |
      A_Generic_Function_Declaration |
      A_Formal_Procedure_Declaration |
      A_Formal_Function_Declaration =>
       declare
  Params : Asis.Parameter_Specification_List
    := Asis.Declarations.Parameter_Profile(Decls(I));
  Initial_Exp : Asis.Element;
       begin
  for I in Params'RANGE loop
     Initial_Exp := Asis.Declarations.Initialization_Expression
       (Params(I));
     if not Asis.Elements.Is_Nil(Initial_Exp) then
Run_Traverse(Initial_Exp);
     end if;
  end loop;
       end;


    when A_Single_Task_Declaration |
      A_Single_Protected_Declaration |
      A_Package_Declaration |
      A_Package_Body_Declaration |
      A_Task_Body_Declaration |
      A_Protected_Body_Declaration |
      A_Generic_Package_Declaration |
      A_Formal_Package_Declaration |
      A_Formal_Package_Declaration_With_Box =>
       null;
    when others =>
       Run_Traverse(Decls(I));
 end case;
      end loop;
      -- 命令文のときは、Nil_ID
      Global_Is_Decl := False;
      Decl_ID := Gela_Ids.Nil_ID;
      for I in Stmts'RANGE loop
 case Asis.Elements.Statement_Kind(Stmts(I)) is
    when A_Block_Statement =>
       if not Asis.Statements.Is_Declare_Block (Stmts(I)) then
  Run_Traverse(Stmts(I));
       end if;
    when others =>
       Run_Traverse(stmts(I));
 end case;
      end loop;
--      Current_Unit.
```

```
   end Analyze;

end Variable_Analyzer;
```

C.2 Run-Time Detection Tool

C.2.1 Dd_Spec Package

```
with Global_Types, Pid;
use Global_Types, Pid;

with Ada.Task_Identification;
use Ada.Task_Identification;

package Dd_Spec is

   function BLOCK_ELABORATION_START(This_Unit_Class : Unit_Class; Block_Name : String) return Boolean;
   function BLOCK_ELABORATION_COMPLETION return Boolean;
   function LIBRARY_PACKAGE_ELABORATION_COMPLETION return Boolean;
   procedure BLOCK_EXECUTION_START;
   procedure BLOCK_EXECUTION_COMPLETION;

   procedure SIMPLE_ENTRY_CALL(Callee_Task : in Task_ID;
       Entry_Name : in String);
   procedure PROTECTED_PROCEDURE_CALL(Callee_Protected_ID : in Protected_ID;
       Subprogram_Name : in String);
--   procedure PROTECTED_SUBPROGRAM_CALL_INTERNAL(
--                                  Subprog_Name : in STRING_BUFFER);
   procedure PROTECTED_PROCEDURE_CALL_COMPLETION;
   procedure PROTECTED_ENTRY_CALL(Callee_Protected_ID : in Protected_ID;
  Entry_Name : in String);
   procedure PROTECTED_ENTRY_CALL_COMPLETION;
   procedure PROCEDURE_CALL_COMPLETION;
   procedure FUNCTION_CALL_COMPLETION;
   procedure PROTECTED_FUNCTION_CALL(Callee_Protected_ID : in Protected_ID;
       Subprogram_Name : in String);
   procedure PROTECTED_FUNCTION_CALL_COMPLETION;
   procedure BLOCK_CALL_COMPLETION;
   procedure ABORT_START(Abort_Tasks : in Task_ID_List);
--   procedure PROTECTED_BLOCK_EXECUTION_START;
--   procedure PROTECTED_BLOCK_EXECUTION_END;
   function TASK_ACTIVATION_START
     (Block_Name : String;
      Parent_Task : Task_ID;
      This_Unit_Class : Unit_Class; Unit_ID_Of_Task : Unit_ID) return Boolean;
   function TASK_ACTIVATION_COMPLETION return Boolean;

   procedure ACCEPT_START(Entry_Name : String; Is_Open_Accept : Boolean);
   procedure RENDEZVOUS_START(Caller_Task : Task_ID);
   procedure RENDEZVOUS_END(Caller_Task : Task_ID);

   procedure ACCEPT_START_WITH_TERMINATE
     (Entry_Name : String; Is_Open_Accept : Boolean);
   procedure ASYNCHRONOUS_ENTRY_CALL(Callee_Task : in Task_ID;
     Entry_Name : in String);
   procedure ASYNCHRONOUS_PROTECTED_ENTRY_CALL(Callee_Task : in Task_ID;
     Entry_Name : in String);
   procedure ASYNCHRONOUS_DELAY;
   procedure ASYNCHRONOUS_TRIGGER_END;
   procedure ASYNCHRONOUS_SELECT_END;

   procedure CONDITIONAL_PROTECTED_ENTRY_CALL;
   procedure TIMED_PROTECTED_ENTRY_CALL;
   procedure CONDITIONAL_ENTRY_CALL;
   procedure TIMED_ENTRY_CALL;
   function GET_CDT(CDT_OF_TASK : CDT; CDT_OF_TASK_TYPE : CDT_For_TT) return
     Boolean;

   procedure REQUEUE_START(Callee_Task : in Task_ID;
   Entry_Name : in String);
   procedure REQUEUE_WITH_ABORT_START(Callee_Task : in Task_ID;
      Entry_Name : in String);
   procedure PROTECTED_REQUEUE_START(Callee_Protected_ID : Protected_ID;
     Entry_Name : in String);
   procedure PROTECTED_REQUEUE_WITH_ABORT_START
     (Callee_Protected_ID : Protected_ID;
      Entry_Name : in String);
--   function GET_UNIT_ID(UNIT_ID_OF_TASK : Unit_ID) return Boolean;
end Dd_Spec;



with Global_Types, Pid;
use Global_Types, Pid;

with Ada.Task_Identification;
use Ada.Task_Identification;

with Event_Driven_Execution_Monitor3;
use Event_Driven_Execution_Monitor3;

with Ada.Text_IO; use Ada.Text_IO;

package body Dd_Spec is

   function BLOCK_ELABORATION_START(This_Unit_Class : Unit_Class; Block_Name : String) return Boolean is
   begin
      TIC.BLOCK_ELABORATION_START(This_Unit_Class, Block_Name);
      return True;
   end BLOCK_ELABORATION_START;

   function BLOCK_ELABORATION_COMPLETION return Boolean is
   begin
      TIC.BLOCK_ELABORATION_COMPLETION;
```

```
      return True;
   end BLOCK_ELABORATION_COMPLETION;

   function LIBRARY_PACKAGE_ELABORATION_COMPLETION return Boolean is
   begin
      TIC.LIBRARY_PACKAGE_ELABORATION_COMPLETION;
      return True;
   end LIBRARY_PACKAGE_ELABORATION_COMPLETION;

   procedure BLOCK_EXECUTION_START is
   begin
      TIC.BLOCK_EXECUTION_START;
   end BLOCK_EXECUTION_START;

   procedure BLOCK_EXECUTION_COMPLETION is
   begin
      TIC.BLOCK_EXECUTION_COMPLETION;
   end BLOCK_EXECUTION_COMPLETION;

   procedure SIMPLE_ENTRY_CALL(Callee_Task : in Task_ID;
       Entry_Name : in String) is
   begin
      TIC.SIMPLE_ENTRY_CALL(Callee_Task, Entry_Name);
   end SIMPLE_ENTRY_CALL;

   procedure PROTECTED_PROCEDURE_CALL(Callee_Protected_ID : in Protected_ID;
       Subprogram_Name : in String) is
   begin
      TIC.PROTECTED_SUBPROGRAM_CALL(Callee_Protected_ID, Subprogram_Name,
    PROTECTED_PROCEDURES);
   end PROTECTED_PROCEDURE_CALL;
--   procedure PROTECTED_SUBPROGRAM_CALL_INTERNAL(
--                                        Subprog_Name : in STRING_BUFFER);
   procedure PROTECTED_PROCEDURE_CALL_COMPLETION is
   begin
      TIC.PROTECTED_SUBPROGRAM_CALL_COMPLETION(PROTECTED_PROCEDURES);
   end PROTECTED_PROCEDURE_CALL_COMPLETION;
   procedure PROTECTED_ENTRY_CALL(Callee_Protected_ID : in Protected_ID;
  Entry_Name : in String) is
   begin
      TIC.PROTECTED_ENTRY_CALL(Callee_Protected_ID, Entry_Name);
   end PROTECTED_ENTRY_CALL;

   procedure PROTECTED_ENTRY_CALL_COMPLETION is
   begin
      TIC.PROTECTED_ENTRY_CALL_COMPLETION;
   end PROTECTED_ENTRY_CALL_COMPLETION;

   procedure PROCEDURE_CALL_COMPLETION is
   begin
      TIC.SUBPROGRAM_CALL_COMPLETION(PROCEDURES);
   end PROCEDURE_CALL_COMPLETION;

   procedure FUNCTION_CALL_COMPLETION is
   begin
      TIC.SUBPROGRAM_CALL_COMPLETION(FUNCTIONS);
   end FUNCTION_CALL_COMPLETION;

   procedure PROTECTED_FUNCTION_CALL(Callee_Protected_ID : in Protected_ID;
       Subprogram_Name : in String) is
   begin
      TIC.PROTECTED_SUBPROGRAM_CALL(Callee_Protected_ID, Subprogram_Name,
    PROTECTED_FUNCTIONS);
   end PROTECTED_FUNCTION_CALL;

   procedure PROTECTED_FUNCTION_CALL_COMPLETION is
   begin
      TIC.PROTECTED_SUBPROGRAM_CALL_COMPLETION(PROTECTED_FUNCTIONS);
   end PROTECTED_FUNCTION_CALL_COMPLETION;

   procedure BLOCK_CALL_COMPLETION is
   begin
      TIC.BLOCK_CALL_COMPLETION;
   end BLOCK_CALL_COMPLETION;

   procedure ABORT_START(Abort_Tasks : in Task_ID_List) is
   begin
      TIC.ABORT_START(Abort_Tasks);
   end ABORT_START;
--   procedure PROTECTED_BLOCK_EXECUTION_START;
--   procedure PROTECTED_BLOCK_EXECUTION_END;
   function TASK_ACTIVATION_START
     (Block_Name : String;
      Parent_Task : Task_ID;
      This_Unit_Class : Unit_Class; Unit_ID_Of_Task : Unit_ID)
      return Boolean is
   begin
      TIC.TASK_ACTIVATION_START(Block_Name, Parent_Task, This_Unit_Class,
Unit_ID_Of_Task);
      return True;
   end TASK_ACTIVATION_START;

   function TASK_ACTIVATION_COMPLETION return Boolean is
   begin
      TIC.TASK_ACTIVATION_COMPLETION;
      return True;
   end TASK_ACTIVATION_COMPLETION;

   procedure ACCEPT_START(Entry_Name : String; Is_Open_Accept : Boolean) is
   begin
      TIC.ACCEPT_START(Entry_Name, Is_Open_Accept);
   end ACCEPT_START;

   procedure RENDEZVOUS_START(Caller_Task : Task_ID) is
   begin
      TIC.RENDEZVOUS_START(Caller_Task);
   end RENDEZVOUS_START;
   procedure RENDEZVOUS_END(Caller_Task : Task_ID) is
   begin
```

```
      TIC.RENDEZVOUS_END(Caller_Task);
   end RENDEZVOUS_END;

   procedure ACCEPT_START_WITH_TERMINATE
   (Entry_Name : String; Is_Open_Accept : Boolean) is
   begin
      TIC.ACCEPT_WITH_TERMINATE_START(Entry_Name, Is_Open_Accept);
   end ACCEPT_START_WITH_TERMINATE;

   procedure ASYNCHRONOUS_ENTRY_CALL(Callee_Task : in Task_ID;
     Entry_Name : in String) is
   begin
      TIC.ASYNCHRONOUS_ENTRY_CALL(Callee_Task, Entry_Name);
   end ASYNCHRONOUS_ENTRY_CALL;

   procedure ASYNCHRONOUS_PROTECTED_ENTRY_CALL(Callee_Task : in Task_ID;
     Entry_Name : in String) is
   begin
      TIC.ASYNCHRONOUS_PROTECTED_ENTRY_CALL(Callee_Task, Entry_Name);
   end ASYNCHRONOUS_PROTECTED_ENTRY_CALL;

   procedure ASYNCHRONOUS_DELAY is
   begin
      TIC.ASYNCHRONOUS_DELAY_STMT;
   end ASYNCHRONOUS_DELAY;

   procedure ASYNCHRONOUS_TRIGGER_END is
   begin
      TIC.ASYNCHRONOUS_TRIGGER_END;
   end ASYNCHRONOUS_TRIGGER_END;

   procedure ASYNCHRONOUS_SELECT_END is
   begin
      TIC.ASYNCHRONOUS_SELECT_END;
   end ASYNCHRONOUS_SELECT_END;

   procedure CONDITIONAL_PROTECTED_ENTRY_CALL is
   begin
      TIC.CONDITIONAL_ENTRY_CALL;
   end CONDITIONAL_PROTECTED_ENTRY_CALL;
   procedure TIMED_PROTECTED_ENTRY_CALL is
   begin
      TIC.TIMED_ENTRY_CALL;
   end TIMED_PROTECTED_ENTRY_CALL;

   procedure CONDITIONAL_ENTRY_CALL is
   begin
      TIC.CONDITIONAL_ENTRY_CALL;
   end CONDITIONAL_ENTRY_CALL;

   procedure TIMED_ENTRY_CALL is
   begin
      TIC.TIMED_ENTRY_CALL;
   end TIMED_ENTRY_CALL;

   function GET_CDT(CDT_OF_TASK : CDT; CDT_OF_TASK_TYPE : CDT_For_TT) return
     Boolean is
   begin
      TIC.GET_CDT(CDT_OF_TASK, CDT_OF_TASK_TYPE);
      return True;
   end GET_CDT;

   procedure REQUEUE_START(Callee_Task : in Task_ID;
   Entry_Name : in String) is
   begin
      TIC.REQUEUE_EXEC(Callee_Task, Entry_Name);
   end REQUEUE_START;

   procedure REQUEUE_WITH_ABORT_START(Callee_Task : in Task_ID;
      Entry_Name : in String) is
   begin
      TIC.REQUEUE_WITH_ABORT_EXEC(Callee_Task, Entry_Name);
   end REQUEUE_WITH_ABORT_START;

   procedure PROTECTED_REQUEUE_START(Callee_Protected_ID : Protected_ID;
     Entry_Name : in String) is
   begin
      TIC.PROTECTED_REQUEUE_EXEC(Callee_Protected_ID, Entry_Name);
   end PROTECTED_REQUEUE_START;

   procedure PROTECTED_REQUEUE_WITH_ABORT_START
     (Callee_Protected_ID : Protected_ID;
      Entry_Name : in String) is
   begin
      TIC.PROTECTED_REQUEUE_WITH_ABORT_EXEC(Callee_Protected_ID, Entry_Name);
   end PROTECTED_REQUEUE_WITH_ABORT_START;
--   function GET_UNIT_ID(UNIT_ID_OF_TASK : Unit_ID) return Boolean;

end Dd_Spec;
```

C.2.2 Event_Driven_Execution_Monitor3 Package

```
with Ada.Task_Identification;
with Global_Types;
use  Ada.Task_Identification;
with Ada.Calendar;            use Ada.Calendar;
with Ada.Sequential_IO;
with Ada.Characters.Handling; use Ada.Characters.Handling;
with Global_Types;            use Global_Types;
with Pid;                     use Pid;
with Task_Information_Collector;
with Task_Wait_For_Graph_Manager;

package Event_Driven_Execution_Monitor3 is

   TIC : Task_Information_Collector.TIC;
```

```
end Event_Driven_Execution_Monitor3;
```

C.2.3 Global_Types Package

```
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Text_IO;             use Ada.Text_IO;
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Characters.Handling; use Ada.Characters.Handling;
with Pid;                     use Pid;
with Task_Indexed_List;

package Global_Types is

   Max_Entry_Name : constant Natural := 256;

   subtype Entry_Name_Length is Integer range 0..Max_Entry_Name;

   -- 各ユニットに固有な ID
   type Unit_ID is new Natural;

   type UNIT_CLASS is (N_U_L_L,
       RESERVED,
       COMPLETED,
       MAIN_PROCEDURE,
       MAIN_TASK,
       TASKS,
       BLOCKS,
       PROCEDURES,
       FUNCTIONS,
       PROTECTED_PROCEDURES,
       PROTECTED_FUNCTIONS,
       PROTECTED_OBJECTS,
       PROTECTED_ENTRY_BODIES,
       LIBRARY_PACKAGES);

   type Task_ID_List is array(Integer range <>) of Task_ID;

   type CDT_Element is
      record
 Callee_Task : Task_ID;
 Entry_Name : String(1..Max_Entry_Name);
 Last : Entry_Name_Length := 0;
      end record;

   Null_CDT_Element : CDT_Element := (Null_Task_ID, (1..Max_Entry_Name => Ascii.Nul), 0);

   type CDT is array(Integer range <>) of CDT_Element;

   Null_CDT : CDT := (1..0 => Null_CDT_Element);

   -- task type のための
   type CDT_Element_For_TT is
      record
 Callee_Unit_ID : Unit_ID;
 Entry_Name : String(1..Max_Entry_Name);
 Last : Entry_Name_Length := 0;
      end record;

   Null_CDT_Element_For_TT : CDT_Element_For_TT := (0, (1..Max_Entry_Name => Ascii.Nul), 0);

   type CDT_For_TT is array(Integer range <>) of CDT_Element_For_TT;

   Null_CDT_For_TT : CDT_For_TT := (1..0 => Null_CDT_Element_For_TT);

   --   Maximum_Master : constant := 50;
   --   type ID_OF_MASTER is new Natural range 0..Maximum_Master;
   --   Null_Master_ID : ID_OF_MASTER := 0;
   Maximum_String_Length : constant := 200;
   type STRING_BUFFER is record
      Text : String (1 .. Maximum_String_Length);
      Last : Natural := 0;
   end record;

   type NULL_TYPE is new Natural;
   --   NULL_DUMMY : ID_OF_MASTER;
   function "=" (Left, Right : STRING_BUFFER) return Boolean;

   function Get_String_Buffer (Input_String : String) return STRING_BUFFER;
   procedure Put_String (Arg : in STRING_BUFFER);
   procedure Set (Result : out STRING_BUFFER; Input_String : in String);
   package Int_IO is new Ada.Text_IO.Integer_IO (Integer);
   use Int_IO;
   package Duration_IO is new Ada.Text_IO.Fixed_IO (Duration);
   use Duration_IO;

   Debug_Mode : Boolean := False;

   type ARC_CLASS is (
      NO_ARC,
      ACTIVATION_WAITING_ARC,
      ACCEPTANCE_WAITING_ARC,
      COMPLETION_WAITING_ARC,
      ENTRY_CALLING_ARC,
      FINALIZATION_WAITING_ARC,
      PROTECTION_WAITING_ARC,
      PROTECTED_ENTRY_CALLING_ARC);

   --   Arc_Class_Names : array(ARC_CLASS) of STRING_BUFFER;
   --   Unit_Class_Names : array(UNIT_CLASS) of STRING_BUFFER;

   type Status_Of_Master is (
     NOT_EXIST,
     ELABORATION_START,
     ELABORATION_COMPLETION,
     ACTIVATION_START,
     SUBTASK_DECLARATION_EVALUATION_START,
```

```
  SUBTASK_DECLARATION_EVALUATION_COMPLETION,
  ACTIVATION_COMPLETION,
  EXECUTION_START,
  SUBTASK_ALLOCATOR_EVALUATION_START,
  SUBTASK_ALLOCATOR_EVALUATION_COMPLETION,
  SIMPLE_ENTRY_CALL,
  CONDITIONAL_ENTRY_CALL,
  TIMED_ENTRY_CALL,
  ENTRY_CALL_CANCELLATION,
  ACCEPTANCE,
  SIMPLE_SELECTION,
  CONDITIONAL_SELECTION,
  TIMED_SELECTION,
  TERMINATION_SELECTION,
  SELECTION_CANCELLATION,
  RENDEZVOUS_START,
  CONTINUATION,
  ABORTS,
  ABORTED,
  LIBRARY_PACKAGE_ELABORATION_START,
  LIBRARY_PACKAGE_ELABORATION_COMPLETION,
  BLOCK_ELABORATION_START,
  BLOCK_ELABORATION_COMPLETION,
  BLOCK_EXECUTION_START,
  BLOCK_EXECUTION_COMPLETION,
  BLOCK_TERMINATION,
  ACTIVATION_EXCEPTION,
  EXECUTION_EXCEPTION,
  COMMUNICATION_EXCEPTION,
  COMPLETION,
  TERMINATION,
  PROTECTED_OBJECT_EXIST,
  BLOCKED_BY_PROTECTION,
  PROTECTED_ENTRY_CALL,
  ASYNCHRONOUS_ENTRY_CALL,
  ASYNCHRONOUS_DELAY,
  PROTECTED_ENTRY_BLOCK_ELABORATION_START,
  PROTECTED_ENTRY_BLOCK_ELABORATION_COMPLETION,
  PROTECTED_SUBPROGRAM_ELABORATION_START,
  PROTECTED_SUBPROGRAM_ELABORATION_COMPLETION);

type Status_Of_Task is (
ELABORATING,
-- (ELABORATION_START, ELABORATION_COMPLETION)

ELABORATED,
-- (ELABORATION_COMPLETION, ACTIVATION_START)
-- (ELABORATION_COMPLETION, ABORTED)

ACTIVATING,
-- (ACTIVATION_START, ACTIVATION_COMPLETION)
-- (ACTIVATION_START, SUBTASK_DECLARATION_EVALUATION_START)
-- (SUBTASK_DECLARATION_EVALUATION_COMPLETION,
--  SUBTASK_DECLARATION_EVALUATION_START)
-- (SUBTASK_DECLARATION_EVALUATION_COMPLETION,
--  ACTIVATION_COMPLETION)
-- (ACTIVATION_START, ABORTED)

SUBTASK_DECLARATION_EVALUATING,
-- (SUBTASK_DECLARATION_EVALUATION_START,
--  SUBTASK_DECLARATION_EVALUATION_COMPLETION)
-- (SUBTASK_DECLARATION_EVALUATION_START, ABORTED)

EXECUTION_WAITING,
-- (ACTIVATION_COMPLETION, EXECUTION_START)
-- (ACTIVATION_COMPLETION, ABORTED)

SUBTASK_ALLOCATOR_EVALUATING,
-- (SUBTASK_ALLOCATOR_EVALUATION_START,
--  SUBTASK_ALLOCATOR_EVALUATION_COMPLETION)
-- (SUBTASK_ALLOCATOR_EVALUATION_START, ABORTED)

SIMPLE_ENTRY_CALLING,
-- (SIMPLE_ENTRY_CALL, RENDEZVOUS_START)
-- (SIMPLE_ENTRY_CALL, ABORTED)

CONDITIONAL_ENTRY_CALLING,
-- (CONDITIONAL_ENTRY_CALL, RENDEZVOUS_START)
-- (CONDITIONAL_ENTRY_CALL, ENTRY_CALL_CANCELLATION)
-- (CONDITIONAL_ENTRY_CALL, ABORTED)

TIMED_ENTRY_CALLING,
-- (TIMED_ENTRY_CALL, RENDEZVOUS_START)
-- (TIMED_ENTRY_CALL, ENTRY_CALL_CANCELLATION)
-- (TIMED_ENTRY_CALL, ABORTED)

ACCEPTING,
-- (ACCEPTANCE, RENDEZVOUS_START)
-- (ACCEPTANCE, ABORTED)

SIMPLE_SELECTING,
-- (SIMPLE_SELECTION, RENDEZVOUS_START)
-- (SIMPLE_SELECTION, ABORTED)

CONDITIONAL_SELECTING,
-- (CONDITIONAL_SELECTION, RENDEZVOUS_START)
-- (CONDITIONAL_SELECTION, SELECTION_CANCELLATION)
-- (CONDITIONAL_SELECTION, ABORTED)

TIMED_SELECTING,
-- (TIMED_SELECTION, RENDEZVOUS_START)
-- (TIMED_SELECTION, SELECTION_CANCELLATION)
-- (TIMED_SELECTION, ABORTED)

TERMINATION_SELECTING,
-- (TERMINATION_SELECTION, RENDEZVOUS_START)
-- (TERMINATION_SELECTION, TERMINATION)
-- (TERMINATION_SELECTION, ABORTED)
```

```
TERMINATION_WAITING,
-- (TERMINATION_SELECTION, TERMINATION)
-- (TERMINATION_SELECTION, ABORTED)

SUSPENDED_BY_RENDEZVOUS,
-- (RENDEZVOUS_START, CONTIUATION)
-- (RENDEZVOUS_START, ABORTED)

ABNORMAL,
-- (ABORTED, COMPLETION)

LIBRARY_PACKAGE_ELABORATING,
-- (LIBRARY_PACKAGE_ELABORATION_START,
--  LIBRARY_PACKAGE_ELABORATION_COMPLETION)
-- (LIBRARY_PACKAGE_ELABORATION_START,
--  SUBTASK_DECLARATION_EVALUATION_START)
-- (SUBTASK_DECLARATION_EVALUATION_COMPLETION,
--  SUBTASK_DECLARATION_EVALUATION_START)
-- (SUBTASK_DECLARATION_EVALUATION_COMPLETION,
--  LIBRARY_PACKAGE_ELABORATION_COMPLETION)

BLOCK_ELABORATING,
PROTECTED_ENTRY_BLOCK_ELABORATING,
PROTECTED_SUBPROGRAM_ELABORATING,
-- (BLOCK_ELABORATION_START, BLOCK_ELABORATION_COMPLETION)
-- (BLOCK_ELABORATION_START,
--  SUBTASK_DECLARATION_EVALUATION_START)
-- (SUBTASK_DECLARATION_EVALUATION_COMPLETION,
--  SUBTASK_DECLARATION_EVALUATION_START)
-- (SUBTASK_DECLARATION_EVALUATION_COMPLETION,
--  BLOCK_ELABORATION_COMPLETION)
-- (BLOCK_ELABORATION_START, ABORTED)

BLOCK_EXECUTION_WAITING,
PROTECTED_SUBPROGRAM_EXECUTION_WAITING,
-- (BLOCK_ELABORATION_COMPLETION, BLOCK_EXECUTION_START)
-- (BLOCK_ELABORATION_COMPLETION, ABORTED)

BLOCK_COMPLETED,
PROTECTED_SUBPROGRAM_COMPLETED,
-- (BLOCK_EXECUTION_COMPLETION, BLOCK_TERMINATION)
-- (BLOCK_EXECUTION_COMPLETION, ABORTED)

COMPLETED,
-- (COMPLETION, TERMINATION)

WORKING_FOR_INTERNAL_AFFAIRS,
-- OTHERWISE

BLOCKED_BY_PROTECTION,
PROTECTED_ENTRY_CALLING,
TERMINATED);

package Master_State_IO is new Ada.Text_IO.Enumeration_IO (
   Status_Of_Master);
use Master_State_IO;
package Task_State_IO is new Ada.Text_IO.Enumeration_IO (Status_Of_Task);
use Task_State_IO;

--    type TAG is range 0..1;

--    type NODE_ARRAY is array (1..Maximum_Master) of ID_OF_MASTER;

--    type PATH_ARRAY is
--       record
--   NODE   : NODE_ARRAY := (others => 0);
--   LENGTH : NATURAL range 0..Maximum_Master := 0;
--       end record;

--    type FLAG_ARRAY is array(ID_OF_MASTER) of Boolean;

-- from prepre data

Maximum_Element : constant := 50;

--    subtype ID_OF_ENTRY is Natural range 0..Maximum_Element;
--    Null_Entry_Id : ID_OF_ENTRY := 0;

--    type CNA is array(1..Maximum_Element) of STRING_BUFFER;
--    type STNA is array(1..Maximum_Element) of STRING_BUFFER;
--    type SENA is array(1..Maximum_Element, 1..Maximum_Element) of
--STRING_BUFFER;
--    type SECA is array(1..Maximum_Element) of Natural;
--    type ECT is  array(1..Maximum_Element, 1..Maximum_Element,
--1..Maximum_Element) of Boolean;
--   -- client, server, entry

--    Client_Names : CNA;
--    Client_Count : Natural := 0;
--    Server_Task_Names : STNA;
--    Server_Task_Count : Natural := 0;
--    Server_Entry_Names : SENA;
--    Server_Entry_Count : SECA := (others => 0);
--    Entry_Call_Table : ECT := (others => (others => ( others =>false )));

type Node_List;
type Node_List_Link is access Node_List;

Master_ID_Counter : Integer := 1;

type Master_Index;
type Master_Index_Link is access Master_Index;

type Node_List is record
   Next     : Node_List_Link    := null;
   Mynode   : Master_Index_Link := null;
   Arcclass : ARC_CLASS;
   Arclabel : STRING_BUFFER;
end record;
```

```
   type Async_Type is (NO_ASYNC, ASYNC_DELAY_STMT, ASYNC_ENTRY_CALL);

   type Asynchronous_State is record
      Status         : Async_Type       := NO_ASYNC;
      Trigger_Master : Master_Index_Link := null;
      -- index is Abortable_Master.
   end record;

   type Master_Index is record
      Master_Class  : UNIT_CLASS        := N_U_L_L;
      Parent_Master : Master_Index_Link := null;
      Master_Name   : STRING_BUFFER;
      Master_Status : Status_Of_Master  := NOT_EXIST;
      -- Calling_Entry : ID_OF_ENTRY := Null_Entry_ID;
      Calling_Entry           : STRING_BUFFER;
      Running_Task_ID         : Task_Id           := Null_Task_Id;
      Protected_Object_Master : Master_Index_Link := null;
      Recursive_Count         : Integer           := 0;
      Asyncstate              : Asynchronous_State;
      ProtectedID             : Protected_ID;
      Nextnodes               : Node_List_Link    := null;
      Prevnodes               : Node_List_Link    := null;
      Seq                     : Integer           := 0;
      -- Seq は、パスの存在確認およびある地点までの最短経路
      --を求めるために用いる
      Master_ID : Integer := 0;
   end record;

   -- master_index を生成するときに、protected object
   --だったら必ずこのリストに
   -- 追加する。消滅するときは消す。
   Protected_Object_List : Node_List_Link := null;

   subtype CDT_Monitor is CDT (1 .. Maximum_Element);
   subtype CDT_For_TT_Monitor is CDT_For_TT (1 .. Maximum_Element);
   type Task_Index is record
      Original_Master     : Master_Index_Link  := null;
      Current_Master      : Master_Index_Link  := null;
      Entry_Caller_Master : Master_Index_Link  := null;
      Server_ID           : Natural            := 0;
      Task_Status         : Status_Of_Task     := ELABORATING;
      Completed_Master    : Master_Index_Link  := null;
      Task_CDT            : CDT_Monitor        :=
(others => Null_CDT_Element);
      Task_CDT_For_TT     : CDT_For_TT_Monitor :=
(others => Null_CDT_Element_For_TT);
      Source_Unit_ID      : Unit_ID;
      -- タスクのノードがまだ生成されていない段階でその緕
      --ッ繝ン繝牙ョ帙↓
      -- 緕、緕代ｉ繧後◆譖 prev)
      -- ノード生成時に全て処理する
      Standby_Arcs : Node_List_Link := null;
   end record;
   type Task_Index_Link is access Task_Index;
   --    Null_Task_Index : Task_Index := (null, null,
   --      null, 0, ELABORATING,
   --      null,
   --      (others => Null_CDT_Element),
   --      (others => Null_CDT_Element_For_TT),
   --      0);

   Null_String_Buffer : STRING_BUFFER :=
     (Text => (1 .. Maximum_String_Length => ' '),
      Last => 0);

   --    type Array_Of_Master_Index is array(ID_OF_MASTER) of Master_Index;

   --    type Array_Of_Arc_Class is array (ID_OF_MASTER, ID_OF_MASTER) of
   --ARC_CLASS;

   --    type Array_Of_Tag is array (ID_OF_MASTER, ID_OF_MASTER) of TAG;

   --    type Array_Of_Label is array(ID_OF_MASTER, ID_OF_MASTER) of
   --STRING_BUFFER;

   --   type Array_Of_Async_State is array(ID_OF_MASTER) of Asynchronous_State;

   type Array_Of_Protected_Id is array (Protected_ID) of Master_Index_Link;

   package Task_Index_List is new Task_Indexed_List (Task_Index_Link, null);

end Global_Types;



package body Global_Types is
   function "=" (Left, Right : STRING_BUFFER) return Boolean is
      Result : Boolean;
   begin -- "="
      Result :=
        ((Left.Last = Right.Last) and
         (To_Lower (Left.Text (1 .. Left.Last)) =
          To_Lower (Right.Text (1 .. Right.Last))));
      return Result;
   end "=";

   --   -- Input ECT from tempolary file

   --    procedure Input_ECT is
   --       type SaveData is record
   --   S_Client_Names : CNA;
   --   S_Client_Count : Natural;
   --   S_Server_Task_Names : STNA;
   --   S_Server_Task_Count : Natural;
   --   S_Server_Entry_Names : SENA;
   --   S_Server_Entry_Count : SECA;
   --   S_Entry_Call_Table : ECT;
   --       end record;
```

```
--       TempData: SaveData;
--       package io is new Ada.Sequential_io(SaveData);
--       fd : io.File_type;
--    begin -- Input_ECT
--       io.open(File => fd, Mode=> io.IN_FILE, Name=> "temp_CDT" );
--       io.read(fd, TempData);
--       io.close(fd);
--       Client_Names := TempData.S_Client_Names;
--       Client_Count := TempData.S_Client_Count;
--       Server_Task_Names := TempData.S_Server_Task_Names;
--       Server_Task_Count := TempData.S_Server_Task_Count;
--       Server_Entry_Names := TempData.S_Server_Entry_Names;
--       Server_Entry_Count := TempData.S_Server_Entry_Count;
--       Entry_Call_Table := TempData.S_Entry_Call_Table;
--       Put_Line("Entry-Call table loading is completed.");
--    exception
--       when Name_Error =>
--   put_line("Can not open temp_ECT");
--    end Input_ECT;

function Get_String_Buffer (Input_String : String) return STRING_BUFFER is
   Temp_Result : STRING_BUFFER;
begin -- Get_String_Buffer
   if Input_String'Length <= Maximum_String_Length then
      Temp_Result.Last                          := Input_String'Length;
      Temp_Result.Text (1 .. Temp_Result.Last) := Input_String;
   else
      Temp_Result.Last                          := Maximum_String_Length;
      Temp_Result.Text (1 .. Temp_Result.Last) :=
        Input_String (
        Input_String'First ..
        Input_String'First - 1 + Maximum_String_Length);
   end if;
   return Temp_Result;
end Get_String_Buffer;

procedure Put_String (Arg : in STRING_BUFFER) is
begin -- Put_String
   if Arg.Last > 0 then
      Put (Arg.Text (1 .. Arg.Last));
   end if;
end Put_String;

procedure Set (Result : out STRING_BUFFER; Input_String : in String) is
   Temp_Result : STRING_BUFFER;
begin -- Set
   if Input_String'Length <= Maximum_String_Length then
      Temp_Result.Last                          := Input_String'Length;
      Temp_Result.Text (1 .. Temp_Result.Last) := Input_String;
   else
      Temp_Result.Last                          := Maximum_String_Length;
      Temp_Result.Text (1 .. Temp_Result.Last) :=
        Input_String (
        Input_String'First ..
        Input_String'First - 1 + Maximum_String_Length);
   end if;
   Result := Temp_Result;
end Set;

--    procedure Init_Arc_Class_Name is
--    begin
--       Set(Arc_Class_Names(NO_ARC), "NO_ARC");
--       Set(Arc_Class_Names(ACTIVATION_WAITING_ARC),
--"ACTIVATION_WAITING_ARC");
--       Set(Arc_Class_Names(ACCEPTANCE_WAITING_ARC),
--"ACCEPTANCE_WAITING_ARC");
--       Set(Arc_Class_Names(COMPLETION_WAITING_ARC),
--"COMPLETION_WAITING_ARC");
--       Set(Arc_Class_Names(ENTRY_CALLING_ARC), "ENTRY_CALLING_ARC");
--       Set(Arc_Class_Names(FINALIZATION_WAITING_ARC),
--    "FINALIZATION_WAITING_ARC");
--       Set(Arc_Class_Names(PROTECTION_WAITING_ARC),
--"PROTECTION_WAITING_ARC");
--       Set(Arc_Class_Names(PROTECTED_ENTRY_CALLING_ARC),
--    "PROTECTED_ENTRY_CALLING_ARC");
--    end Init_Arc_Class_Name;

--    procedure Init_Unit_Class_Name is
--    begin
--       Set(Unit_Class_Names(N_U_L_L), "null");
--       Set(Unit_Class_Names(RESERVED), "reserved master");
--       Set(Unit_Class_Names(COMPLETED), "already completed master");
--       Set(Unit_Class_Names(MAIN_PROCEDURE), "main procedure");
--       Set(Unit_Class_Names(TASKS), "task");
--       Set(Unit_Class_Names(BLOCKS), "block");
--       Set(Unit_Class_Names(SUBPROGRAMS), "subprogram");
--       Set(Unit_Class_Names(PROTECTED_SUBPROGRAMS), "protected
--subprogram");
--       Set(Unit_Class_Names(PROTECTED_OBJECTS), "protected object(or its
--entry body)");
--       Set(Unit_Class_Names(PROTECTED_ENTRY_BODY), "protected entry body
--of");
--       Set(Unit_Class_Names(LIBRARY_PACKAGES), "library package");
--    end Init_Unit_Class_Name;

--    function Search_Client_ID(C_Name : STRING_BUFFER) return Natural is
--    begin
--       for I in 1..Client_Count loop
--   if C_Name = Client_Names(I) then
--      return I;
--   end if;
--       end loop;
--       return 0;
--    end Search_Client_ID;

--    function Search_Server_ID(T_Name : STRING_BUFFER) return Natural is
--    begin
--       for I in 1..Server_Task_Count loop
--   if T_Name = Server_Task_Names(I) then
```

```
   --      return I;
   --   end if;
   --       end loop;
   --       return 0;
   --    end Search_Server_ID;

   --    function Search_Entry_ID(Server_ID : Natural; Entry_Name :
   --STRING_BUFFER)
   --      return ID_OF_ENTRY is
   --    begin
   --       for I in  1..Server_Entry_Count(Server_ID) loop
   --   if Server_Entry_Names(Server_ID, I) = Entry_Name then
   --      return I;
   --   end if;
   --       end loop;
   --       return 0;
   --    end Search_Entry_ID;

end Global_Types;
```

C.2.4 Pid Package

```
with V_Strings;
use V_Strings;

package Pid is

   Max_Protected_ID : constant Natural := 100;

   subtype Protected_ID is Natural range 0..Max_Protected_ID;

   function New_ID(Name : String) return Protected_ID;
   function Protected_Name(ID : Protected_Id) return String;

private

   Last_ID : Protected_ID := 0;
   type V_String_Link is access V_String;
   Names : array(Protected_ID) of V_String_Link;

end Pid;



with V_Strings;
use V_Strings;

package body Pid is

   function New_ID(Name : String) return Protected_ID is
   begin
      Last_ID := Last_ID + 1;
      Names(Last_ID) := new V_String;
      Names(Last_ID).all := To_V(Name);
      return Last_ID;
   end New_ID;

   function Protected_Name(ID : Protected_Id) return String is
   begin
      if Names(ID) /= null then
 return To_S(Names(ID).all);
      end if;
      return "";
   end Protected_Name;

end Pid;
```

C.2.5 Task_Indexed_List Package

```
with Ada.Task_Identification; use Ada.Task_Identification;

generic
type ITEM is private;
Null_ITEM : ITEM;

package Task_Indexed_List is
   type NODE;
   type A_NODE is access NODE;

   type NODE is
      record
 Index : Task_ID := Null_Task_ID;
 Object : ITEM;
 Next : A_NODE := null;
      end record;

   Null_Node : NODE := (Null_Task_ID, Null_ITEM, null);

   function SEARCH_NODE(TOP : A_NODE; CHILD : Task_ID) return ITEM;
   procedure SET_NODE(TOP : in out A_NODE ; CHILD : in Task_ID ;
    PARENT : in ITEM);
   function GET_NEXT_NODE(TOP : NODE) return NODE;

   procedure DELETE_NODE(TOP : in out A_NODE; CHILD : in Task_ID);
   function IsNULL(TOP : A_NODE) return Boolean;
end Task_Indexed_List;



with Ada.Text_IO;             use Ada.Text_IO;

package body Task_Indexed_List is
```

```
--   procedure FREE_NODE is new UNCHECKED_DEALLOCATION(NODE, A_NODE);

   function SEARCH_NODE(TOP : A_NODE; CHILD : Task_ID) return ITEM is
       function SEARCH_ITER(CURRENT_NODE : A_NODE) return ITEM is
       begin
--  New_Line;
--  New_Line;
--  Put_Line(Image(CHILD));
--  New_Line;
  if CURRENT_NODE = null then
--     Put_Line("null!");
     return Null_ITEM;
  end if;
  if CURRENT_NODE.Index = CHILD then
--     Put_Line("GET!");
--     New_Line;
     return CURRENT_NODE.Object;
  end if;
  return SEARCH_ITER(CURRENT_NODE.Next);
       end SEARCH_ITER;

    begin
       return SEARCH_ITER(TOP);
    end SEARCH_NODE;


   function GET_NEXT_NODE(TOP : NODE) return NODE is
   begin
      if TOP.Next = null then
 return Null_NODE;
      end if;
      return TOP.Next.all;
   end GET_NEXT_NODE;

   procedure SET_NODE(TOP : in out A_NODE ; CHILD : in Task_ID ; PARENT : in ITEM) is
       procedure SET_ITER(CURRENT_NODE : in out A_NODE) is
       begin
  if CURRENT_NODE = null then
     CURRENT_NODE := new NODE;
     CURRENT_NODE.Index := CHILD;
     CURRENT_NODE.Object := PARENT;
     CURRENT_NODE.Next := null;
  elsif CURRENT_NODE.Index = CHILD then
     CURRENT_NODE.Object := PARENT;
  else
     SET_ITER(CURRENT_NODE.Next);
  end if;
       end SET_ITER;
    begin
       SET_ITER(TOP);
    end SET_NODE;

   procedure DELETE_NODE(TOP : in out A_NODE ; CHILD : in Task_ID) is
      procedure DELETE_ITER(CURRENT_NODE : in out A_NODE) is
      begin
 if CURRENT_NODE.Index = CHILD then
--    FREE_NODE(CURRENT_NODE);
    CURRENT_NODE := CURRENT_NODE.Next;
 elsif CURRENT_NODE /= null then
    DELETE_ITER(CURRENT_NODE.Next);
 end if;
      end DELETE_ITER;
   begin
      DELETE_ITER(TOP);
   end DELETE_NODE;

   function IsNULL(TOP : A_NODE) return Boolean is
   begin
     return (TOP = null);
   end IsNull;

end Task_Indexed_List;
```

C.2.6 Task_Information_Collector Package

```
with Ada.Task_Identification;
use  Ada.Task_Identification;
with Global_Types;            use Global_Types;
with Pid;                     use Pid;

package Task_Information_Collector is

   protected type TIC is

      procedure BLOCK_ELABORATION_START
        (This_Unit_Class : UNIT_CLASS;
         Block_Name      : String);
      procedure BLOCK_ELABORATION_COMPLETION;
      procedure LIBRARY_PACKAGE_ELABORATION_COMPLETION;
      procedure BLOCK_EXECUTION_START;
      procedure BLOCK_EXECUTION_COMPLETION;

      --       function GET_PREVIOUS_MASTER_ID return ID_OF_MASTER;
      --       function GET_MASTER_ID(U_Class : UNIT_CLASS; U_Name :
      --STRING_BUFFER)
      --       return ID_OF_MASTER;
      --       function GET_MASTER_ID(U_Class : UNIT_CLASS; U_Name :
      --STRING_BUFFER;
      --     ID : Task_ID;
      --     Parent_ID : ID_OF_MASTER)
      --     return ID_OF_MASTER;

      procedure SIMPLE_ENTRY_CALL
        (Callee_Task : in Task_Id;
         Entry_Name  : in String);
      --   procedure CONDITIONAL_ENTRY_CALL(Object_Task : in Task_ID;
```

```
      -- Entry_Name : in STRING_BUFFER);
      procedure PROTECTED_SUBPROGRAM_CALL
        (Callee_Protected_ID : in Protected_ID;
         Subprogram_Name     : in String;
         Callee_Unit_Class   : UNIT_CLASS);
      --       procedure PROTECTED_SUBPROGRAM_CALL_INTERNAL(
      --    Subprog_Name : in STRING_BUFFER);
      procedure PROTECTED_SUBPROGRAM_CALL_COMPLETION
        (Callee_Unit_Class : UNIT_CLASS);
      procedure PROTECTED_ENTRY_CALL
        (Callee_Protected_ID : in Protected_ID;
         Entry_Name          : in String);

      procedure PROTECTED_ENTRY_CALL_COMPLETION;
      procedure SUBPROGRAM_CALL_COMPLETION (Callee_Unit_Class : UNIT_CLASS);
      --      function SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION return
      --NULL_TYPE;
      procedure BLOCK_CALL_COMPLETION;
      procedure ABORT_START (Object_Tasks : in Task_ID_List);
      --       procedure PROTECTED_BLOCK_EXECUTION_START;
      --       procedure PROTECTED_BLOCK_EXECUTION_END;
      procedure TASK_ACTIVATION_START
        (Block_Name      : String;
         Parent_Task     : Task_Id;
         This_Unit_Class : UNIT_CLASS;
         Unit_ID_Of_Task : Unit_ID);
      procedure TASK_ACTIVATION_COMPLETION;
      --      procedure TASK_EXECUTION_START;
      procedure ACCEPT_START
        (Entry_Name     : String;
         Is_Open_Accept : Boolean);
      procedure RENDEZVOUS_START (Caller_Task : Task_Id);
      procedure RENDEZVOUS_END (Caller_Task : Task_Id);
      --      procedure TASK_EXECUTION_END;
      procedure ACCEPT_WITH_TERMINATE_START
        (Entry_Name     : String;
         Is_Open_Accept : Boolean);
      procedure ASYNCHRONOUS_ENTRY_CALL
        (Callee_Task : in Task_Id;
         Entry_Name  : in String);
      procedure ASYNCHRONOUS_PROTECTED_ENTRY_CALL
        (Callee_Task : in Task_Id;
         Entry_Name  : in String);
      procedure ASYNCHRONOUS_DELAY_STMT;
      procedure ASYNCHRONOUS_TRIGGER_END;
      procedure ASYNCHRONOUS_SELECT_END;
      procedure CONDITIONAL_ENTRY_CALL;
      procedure TIMED_ENTRY_CALL;
      procedure REQUEUE_EXEC
        (Object_Task : in Task_Id;
         Entry_Name  : in String);
      procedure REQUEUE_WITH_ABORT_EXEC
        (Object_Task : in Task_Id;
         Entry_Name  : in String);
      procedure PROTECTED_REQUEUE_EXEC
        (Callee_Protected_ID : Protected_ID;
         Entry_Name          : in String);

      procedure PROTECTED_REQUEUE_WITH_ABORT_EXEC
        (Callee_Protected_ID : Protected_ID;
         Entry_Name          : in String);

      procedure GET_CDT (CDT_OF_TASK : CDT; CDT_OF_TASK_TYPE : CDT_For_TT);
      --      procedure GET_UINT_ID(UNIT_ID_OF_TASK : Unit_ID);

   end TIC;

end Task_Information_Collector;



with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Calendar; use Ada.Calendar;
with Ada.Sequential_io;
with Task_Indexed_List;
with Ada.Characters.Handling;
use Ada.Characters.Handling;
with Global_Types; use Global_Types;
with Task_Wait_For_Graph_Manager;

package body Task_Information_Collector is
   TWFG : Task_Wait_For_Graph_Manager.TWFG;

   protected body TIC is

      procedure BLOCK_ELABORATION_START(This_Unit_Class : Unit_Class; Block_Name : String) is
      begin
 TWFG.BLOCK_ELABORATION_START(This_Unit_Class, Block_Name);
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in BLOCK_ELABORATION_START!");
      end BLOCK_ELABORATION_START;

      procedure BLOCK_ELABORATION_COMPLETION is
      begin
 TWFG.BLOCK_ELABORATION_COMPLETION;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in BLOCK_ELABORATION_COMPLETION!");
      end BLOCK_ELABORATION_COMPLETION;

      procedure LIBRARY_PACKAGE_ELABORATION_COMPLETION is
      begin
 TWFG.LIBRARY_PACKAGE_ELABORATION_COMPLETION;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in LIBRARY_PACKAGE_ELABORATION_COMPLETION!");
      end LIBRARY_PACKAGE_ELABORATION_COMPLETION;
```

```
      procedure BLOCK_EXECUTION_START is
      begin
 TWFG.BLOCK_EXECUTION_START;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in BLOCK_EXECUTION_START!");
      end BLOCK_EXECUTION_START;

      procedure BLOCK_EXECUTION_COMPLETION is
      begin
 TWFG.BLOCK_EXECUTION_COMPLETION;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in BLOCK_EXECUTION_COMPLETION!");
      end BLOCK_EXECUTION_COMPLETION;


      --       function GET_PREVIOUS_MASTER_ID return ID_OF_MASTER is
      --    Result : ID_OF_MASTER;
      --       begin
      --    TWFG.GET_PREVIOUS_MASTER_ID(Result);
      --    return Result;
      --    exception
      --       when TASKING_ERROR =>
      --          Put_Line("!!! tasking error is raised in GET_PREVIOUS_MASTER_ID!");
      --          return TWFG.Get_Parent_Master(TWFG.Get_Current_Master(Current_Task));
      --       end GET_PREVIOUS_MASTER_ID;

      --       function GET_MASTER_ID(U_Class : UNIT_CLASS; U_Name : STRING_BUFFER)
      --       return ID_OF_MASTER is
      --    Result : ID_OF_MASTER;
      --       begin
      --    TWFG.GET_MASTER_ID_ENTRANCE(U_Class, U_Name, Result);
      --    return Result;
      --    exception
      --       when TASKING_ERROR =>
      --          Put_Line("!!! tasking error is raised in GET_MASTER_ID!");
      --          return Result;
      --       end GET_MASTER_ID;

      --       function GET_MASTER_ID(U_Class : UNIT_CLASS; U_Name : STRING_BUFFER;
      --     ID : Task_ID;
      --     Parent_ID : ID_OF_MASTER)
      --       return ID_OF_MASTER is
      --    Result : ID_OF_MASTER;
      --       begin
      --    TWFG.GET_MASTER_ID_ENTRANCE(U_Class, U_Name, ID, Parent_ID, Result);
      --    return Result;
      --    exception
      --       when TASKING_ERROR =>
      --          Put_Line("!!! tasking error is raised in GET_MASTER_ID(task)!");
      --          return Result;
      --       end GET_MASTER_ID;

      procedure SIMPLE_ENTRY_CALL(Callee_Task : in Task_ID;
  Entry_Name : in String) is
      begin
 TWFG.SIMPLE_ENTRY_CALL(Callee_Task, Entry_Name);
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in SIMPLE_ENTRY_CALL!");
      end SIMPLE_ENTRY_CALL;

      procedure PROTECTED_SUBPROGRAM_CALL
(Callee_Protected_ID : in Protected_ID;
  Subprogram_Name : in String; Callee_Unit_Class : Unit_Class) is
      begin
 TWFG.PROTECTED_SUBPROGRAM_CALL(Callee_Protected_ID, Subprogram_Name, Callee_Unit_Class);
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in PROTECTED_SUBPROGRAM_CALL!");
      end PROTECTED_SUBPROGRAM_CALL;

      --       procedure PROTECTED_SUBPROGRAM_CALL_INTERNAL(
      --     Subprog_Name : in STRING_BUFFER) is
      --       begin
      --    TWFG.PROTECTED_SUBPROGRAM_CALL_INTERNAL(Subprog_Name);
      --    exception
      --       when TASKING_ERROR =>
      --          Put_Line("!!! tasking error is raised in PROTECTED_SUBPROGRAM_CALL_INTERNAL!");
      --       end PROTECTED_SUBPROGRAM_CALL_INTERNAL;

      procedure PROTECTED_SUBPROGRAM_CALL_COMPLETION(Callee_Unit_Class : Unit_Class) is
      begin
 TWFG.PROTECTED_SUBPROGRAM_CALL_COMPLETION(Callee_Unit_Class);
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in PROTECTED_SUBPROGRAM_CALL_COMPLETION!");
      end PROTECTED_SUBPROGRAM_CALL_COMPLETION;

      procedure PROTECTED_ENTRY_CALL_COMPLETION is
      begin
 TWFG.PROTECTED_ENTRY_CALL_COMPLETION;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in PROTECTED_ENTRY_CALL_COMPLETION!");
      end PROTECTED_ENTRY_CALL_COMPLETION;

      procedure PROTECTED_ENTRY_CALL
(Callee_Protected_ID : in Protected_ID;
  entry_Name : in String) is
      begin
 TWFG.PROTECTED_ENTRY_CALL(Callee_Protected_ID, Entry_Name);
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in PROTECTED_ENTRY_CALL!");
      end PROTECTED_ENTRY_CALL;
```

```
      procedure SUBPROGRAM_CALL_COMPLETION(Callee_Unit_Class : Unit_Class) is
      begin
TWFG.SUBPROGRAM_CALL_COMPLETION(Callee_Unit_Class);
      exception
when TASKING_ERROR =>
   Put_Line("!!! tasking error is raised in SUBPROGRAM_CALL_COMPLETION!");
      end SUBPROGRAM_CALL_COMPLETION;

      --       function SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION return NULL_TYPE is
      --       begin
      --   TWFG.SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION;
      --   return 0;
      --   exception
      --      when TASKING_ERROR =>
      --         Put_Line("!!! tasking error is raised in SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION!");
      --         return 0;
      --       end SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION;

      procedure BLOCK_CALL_COMPLETION is
      begin
TWFG.BLOCK_CALL_COMPLETION;
      exception
when TASKING_ERROR =>
   Put_Line("!!! tasking error is raised in BLOCK_CALL_COMPLETION!");
      end BLOCK_CALL_COMPLETION;

      procedure ABORT_START(Object_Tasks : in Task_ID_List) is
      begin
TWFG.ABORT_START(Object_Tasks);
      exception
when TASKING_ERROR =>
   Put_Line("!!! tasking error is raised in Abort_Start!");

      end ABORT_START;

      --       procedure PROTECTED_BLOCK_EXECUTION_START is
      --       begin
      --   TWFG.PROTECTED_BLOCK_EXECUTION_START;
      --   exception
      --      when TASKING_ERROR =>
      --         Put_Line("!!! tasking error is raised in PROTECTED_BLOCK_EXECUTION_START!");
      --       end PROTECTED_BLOCK_EXECUTION_START;

      --       procedure PROTECTED_BLOCK_EXECUTION_END is
      --       begin
      --   TWFG.PROTECTED_BLOCK_EXECUTION_END;
      --   exception
      --      when TASKING_ERROR =>
      --         Put_Line("!!! tasking error is raised in PROTECTED_BLOCK_EXECUTION_END!");
      --       end PROTECTED_BLOCK_EXECUTION_END;

      procedure TASK_ACTIVATION_START
(Block_Name : String;
  Parent_Task : Task_ID;
  This_Unit_Class : Unit_Class; Unit_ID_Of_Task : Unit_ID) is
      begin
if Debug_Mode then
   Put_Line("attempt to entry call TASK_ACTIVATION_START...");
end if;
TWFG.TASK_ACTIVATION_START(Block_Name, Parent_Task,
    This_Unit_Class, Unit_ID_Of_Task);
if Debug_Mode then
   Put_Line("TASK_ACTIVATION_START is over...");
end if;
      exception
when TASKING_ERROR =>
   Put_Line("!!! tasking error is raised in TASK_ACTIVATION_START!");
      end TASK_ACTIVATION_START;

      procedure TASK_ACTIVATION_COMPLETION is
      begin
TWFG.TASK_ACTIVATION_COMPLETION;
      exception
when TASKING_ERROR =>
   Put_Line("!!! tasking error is raised in TASK_ACTIVATION_COMPLETION!");
      end TASK_ACTIVATION_COMPLETION;


      --       procedure TASK_EXECUTION_START is
      --       begin
      --   TWFG.TASK_EXECUTION_START;
      --   exception
      --      when TASKING_ERROR =>
      --         Put_Line("!!! tasking error is raised in TASK_EXECUTION_START!");
      --       end TASK_EXECUTION_START;

      procedure ACCEPT_START(Entry_Name : String; Is_Open_Accept : Boolean) is
      begin
TWFG.ACCEPT_START(Entry_Name, Is_Open_Accept);
      exception
when TASKING_ERROR =>
   Put_Line("!!! tasking error is raised in ACCEPT_START!");
      end ACCEPT_START;

      procedure RENDEZVOUS_START(Caller_Task : Task_ID) is
      begin
TWFG.RENDEZVOUS_START(Caller_Task);
      exception
when TASKING_ERROR =>
   Put_Line("!!! tasking error is raised in RENDEZVOUS_START!");
      end RENDEZVOUS_START;

      procedure RENDEZVOUS_END(Caller_Task : Task_ID) is
      begin
TWFG.RENDEZVOUS_END(Caller_Task);
      exception
when TASKING_ERROR =>
   Put_Line("!!! tasking error is raised in RENDEZVOUS_END!");
      end RENDEZVOUS_END;
```

```
      --       procedure TASK_EXECUTION_END is
      --       begin
      --    TWFG.TASK_EXECUTION_END;
      --    exception
      --       when TASKING_ERROR =>
      --          Put_Line("!!! tasking error is raised in TASK_EXECUTION_END!");

      --       end TASK_EXECUTION_END;

      procedure ACCEPT_WITH_TERMINATE_START
(Entry_Name : String; Is_Open_Accept : Boolean) is
      begin
 TWFG.ACCEPT_WITH_TERMINATE_START
   (Entry_Name, Is_Open_Accept);
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in ACCEPT_WITH_TERMINATE_START!");
      end ACCEPT_WITH_TERMINATE_START;

      procedure ASYNCHRONOUS_ENTRY_CALL(Callee_Task : in Task_ID;
Entry_Name : in String) is
      begin
 TWFG.ASYNCHRONOUS_ENTRY_CALL(Callee_Task, Entry_Name);
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in ASYNCHRONOUS_ENTRY_CALL!");
      end ASYNCHRONOUS_ENTRY_CALL;

      procedure ASYNCHRONOUS_PROTECTED_ENTRY_CALL(Callee_Task : in Task_ID;
  Entry_Name : in String) is
      begin
 TWFG.ASYNCHRONOUS_PROTECTED_ENTRY_CALL(Callee_Task, Entry_Name);
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in ASYNCHRONOUS_PROTECTED_ENTRY_CALL!");
      end ASYNCHRONOUS_PROTECTED_ENTRY_CALL;

      procedure ASYNCHRONOUS_DELAY_STMT is
      begin
 TWFG.ASYNCHRONOUS_DELAY_STMT;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in ASYNCHRONOUS_DELAY_STMT!");
      end ASYNCHRONOUS_DELAY_STMT;

      procedure ASYNCHRONOUS_TRIGGER_END is
      begin
 TWFG.ASYNCHRONOUS_TRIGGER_END;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in ASYNCHRONOUS_TRIGGER_END!");
      end ASYNCHRONOUS_TRIGGER_END;



      procedure ASYNCHRONOUS_SELECT_END is
      begin
 TWFG.ASYNCHRONOUS_SELECT_END;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in ASYNCHRONOUS_ASYNCHRONOUS_SELECT_END!");
      end ASYNCHRONOUS_SELECT_END;

      procedure CONDITIONAL_ENTRY_CALL is
      begin
 TWFG.CONDITIONAL_ENTRY_CALL;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in CONDITIONAL_ENTRY_CALL!");
      end CONDITIONAL_ENTRY_CALL;

      procedure TIMED_ENTRY_CALL is
      begin
 TWFG.TIMED_ENTRY_CALL;
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in TIMED_ENTRY_CALL!");
      end TIMED_ENTRY_CALL;

      procedure REQUEUE_EXEC(Object_Task : in Task_ID;
     Entry_Name : in String) is
      begin
 TWFG.REQUEUE_EXEC(Object_Task, Entry_Name);
      end REQUEUE_EXEC;

      procedure REQUEUE_WITH_ABORT_EXEC(Object_Task : in Task_ID;
Entry_Name : in String) is
      begin
 TWFG.REQUEUE_WITH_ABORT_EXEC(Object_Task, Entry_Name);
      end REQUEUE_WITH_ABORT_EXEC;

      procedure PROTECTED_REQUEUE_EXEC(Callee_Protected_ID : Protected_ID;
       Entry_Name : in String) is
      begin
 TWFG.PROTECTED_REQUEUE_EXEC(Callee_Protected_ID , Entry_Name);
      end PROTECTED_REQUEUE_EXEC;

      procedure PROTECTED_REQUEUE_WITH_ABORT_EXEC
(Callee_Protected_ID : Protected_ID;
  Entry_Name : in String) is
      begin
 TWFG.PROTECTED_REQUEUE_WITH_ABORT_EXEC(Callee_Protected_ID ,
 Entry_Name);
      end PROTECTED_REQUEUE_WITH_ABORT_EXEC;

--       procedure ENQUEUE_CALL_START(Caller_Task : Task_ID;
--     Queue_Name: in String) is
--       begin
```

```
--    TWFG.ENQUEUE_CALL_START(Caller_Task, Queue_Name);
--        end ENQUEUE_CALL_START;

      procedure GET_CDT(CDT_OF_TASK : CDT; CDT_OF_TASK_TYPE : CDT_For_TT) is
      begin
 TWFG.GET_CDT(CDT_OF_TASK, CDT_OF_TASK_TYPE);
      exception
 when TASKING_ERROR =>
    Put_Line("!!! tasking error is raised in GET_CDT!");
      end GET_CDT;
   end TIC;

end Task_Information_Collector;
```

C.2.7 Task_Wait_For_Graph_Manager Package

```
with Ada.Task_Identification; use  Ada.Task_Identification;
with Global_Types;            use Global_Types;
with Pid;                     use Pid;
--with TWFGM_Facilities;        use TWFGM_Facilities;
package Task_Wait_For_Graph_Manager is

   protected type TWFG is

      procedure BLOCK_ELABORATION_COMPLETION;
      procedure LIBRARY_PACKAGE_ELABORATION_COMPLETION;
      procedure BLOCK_ELABORATION_START
        (This_Unit_Class : UNIT_CLASS;
         Block_Name      : String);
      procedure BLOCK_EXECUTION_START;
      procedure BLOCK_EXECUTION_COMPLETION;
      --      procedure GET_PREVIOUS_MASTER_ID(Result : out ID_OF_MASTER);
      --       procedure GET_MASTER_ID_ENTRANCE(U_Class : in UNIT_CLASS;
      --         U_Name : in STRING_BUFFER;
      --         Ret_ID : out ID_OF_MASTER);
      --       procedure GET_MASTER_ID_ENTRANCE(U_Class : in UNIT_CLASS;
      --         U_Name : in STRING_BUFFER;
      --         ID : in Task_ID;
      --         Parent_ID : in ID_OF_MASTER;
      --         Ret_ID : out ID_OF_MASTER);
      procedure SIMPLE_ENTRY_CALL
        (Callee_Task : in Task_Id;
         Entry_Name  : in String);
      --   procedure CONDITIONAL_ENTRY_CALL(Object_Task : in Task_ID;
      -- Entry_Name : in STRING_BUFFER);
      procedure PROTECTED_SUBPROGRAM_CALL
        (Callee_Protected_ID : in Protected_ID;
         Subprogram_Name     : in String;
         Callee_Unit_Class   : UNIT_CLASS);
      --       procedure PROTECTED_SUBPROGRAM_CALL_INTERNAL(
      --    Subprog_Name : in STRING_BUFFER);
      procedure PROTECTED_SUBPROGRAM_CALL_COMPLETION
        (Callee_Unit_Class : UNIT_CLASS);
      procedure PROTECTED_ENTRY_CALL
        (Callee_Protected_ID : in Protected_ID;
         Entry_Name          : in String);

      procedure PROTECTED_ENTRY_CALL_COMPLETION;
      procedure SUBPROGRAM_CALL_COMPLETION (Callee_Unit_Class : UNIT_CLASS);
      --      procedure SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION;
      procedure BLOCK_CALL_COMPLETION;
      procedure ABORT_START (Object_Tasks : in Task_ID_List);
      --       procedure PROTECTED_BLOCK_EXECUTION_START;
      --       procedure PROTECTED_BLOCK_EXECUTION_END;
      procedure TASK_ACTIVATION_START
        (Block_Name      : String;
         Parent_Task     : Task_Id;
         This_Unit_Class : UNIT_CLASS;
         Unit_ID_Of_Task : Unit_ID);
      procedure TASK_ACTIVATION_COMPLETION;
      --      procedure TASK_EXECUTION_START;
      procedure ACCEPT_START
        (Entry_Name     : String;
         Is_Open_Accept : Boolean);
      procedure RENDEZVOUS_START (Caller_Task : Task_Id);
      procedure RENDEZVOUS_END (Caller_Task : Task_Id);
      --      procedure TASK_EXECUTION_END;
      procedure ACCEPT_WITH_TERMINATE_START
        (Entry_Name     : String;
         Is_Open_Accept : Boolean);
      procedure ASYNCHRONOUS_ENTRY_CALL
        (Callee_Task : in Task_Id;
         Entry_Name  : in String);
      procedure ASYNCHRONOUS_PROTECTED_ENTRY_CALL
        (Callee_Task : in Task_Id;
         Entry_Name  : in String);

      procedure ASYNCHRONOUS_DELAY_STMT;
      procedure ASYNCHRONOUS_TRIGGER_END;
      procedure ASYNCHRONOUS_SELECT_END;
      procedure CONDITIONAL_ENTRY_CALL;
      procedure TIMED_ENTRY_CALL;
      procedure REQUEUE_EXEC
        (Object_Task : in Task_Id;
         Entry_Name  : in String);
      procedure REQUEUE_WITH_ABORT_EXEC
        (Object_Task : in Task_Id;
         Entry_Name  : in String);
      procedure PROTECTED_REQUEUE_EXEC
        (Callee_Protected_ID : Protected_ID;
         Entry_Name          : in String);
      procedure PROTECTED_REQUEUE_WITH_ABORT_EXEC
        (Callee_Protected_ID : Protected_ID;
         Entry_Name          : in String);
      procedure Get_CDT (CDT_OF_TASK : CDT; CDT_OF_TASK_TYPE : CDT_For_TT);
```

```
procedure Get_Master_Index
  (U_Class       : in UNIT_CLASS;
   U_Name        : in STRING_BUFFER;
   Parent_Master : in Master_Index_Link;
   New_Master    : out Master_Index_Link);
procedure Set_Current_Master (Master : in Master_Index_Link);
procedure Set_Current_Master
  (Master : in Master_Index_Link;
   ID     : in Task_Id);
procedure Set_Original_Master (Master : in Master_Index_Link);
--      function Get_Current_Master return ID_OF_MASTER;
function Get_Current_Master
  (ID   : Task_Id := Current_Task)
   return Master_Index_Link;

function Get_Original_Master return Master_Index_Link;
function Get_Original_Master (ID : Task_Id) return Master_Index_Link;
procedure Set_Original_Master
  (Master : in Master_Index_Link;
   ID     : Task_Id);
--       function Get_Parent_Master return Master_Index_Link;
--       function Get_Parent_Master(Master : Master_Index_Link) return
--ID_OF_MASTER;
--       procedure Set_Running_Task(Master_ID : in ID_OF_MASTER;
--   Running_Task_ID : in Task_ID);
--       function Get_Running_Task(Master_ID : ID_OF_MASTER) return
--Task_ID;
procedure Set_Arc
  (Source, Destination : in Master_Index_Link;
   Arc                 : in ARC_CLASS);
procedure Remove_All_Out_Arc (Master : in Master_Index_Link);
procedure Remove_All_Out_Arc
  (Master : in Master_Index_Link;
   Arc    : in ARC_CLASS);
function Search_Protected_Object
  (U_Name : STRING_BUFFER)
   return   Master_Index_Link;
function Get_An_Out_Arc_Destination
  (Source : Master_Index_Link)
   return   Master_Index_Link;
procedure Set_Master_Name
  (Master : in Master_Index_Link;
   U_Name : in STRING_BUFFER);
function Get_Master_Name
  (Master : in Master_Index_Link)
   return   STRING_BUFFER;
procedure Set_Task_Status
  (ID     : in Task_Id;
   Status : in Status_Of_Task);
function Get_Task_Status (ID : Task_Id) return Status_Of_Task;
--       procedure Set_Server_ID(ID : in Task_ID; T_Name : in
--STRING_BUFFER);
--       function Get_Server_ID(ID : Task_ID) return Natural;

procedure Set_Calling_Entry
  (Object_Task : in Task_Id;
   E_Name      : in STRING_BUFFER);
procedure Set_Calling_Entry
  (E_Name : in STRING_BUFFER;
   Master : in Master_Index_Link);
function Get_Calling_Entry
  (Master : Master_Index_Link)
   return   STRING_BUFFER;
function Get_Arc_Class
  (Source, Destination : Master_Index_Link)
   return                ARC_CLASS;
function Get_Master_Class
  (Master : Master_Index_Link)
   return   UNIT_CLASS;
procedure Remove_Node (Master : in Master_Index_Link);
procedure Remove_All_Out_Node
  (Master  : in Master_Index_Link;
   U_Class : in UNIT_CLASS);
procedure Set_All_Acceptance_Arc (Entry_Name : in STRING_BUFFER);
procedure Set_All_Acceptance_Arc (Master : Master_Index_Link);
procedure Set_All_Finalization_Arc (Master : Master_Index_Link);
procedure Set_Arc_Label
  (Source, Destination : in Master_Index_Link;
   Arc_Label           : in STRING_BUFFER);
procedure Remove_Task;
procedure Set_Status (Master_Status : in Status_Of_Master);
procedure Set_Status
  (Master_Status : in Status_Of_Master;
   Master        : in Master_Index_Link);
function Get_Status
  (Master : Master_Index_Link)
   return   Status_Of_Master;
procedure Set_Completion_Arc;
procedure Set_Reverse_Completion_Arc;
procedure Dump_Master_List;
procedure Set_Entry_Caller (Master : in Master_Index_Link);
function Get_Entry_Caller (ID : Task_Id) return Master_Index_Link;
procedure EXIST_A_PATH
  (START    : in Master_Index_Link;
   TERMINAL : in Master_Index_Link;
   Result   : out Boolean);
procedure FIND_A_PATH
  (START    : in Master_Index_Link;
   TERMINAL : in Master_Index_Link;
   PATH     : out Node_List_Link);
procedure SATISFY_CONDITION_OF_DEADLOCK
  (PATH         : in Node_List_Link;
   Result       : out Boolean;
   Async_Result : out Boolean;
   Async_Depend : out Node_List_Link);

procedure CHECK_CIRCULAR_DEADLOCKS
  (TAIL_MASTER : in Master_Index_Link;
   HEAD_MASTER : in Master_Index_Link);
```

```
      procedure Dump_Task_Status;
      function Get_Communicate_Task
        (Master : Master_Index_Link)
         return   Master_Index_Link;
      function Get_Arc_Label
        (Source, Destination : Master_Index_Link)
         return               STRING_BUFFER;
      procedure Abort_Node (Master : in Master_Index_Link);
      procedure Put_Master_Class (Master : in Master_Index_Link);
      procedure Set_Completed_Master (Master : in Master_Index_Link);
      procedure Set_Completed_Master
        (Master : in Master_Index_Link;
         ID     : Task_Id);
      function Get_Completed_Master (ID : Task_Id) return Master_Index_Link;
      procedure Set_Asynchronous_State
        (Master  : in Master_Index_Link;
         State   : in Async_Type;
         Trigger : in Master_Index_Link);
      function Get_Asynchronous_State
        (Master : Master_Index_Link)
         return   Async_Type;
      function Get_Asynchronous_Trigger
        (Master : Master_Index_Link)
         return   Master_Index_Link;
      function Is_Entry_Call
        (Master     : Master_Index_Link;
         Entry_Name : STRING_BUFFER)
         return       Boolean;

      procedure Set_Source_Unit_ID (ID : Unit_ID);
      procedure Remove_Arc (Source, Destination : Master_Index_Link);

      procedure Set_Standby_Arc
        (Callee_Task : in Task_Id;
         Source      : in Master_Index_Link;
         Arc         : in ARC_CLASS;
         Label       : in STRING_BUFFER);
      function Get_Standby_Arc
        (ID   : Task_Id := Current_Task)
         return Node_List_Link;

   private
      -- List of existing Task_ID with Master_ID which has the Task thread.
      -- List of existing Task_ID with Master_ID which is original task Master.
      -- List of entry called by a task.
      All_Task_ID : Task_Index_List.A_NODE;

      -- List of existing Master_ID with some information
      --      Master_ID_List : Array_Of_Master_Index;
      Master_List     : Node_List_Link := null;
      Now_Master_List : Node_List_Link := null;

      -- adjacency matrix for task-wait-for graph
      --       AM : Array_Of_ARC_CLASS
      --  := (others => (others => NO_ARC));
      -- path matrix for task-wait-for graph
      --       PM : Array_Of_TAG
      --:= (others => (others => 0));
      -- arc label
      --       AL : Array_Of_Label
      --  := (others => (others => Null_String_Buffer));

      --      Async_state : Array_Of_Async_State;

      Set_Count : Integer := 0;

      -- protected ID と master ID の対応
      Pid_To_Mid : Array_Of_Protected_Id := (others => null);

   end TWFG;

end Task_Wait_For_Graph_Manager;



with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Text_IO;             use Ada.Text_IO;
with Ada.Calendar;            use Ada.Calendar;
with Ada.Characters.Handling; use Ada.Characters.Handling;
with Ada.Sequential_IO;
with Task_Indexed_List;
with Global_Types;            use Global_Types;
-- with TWFGM_Facilities;        use TWFGM_Facilities;

package body Task_Wait_For_Graph_Manager is
   Start_Time : Time;

   protected body TWFG is


      procedure BLOCK_ELABORATION_START
        (This_Unit_Class : Unit_Class; Block_Name : String) is
         Result, Protected_Master : Master_Index_Link;
      begin
         if Debug_Mode then
            Put_Line("BLOCK_ELABORATION_START : " & Image(Current_Task));
            Put_Line("Block name : " & Block_Name);
         end if;
         case This_Unit_Class is
         when MAIN_PROCEDURE =>
            TWFG.Get_Master_Index(This_Unit_Class, Get_String_Buffer(Block_Name), null, Result);
            TWFG.Set_Current_Master(Result);
            TWFG.Set_Original_Master(Result);
            TWFG.Set_Status(BLOCK_ELABORATION_START);
            TWFG.Set_Task_Status(Current_Task, BLOCK_ELABORATING);
         when BLOCKS|PROCEDURES|FUNCTIONS =>
            declare
               CM : Master_Index_Link := TWFG.Get_Current_Master(Current_Task);
```

```
      begin
         -- サブプログラムからサブプログラムへの呼び出し（含む再帰呼び出し）
         -- に関しては、新しいマスタは生成しない
         if Debug_Mode then
            Put_Line("current master class:" & Unit_Class'IMAGE(CM.Master_Class));
         end if;
         if CM.Master_Class in PROCEDURES..FUNCTIONS then
            CM.Master_Name
              := Get_String_Buffer(Block_Name);
            CM.Recursive_Count :=
              CM.Recursive_Count + 1;
            if Debug_Mode then
               Put("recursive master name:");
               Put_String(CM.Master_Name);
               New_Line;
            end if;
         else
            TWFG.Get_Master_Index(This_Unit_Class,
                                  Get_String_Buffer(Block_Name),
                                  CM, Result);
            TWFG.Set_Asynchronous_State(Result, TWFG.Get_Asynchronous_State(Result.Parent_Master), TWFG.Get_Asynchronous_Trigger(Result.Parent_Master));
            TWFG.Set_Current_Master(Result);
            TWFG.Set_Arc(Result.Parent_Master, Result, FINALIZATION_WAITING_ARC);
         end if;
      end;
      TWFG.Set_Status(BLOCK_ELABORATION_START);
      TWFG.Set_Task_Status(Current_Task, BLOCK_ELABORATING);
      --       Put_Line("No.1");
      --       TWFG.Set_All_Acceptance_Arc(Result);
      --       TWFG.Dump_Master_List;
   when LIBRARY_PACKAGES =>
      TWFG.Get_Master_Index(This_Unit_Class,
                            Get_String_Buffer(Block_Name),
                            null, Result);
      TWFG.Set_Original_Master(Result, Current_Task);
      TWFG.Set_Current_Master(Result, Current_Task);
      TWFG.Set_Task_Status(Current_Task, LIBRARY_PACKAGE_ELABORATING);
   when TASKS =>
      Result := TWFG.Get_Current_Master(Current_Task);
      -- Task Type に対応するための対処が必要
      --       TWFG.Set_Server_ID(Current_Task, U_Name);
      --       TWFG.Set_Master_Name(Result, U_Name);
      TWFG.Set_All_Acceptance_Arc(Result);
   when PROTECTED_ENTRY_BODIES =>
      Result := TWFG.Get_Current_Master(Current_Task);
      Result.Parent_Master :=
        Get_Entry_Caller(Current_Task);
      if TWFG.Get_Arc_Class(TWFG.Get_Entry_Caller(Current_Task),
                            TWFG.Get_Current_Master(Current_Task))
        /= PROTECTED_ENTRY_CALLING_ARC then
         Put_Line("prca1");
         TWFG.Set_Arc(TWFG.Get_Entry_Caller(Current_Task),
                      TWFG.Get_Current_Master(Current_Task) ,
                      PROTECTED_ENTRY_CALLING_ARC);
         TWFG.Set_Arc_Label(TWFG.Get_Entry_Caller(Current_Task),
                            TWFG.Get_Current_Master(Current_Task),
                            Get_String_Buffer(Block_Name));
      end if;

      TWFG.Set_Asynchronous_State
        (Result,
         TWFG.Get_Asynchronous_State(Result.Parent_Master),
         TWFG.Get_Asynchronous_Trigger
           (Result.Parent_Master));
      Result.Running_Task_ID := Current_Task;
      TWFG.Set_Status(PROTECTED_ENTRY_BLOCK_ELABORATION_START);
      TWFG.Set_Task_Status(Current_Task,
                           PROTECTED_ENTRY_BLOCK_ELABORATING);
   when PROTECTED_PROCEDURES | PROTECTED_FUNCTIONS =>
      Result := TWFG.Get_Current_Master(Current_Task);
      TWFG.Set_Asynchronous_State(Result, TWFG.Get_Asynchronous_State(Result.Parent_Master), TWFG.Get_Asynchronous_Trigger(Result.Parent_Master));
      TWFG.Set_Status(PROTECTED_SUBPROGRAM_ELABORATION_START);
      TWFG.Set_Task_Status(Current_Task, PROTECTED_SUBPROGRAM_ELABORATING);
      --       Put_Line("No.2");
      Protected_Master := TWFG.Get_An_Out_Arc_Destination(Result);
      if Protected_Master /= null then
         TWFG.Set_Arc(Protected_Master, Result,
                      FINALIZATION_WAITING_ARC);
      end if;
      TWFG.Get_An_Out_Arc_Destination(Result).Running_Task_ID :=
        Current_Task;
      TWFG.Remove_All_Out_Arc(Result);
      --       TWFG.Set_All_Acceptance_Arc(Result);
      --    when PROTECTED_OBJECTS =>
      --         Result := TWFG.Search_Protected_Object(This_Unit_Name);
      --         if Result = Null_Master_ID then
      --    TWFG.Get_Master_ID(This_Unit_Class, Block_Name, Null_Master_ID, Result);
      --    TWFG.Set_Status(PROTECTED_OBJECT_EXIST, Result);
      --         end if;
      --       TWFG.Set_All_Acceptance_Arc(Result);
   when others => null;
   end case;
   -- Put("TIC.Get_Master_ID で得た ID -");
   -- Put(Integer(Result));
   -- New_Line;
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in GET_MASTER_ID_ENTRANCE!");
end BLOCK_ELABORATION_START;

procedure BLOCK_ELABORATION_COMPLETION is
begin
   -- Put_Line("BLOCK_ELABORATION_COMPLETION - " & Image(Current_Task));
   TWFG.Set_Status(BLOCK_ELABORATION_COMPLETION);
   TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in BLOCK_ELABORATION_COMPLETION!");
end BLOCK_ELABORATION_COMPLETION;
```

```
procedure LIBRARY_PACKAGE_ELABORATION_COMPLETION is
begin
   -- Put_Line("BLOCK_ELABORATION_COMPLETION - " & Image(Current_Task));
   TWFG.Set_Status(LIBRARY_PACKAGE_ELABORATION_COMPLETION);
   TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in LIBRARY_PACKAGE_ELABORATION_COMPLETION!");
end LIBRARY_PACKAGE_ELABORATION_COMPLETION;


procedure BLOCK_EXECUTION_START is
begin
   TWFG.Set_Status(BLOCK_EXECUTION_START);
   if Debug_Mode then
      Put_Line("BLOCK_EXECUTION_START(" & Unit_Class'IMAGE(TWFG.Get_Master_Class(TWFG.Get_Current_Master(Current_Task))) & " - " & Image(Current_Task));
   end if;

   case TWFG.Get_Master_Class(TWFG.Get_Current_Master(Current_Task)) is
   when PROCEDURES | FUNCTIONS | BLOCKS | MAIN_PROCEDURE =>
      TWFG.Set_Task_Status(Current_Task, BLOCK_EXECUTION_WAITING);
   when PROTECTED_PROCEDURES | PROTECTED_FUNCTIONS | PROTECTED_ENTRY_BODIES =>
      TWFG.Set_Task_Status(Current_Task,
                           PROTECTED_SUBPROGRAM_EXECUTION_WAITING);
   when TASKS =>
      TWFG.Set_Status(ACTIVATION_COMPLETION);
      TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
   when others =>
      null;
   end case;
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in BLOCK_EXECUTION_START!");
end BLOCK_EXECUTION_START;

--       procedure GET_PREVIOUS_MASTER_ID(Result : out ID_OF_MASTER) is
--       begin
-- -- Put_Line("GET_PREVIOUS_MASTER_ID - " & Image(Current_Task));
--   Result :=TWFG.Get_Parent_Master(TWFG.Get_Current_Master(Current_Task));
--   exception
--      when TASKING_ERROR =>
--         Put_Line("!!! tasking error is raised in GET_PREVIOUS_MASTER_ID!");
--       end GET_PREVIOUS_MASTER_ID;

procedure BLOCK_EXECUTION_COMPLETION is
begin
   -- Put_Line("BLOCK_EXECUTION_COMPLETION - " & Image(Current_Task));
   TWFG.Set_Status(BLOCK_EXECUTION_COMPLETION);
   TWFG.Set_Completed_Master(TWFG.Get_Current_Master(Current_Task));
   case TWFG.Get_Master_Class(TWFG.Get_Current_Master(Current_Task)) is
   when PROCEDURES | FUNCTIONS | BLOCKS =>
      declare
         CM : Master_Index_Link :=
           TWFG.Get_Current_Master(Current_Task);
      begin
         -- サブプログラムからサブプログラムへの呼び出し（含む再帰呼び出し）
         -- に関しては、新しいマスタは生成しない
         if CM.Recursive_Count > 0 then
            CM.Recursive_Count :=
              CM.Recursive_Count - 1;
         else
            TWFG.Set_Task_Status(Current_Task, BLOCK_COMPLETED);
            TWFG.Set_All_Finalization_Arc(TWFG.Get_Current_Master(Current_Task));
            TWFG.Set_Current_Master
              (TWFG.Get_Current_Master(Current_Task).Parent_Master);
         end if;
      end;
   when MAIN_PROCEDURE =>
      TWFG.Set_Task_Status(Current_Task, BLOCK_COMPLETED);
      TWFG.Set_All_Finalization_Arc(TWFG.Get_Current_Master(Current_Task));
   when PROTECTED_PROCEDURES | PROTECTED_FUNCTIONS | PROTECTED_ENTRY_BODIES | PROTECTED_OBJECTS =>
      if Debug_Mode then
         Put_Line
           ("BLOCK_EXECUTION_COMPLETION: Current_Master: "
            & Integer'IMAGE(TWFG.Get_Current_Master
              (Current_Task).Master_ID) &
              " -> "
            & Integer'IMAGE(TWFG.Get_Current_Master(Current_Task).
                Parent_Master.Master_ID));
      end if;
      TWFG.Set_Task_Status(Current_Task, PROTECTED_SUBPROGRAM_COMPLETED);
      TWFG.Set_All_Finalization_Arc(TWFG.Get_Current_Master(Current_Task));

      TWFG.Set_Current_Master(TWFG.Get_Current_Master(Current_Task).Parent_Master);
   when TASKS =>
      TWFG.Set_Task_Status(Current_Task, COMPLETED);
      TWFG.Remove_Node(TWFG.Get_Current_Master(Current_Task));
      TWFG.Remove_Task;
   when others =>
      null;
   end case;
   -- TWFG.Dump_Master_List;
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in BLOCK_EXECUTION_COMPLETION!");
end BLOCK_EXECUTION_COMPLETION;

--       procedure GET_MASTER_ID_ENTRANCE(U_Class : in UNIT_CLASS;
--         U_Name : in STRING_BUFFER;
--         Ret_ID : out ID_OF_MASTER) is
--   Result, Protected_ID : ID_OF_MASTER;
--       begin
-- -- Put_Line("GET_MASTER_ID - " & Image(Current_Task));
-- -- Put("Now " & Image(Current_Task) & " is hold by ");
-- -- Put_String(U_Name);

--   case U_Class is
--      when MAIN_PROCEDURE =>
```

```
--         TWFG.Get_Master_ID(U_Class, U_Name, Null_Master_ID, Result);
--         TWFG.Set_Current_Master(Result);
--         TWFG.Set_Original_Master(Result);
--         TWFG.Set_Status(BLOCK_ELABORATION_START);
--         TWFG.Set_Task_Status(Current_Task, BLOCK_ELABORATING);
--      when BLOCKS|SUBPROGRAMS =>
--         TWFG.Get_Master_ID(U_Class, U_Name,
--    TWFG.Get_Current_Master(Current_Task),
--    Result);
--         TWFG.Set_Asynchronous_State(Result, TWFG.Get_Asynchronous_State(TWFG.Get_Parent_Master(Result)), TWFG.Get_Asynchronous_Trigger(TWFG.Get_Parent_Master(Result)));
--         TWFG.Set_Current_Master(Result);
--         TWFG.Set_Status(BLOCK_ELABORATION_START);
--         TWFG.Set_Task_Status(Current_Task, BLOCK_ELABORATING);
-- --       Put_Line("No.1");
--         TWFG.Set_Arc(TWFG.Get_Parent_Master(Result), Result, FINALIZATION_WAITING_ARC);
-- --       TWFG.Set_All_Acceptance_Arc(Result);
-- --       TWFG.Dump_Master_List;
--      when LIBRARY_PACKAGES =>
--         TWFG.Get_Master_ID(U_Class, U_Name, Null_Master_ID, Result);
--         TWFG.Set_Original_Master(Result, Current_Task);
--         TWFG.Set_Current_Master(Result, Current_Task);
--         TWFG.Set_Task_Status(Current_Task, LIBRARY_PACKAGE_ELABORATING);
--      when TASKS =>
--         Result := TWFG.Get_Current_Master(Current_Task);
-- -- Task Type に対応するための対処が必要
-- --       TWFG.Set_Server_ID(Current_Task, U_Name);
-- --       TWFG.Set_Master_Name(Result, U_Name);
--         TWFG.Set_All_Acceptance_Arc(Result);
--      when PROTECTED_ENTRY_BODY =>
--         Result := TWFG.Get_Current_Master(Current_Task);
--         Master_ID_List(Result).Parent_Master_ID :=
--   Get_Entry_Caller(Current_Task);
--         if TWFG.Get_Arc_Class(TWFG.Get_Entry_Caller(Current_Task),
--       TWFG.Get_Current_Master(Current_Task))
--   /= PROTECTED_ENTRY_CALLING_ARC then
--    TWFG.Set_Arc(TWFG.Get_Entry_Caller(Current_Task),
--         TWFG.Get_Current_Master(Current_Task) ,
--         PROTECTED_ENTRY_CALLING_ARC);
--    TWFG.Set_Arc_Label(TWFG.Get_Entry_Caller(Current_Task),
--      TWFG.Get_Current_Master(Current_Task), U_Name);
--         end if;

--         TWFG.Set_Asynchronous_State(Result, TWFG.Get_Asynchronous_State(TWFG.Get_Parent_Master(Result)), TWFG.Get_Asynchronous_Trigger(TWFG.Get_Parent_Master(Result)));
--         TWFG.Set_Running_Task(Result, Current_Task);
--         TWFG.Set_Status(PROTECTED_ENTRY_BLOCK_ELABORATION_START);
--         TWFG.Set_Task_Status(Current_Task,
--      PROTECTED_ENTRY_BLOCK_ELABORATING);
--      when PROTECTED_SUBPROGRAMS =>
--         Result := TWFG.Get_Current_Master(Current_Task);
--         TWFG.Set_Asynchronous_State(Result, TWFG.Get_Asynchronous_State(TWFG.Get_Parent_Master(Result)), TWFG.Get_Asynchronous_Trigger(TWFG.Get_Parent_Master(Result)));
--         TWFG.Set_Status(PROTECTED_SUBPROGRAM_ELABORATION_START);
--         TWFG.Set_Task_Status(Current_Task, PROTECTED_SUBPROGRAM_ELABORATING);
-- --       Put_Line("No.2");
--         Protected_ID := TWFG.Get_A_Out_Arc_Destination(Result);
--         if Protected_ID /= Null_Master_ID then
--    TWFG.Set_Arc(Protected_ID, Result,
--      FINALIZATION_WAITING_ARC);
--         end if;
--         TWFG.Set_Running_Task(TWFG.Get_A_Out_Arc_Destination(Result),
--       Current_Task);
--         TWFG.Remove_All_Out_Arc(Result);
-- --       TWFG.Set_All_Acceptance_Arc(Result);
--      when PROTECTED_OBJECTS =>
--         Result := TWFG.Search_Protected_Object(U_Name);
--         if Result = Null_Master_ID then
--    TWFG.Get_Master_ID(U_Class, U_Name, Null_Master_ID, Result);
--    TWFG.Set_Status(PROTECTED_OBJECT_EXIST, Result);
--         end if;
-- --       TWFG.Set_All_Acceptance_Arc(Result);
--      when others => null;
--   end case;
-- -- Put("TIC.Get_Master_ID で得た ID -");
-- -- Put(Integer(Result));
-- -- New_Line;
--   Ret_ID := Result;
--   exception
--      when TASKING_ERROR =>
--         Put_Line("!!! tasking error is raised in GET_MASTER_ID_ENTRANCE!");
--       end GET_MASTER_ID_ENTRANCE;

--       procedure GET_MASTER_ID_ENTRANCE(U_Class : in UNIT_CLASS;
--         U_Name : in STRING_BUFFER;
--         ID : in Task_ID;
--         Parent_ID : in ID_OF_MASTER;
--         Ret_ID : out ID_OF_MASTER) is
--   Result : ID_OF_MASTER;
--       begin
--   if Debug_Mode then
--      Put("task decl....");
--      Put_String(U_Name);
--      New_Line;
--   end if;
--   TWFG.Get_Master_ID(TASKS, U_Name, Parent_ID, Result);
--   TWFG.Set_Original_Master(Result, ID);
--   TWFG.Set_Current_Master(Result, ID);
-- -- Put_line(Image(ID));
--   TWFG.Set_Task_Status(ID, ELABORATING);
--   TWFG.Set_Server_ID(ID, U_Name);
--   Ret_ID := Result;
--   exception
--      when TASKING_ERROR =>
--         Put_Line("!!! tasking error is raised in GET_MASTER_ID(task)!");
--       end GET_MASTER_ID_ENTRANCE;

procedure SIMPLE_ENTRY_CALL(Callee_Task : in Task_ID;
                            Entry_Name : in String) is
   Callee_Original_Master : Master_Index_Link
     := TWFG.Get_Original_Master(Callee_Task);
begin
```

```
   if Debug_Mode then
      Put("SIMPLE_ENTRY_CALL - " & Image(Current_Task) & " to " &
            Image(Callee_Task) & "(");
      Put(Entry_Name);
      Put_Line(")");
   end if;
   TWFG.Set_Status(SIMPLE_ENTRY_CALL);
   TWFG.Set_Task_Status(Current_Task, SIMPLE_ENTRY_CALLING);
   TWFG.Set_Entry_Caller(TWFG.Get_Current_Master(Current_Task));


   if Callee_Original_Master = null then
      TWFG.Set_Standby_Arc(Callee_Task,
                           TWFG.Get_Current_Master(Current_Task),
                           ENTRY_CALLING_ARC,
                           Get_String_Buffer(Entry_Name));
   else
      TWFG.Set_Arc(TWFG.Get_Current_Master(Current_Task),
                   Callee_Original_Master,
                   ENTRY_CALLING_ARC);
      TWFG.Set_Arc_Label(TWFG.Get_Current_Master(Current_Task),
                         Callee_Original_Master,
                         Get_String_Buffer(Entry_Name));
      TWFG.Set_Calling_Entry(Callee_Task,
                             Get_String_Buffer(Entry_Name));
      --   if AM(TWFG.Get_Original_Master(Callee_Task),
      --         TWFG.Get_Current_Master(Current_Task)) /= ACCEPTANCE_WAITING_ARC
      --     or AL(TWFG.Get_Original_Master(Callee_Task),
      --   TWFG.Get_Current_Master(Current_Task)) /= Get_String_Buffer(Entry_Name) then
      if TWFG.Get_Arc_Class(Callee_Original_Master,
                            TWFG.Get_Original_Master(Current_Task)) /= ACCEPTANCE_WAITING_ARC
        or TWFG.Get_Arc_Label(Callee_Original_Master,
                              TWFG.Get_Original_Master(Current_Task)) /= Get_String_Buffer(Entry_Name) then
         TWFG.CHECK_CIRCULAR_DEADLOCKS(TWFG.Get_Current_Master(Current_Task),
                                       Callee_Original_Master);

      end if;
   end if;
   -- Put_Line("E");

exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in SIMPLE_ENTRY_CALL!");
   when others =>
      put_Line("!!! other error is raised in SIMPLE_ENTRY_CALL!");
end SIMPLE_ENTRY_CALL;

--       procedure PROTECTED_PROCEDURE_CALL(Protected_Name : in STRING_BUFFER;
--    Subprog_Name : in STRING_BUFFER) is

procedure PROTECTED_SUBPROGRAM_CALL
  (Callee_Protected_ID : in Protected_ID;
   Subprogram_Name : in String; Callee_Unit_Class : Unit_Class) is
   Subprog_Index, Protected_Index : Master_Index_Link;
   Current_Master : Master_Index_Link
     := Get_Current_Master(Current_Task);
begin
   -- function の場合にも対応しなければならない
   TWFG.Get_Master_Index(Callee_Unit_Class,
                         Get_String_Buffer(Subprogram_Name),
                         Current_Master, Subprog_Index);
   TWFG.Set_Status(BLOCKED_BY_PROTECTION, Subprog_Index);
   TWFG.Set_Task_Status(Current_Task, BLOCKED_BY_PROTECTION);
   TWFG.Set_Arc(Current_Master, Subprog_Index,
                FINALIZATION_WAITING_ARC);
   Protected_Index := Pid_To_Mid(Callee_Protected_ID);
   if Current_Master.Protected_Object_Master =
     Protected_Index then
      -- Put_line("No.3");
      Subprog_Index.Protected_Object_Master :=
        Current_Master.Protected_Object_Master;
   else

      if Protected_Index = null then
         TWFG.Get_Master_Index(PROTECTED_OBJECTS,
                               Get_String_Buffer(Protected_Name(Callee_Protected_ID)),
                               null, Protected_Index);
         Protected_Index.Protected_Object_Master :=
           Protected_Index;
         Pid_To_Mid(Callee_Protected_ID) := Protected_Index;
      end if;
      Subprog_Index.Protected_Object_Master :=
        Protected_Index;
      TWFG.Set_Asynchronous_State(Protected_Index,
                                  NO_ASYNC, null);

      if Current_Master.Protected_Object_Master /=
        Protected_Index then
         TWFG.Set_Arc(Subprog_Index, Protected_Index,
                      PROTECTION_WAITING_ARC);
         TWFG.CHECK_CIRCULAR_DEADLOCKS
           (TWFG.Get_Current_Master(Current_Task), Subprog_Index);
      end if;
   end if;

   TWFG.Set_Current_Master(Subprog_Index);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in PROTECTED_SUBPROGRAM_CALL!");
end PROTECTED_SUBPROGRAM_CALL;

--       procedure PROTECTED_SUBPROGRAM_CALL_INTERNAL(
--    Subprog_Name : in STRING_BUFFER) is
--   Subprog_ID : ID_OF_MASTER;
--   Current_Master : ID_OF_MASTER :=TWFG.Get_Current_Master(Current_Task);
--       begin

-- -- Put("PROTECTED_SUBPROGRAM_CALL - " & Image(Current_Task) & "(");
-- -- Put_String(Protected_Name);
```

```
-- -- Put(".");
-- -- Put_String(Subprog_Name);
-- -- Put_Line(")");
--   TWFG.Get_Master_ID(PROTECTED_SUBPROGRAMS, Subprog_Name,
--     Current_Master, Subprog_ID);
--   TWFG.Set_Status(BLOCKED_BY_PROTECTION, Subprog_ID);
--   TWFG.Set_Task_Status(Current_Task, BLOCKED_BY_PROTECTION);
-- -- Put_line("No.3");
--   Master_ID_List(Subprog_ID).Protected_Object_Master_ID :=
--     Master_ID_List(Current_Master).Protected_Object_Master_ID;
--   TWFG.Set_Arc(TWFG.Get_Current_Master(Current_Task), Subprog_ID,
--       FINALIZATION_WAITING_ARC);

--   TWFG.Set_Current_Master(Subprog_ID);
--   exception
--      when TASKING_ERROR =>
--         Put_Line("!!! tasking error is raised in PROTECTED_SUBPROGRAM_CALL_INTERNAL!");
--       end PROTECTED_SUBPROGRAM_CALL_INTERNAL;

procedure PROTECTED_SUBPROGRAM_CALL_COMPLETION
  (Callee_Unit_Class : Unit_Class) is
begin
   -- Put_Line("PROTECTED_SUBPROGRAM_CALL_COMPLETION - " & Image(Current_Task));
   -- function の場合にも対応しなければならない
   TWFG.Remove_All_Out_Node(TWFG.Get_Current_Master(Current_Task),
                            Callee_Unit_Class);
   TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in PROTECTED_SUBPROGRAM_CALL_COMPLETION!");
end PROTECTED_SUBPROGRAM_CALL_COMPLETION;

procedure PROTECTED_ENTRY_CALL_COMPLETION is
begin
   if Debug_Mode then
      Put_Line
        ("PROTECTED_ENTRY_CALL_COMPLETION - " & Image(Current_Task));
   end if;
   TWFG.Remove_All_Out_Arc(TWFG.Get_Current_Master(Current_Task),
                           PROTECTED_ENTRY_CALLING_ARC);
   TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in PROTECTED_ENTRY_CALL_COMPLETION!");
end PROTECTED_ENTRY_CALL_COMPLETION;

--       procedure PROTECTED_ENTRY_CALL(Protected_Name : in STRING_BUFFER;
--       Entry_Name : in STRING_BUFFER) is
procedure PROTECTED_ENTRY_CALL(Callee_Protected_ID : in Protected_ID;
                               Entry_Name : in String) is
   Protected_Index : Master_Index_Link;
   Parent_Master : Master_Index_Link := Get_Current_Master(Current_Task);
begin
   -- Put("PROTECTED_ENTRY_CALL - " & Image(Current_Task) & "(");
   -- Put_String(Protected_Name);
   -- Put(".");
   -- Put_String(Entry_Name);
   -- Put_Line(")");
   if Debug_Mode then
      Put_Line("PROTECTED_ENTRY_CALL: protected ID - "
               & Natural'IMAGE(Callee_Protected_ID));
   end if;

   Protected_Index := Pid_To_Mid(Callee_Protected_ID);
   if Protected_Index = null then
      if Debug_Mode then
         Put_Line("New master for a protected object was created.");
      end if;
      TWFG.Get_Master_Index(PROTECTED_OBJECTS,
                            Get_String_Buffer(Protected_Name(Callee_Protected_ID)),
                            TWFG.Get_Current_Master(Current_Task),
                            Protected_Index);
      Pid_To_Mid(Callee_Protected_ID) := Protected_Index;
      Protected_Index.Protected_Object_Master :=
        Protected_Index;
   end if;
   Put_Line("rprca2");
   TWFG.Set_Arc(TWFG.Get_Current_Master(Current_Task), Protected_Index,
                PROTECTED_ENTRY_CALLING_ARC);
   TWFG.Set_Arc_Label(TWFG.Get_Current_Master(Current_Task),
                      Protected_Index, Get_String_Buffer(Entry_Name));
   TWFG.Set_Status(PROTECTED_ENTRY_CALL);
   TWFG.Set_Task_Status(Current_Task, PROTECTED_ENTRY_CALLING);
   if TWFG.Get_Current_Master(Current_Task) /= null
     and Protected_Index /= null then
      TWFG.CHECK_CIRCULAR_DEADLOCKS
        (TWFG.Get_Current_Master(Current_Task),
         Protected_Index);
   end if;
   TWFG.Set_Current_Master(Protected_Index);
   TWFG.Set_Entry_Caller(Parent_Master);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in PROTECTED_ENTRY_CALL!");
end PROTECTED_ENTRY_CALL;

procedure SUBPROGRAM_CALL_COMPLETION(Callee_Unit_Class : Unit_Class) is
begin
   -- Put_Line("SUBPROGRAM_CALL_COMPLETION - " & Image(Current_Task));
   -- function の場合の対応が...
   TWFG.Remove_All_Out_Node(TWFG.Get_Current_Master(Current_Task)
                            , Callee_Unit_Class);

   TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in SUBPROGRAM_CALL_COMPLETION!");
end SUBPROGRAM_CALL_COMPLETION;
```

```
--       procedure SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION is
--       begin
-- -- Put_Line("SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION- " & Image(Current_Task));
--   TWFG.Remove_All_Out_Node(TWFG.Get_Current_Master(Current_Task), SUBPROGRAMS);
--   TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
--   exception
--      when TASKING_ERROR =>
--        Put_Line("!!! tasking error is raised in SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION!");
--       end SUBPROGRAM_CALL_COMPLETION_IN_DECLARATION;

procedure BLOCK_CALL_COMPLETION is
begin
   -- Put_Line("BLOCK_CALL_COMPLETION - " & Image(Current_Task));
   TWFG.Remove_All_Out_Node(TWFG.Get_Current_Master(Current_Task), BLOCKS);
   TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in BLOCK_CALL_COMPLETION!");
end BLOCK_CALL_COMPLETION;

procedure ABORT_START(Object_Tasks : in Task_ID_List) is
begin
   for I in Object_Tasks'RANGE loop
      if Debug_Mode then
         Put_Line("Abort: " & Image(Object_Tasks(I)));
      end if;
      TWFG.Set_Task_Status(Object_Tasks(I), ABNORMAL);
      TWFG.Abort_Node(TWFG.Get_Original_Master(Object_Tasks(I)));
      if Debug_Mode then
         Put_Line("Abort: done.");
      end if;
   end loop;
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in Abort_Start!");

end ABORT_START;

--       -- いらん
--       procedure PROTECTED_BLOCK_EXECUTION_START is
--       begin
-- -- Put_Line("PROTECTED_BLOCK_EXECUTION_START - " & Image(Current_Task));
--   TWFG.Set_Status(BLOCK_EXECUTION_START);
--   TWFG.Set_Task_Status(Current_Task,
--        PROTECTED_SUBPROGRAM_EXECUTION_WAITING);
--   exception
--      when TASKING_ERROR =>
--        Put_Line("!!! tasking error is raised in PROTECTED_BLOCK_EXECUTION_START!");
--       end PROTECTED_BLOCK_EXECUTION_START;

--       -- いらん
--       procedure PROTECTED_BLOCK_EXECUTION_END is
--       begin
-- -- Put_Line("PROTECTED_BLOCK_EXECUTION_END - " & Image(Current_Task));
--   TWFG.Set_Status(BLOCK_EXECUTION_COMPLETION);
--   TWFG.Set_Completed_Master(TWFG.Get_Current_Master(Current_Task));
--   TWFG.Set_Current_Master(TWFG.Get_Parent_Master(TWFG.Get_Current_Master(Current_Task)));
--   TWFG.Set_Task_Status(Current_Task, PROTECTED_SUBPROGRAM_COMPLETED);
--   exception
--      when TASKING_ERROR =>
--        Put_Line("!!! tasking error is raised in PROTECTED_BLOCK_EXECUTION_END!");
--       end PROTECTED_BLOCK_EXECUTION_END;

procedure Get_CDT(CDT_OF_TASK : CDT; CDT_OF_TASK_TYPE : CDT_For_TT) is
   Temp_Task_Index : Task_Index_Link;
   Index : Integer;
begin
   if Debug_Mode then
      Put_Line("Get_CDT: " & Image(Current_Task));
   end if;
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                  Current_Task);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   end if;
   -- CDT を取り込み
   Index := 1;
   while Temp_Task_Index.Task_Cdt(Index) /= Null_Cdt_Element loop
      Index := Index + 1;
   end loop;
   for I in CDT_OF_TASK'RANGE loop
      Temp_Task_Index.Task_CDT(Index) := CDT_OF_TASK(I);
      Temp_Task_Index.Task_CDT(Index).
      Index := Index + 1;
   end loop;

   -- CDT for task type を取り込み
   Index := 1;
   while Temp_Task_Index.Task_Cdt_For_TT(Index) /= Null_Cdt_Element_for_TT loop
      Index := Index + 1;
   end loop;
   for I in CDT_OF_TASK_TYPE'RANGE loop
      Temp_Task_Index.Task_CDT_For_TT(Index) := CDT_OF_TASK_TYPE(I);
      Index := Index + 1;
   end loop;
   -- 書き戻し
   Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                  Current_Task);
   if Debug_Mode then
      Put_Line("Get_CDT(end): " & Image(Current_Task));
   end if;
end Get_CDT;

procedure TASK_ACTIVATION_START (Block_Name : String;
                                 Parent_Task : Task_ID;
                                 This_Unit_Class : Unit_Class;
                                 Unit_ID_Of_Task : Unit_ID)
```

```
--when True
is
   --       when Task_Index_List.SEARCH_NODE(All_Task_ID, Current_Task)
   --  /= Null_Task_Index is
   New_Master : Master_Index_Link;
   Nownode : Node_List_Link;
   --   Temp_Task_Index : Task_Index_Link;
   --   Hoge : Task_Index;
begin
   -- Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, Current_Task);
   if Debug_Mode then
      Put_Line("TASK_ACTIVATION_START - " & Image(Current_Task));
      --    Put_String(TWFG.Get_Master_Name(TWFG.Get_Current_Master(Current_Task)));
      --    New_Line;
   end if;
   TWFG.Get_Master_Index(TASKS, Get_String_Buffer(Block_Name),
                         TWFG.Get_Current_Master(Parent_Task),
                         New_Master);
   -- TWFG.Get_Master_ID(TASKS, Get_String_Buffer(Block_Name), Task_Index_List.SEARCH_NODE(All_Task_ID, Current_Task).Current_Master, New_Master_ID);
   TWFG.Set_Original_Master(New_Master, Current_Task);
   TWFG.Set_Current_Master(New_Master, Current_Task);
   -- Put_line(Image(ID));
   -- TWFG.Set_Server_ID(Current_Task, U_Name); --?

   -- TWFG.Get_Master_ID(TASKS, Null_STRING_BUFFER, MASTER_ID,
   --    New_Master_ID);
   -- TWFG.Set_Original_Master(New_Master_ID);
   -- TWFG.Set_Current_Master(New_Master_ID);
   TWFG.Set_Status(ACTIVATION_START);
   TWFG.Set_Task_Status(Current_Task, ACTIVATING);
   TWFG.Set_Source_Unit_ID(Unit_ID_Of_Task);
   TWFG.Set_All_Acceptance_Arc(New_Master);
   TWFG.Set_Arc(TWFG.Get_Current_Master(Parent_Task), New_Master,
                ACTIVATION_WAITING_ARC);
   TWFG.Set_Reverse_Completion_Arc;
   -- ノード生成前からエントリ待ちしていたノードからの枝を生成
   Nownode := TWFG.Get_Standby_Arc;
   while Nownode /= null loop
      TWFG.Set_Arc(Nownode.Mynode, New_Master, Nownode.Arcclass);
      TWFG.Set_Arc_Label(Nownode.Mynode, New_Master, Nownode.Arclabel);
      Nownode := Nownode.Next;
   end loop;

   if Debug_Mode then
      Put_Line("Task was activated successfully. - " & Image(Current_Task));
   end if;
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in TASK_ACTIVATION_START!");
end TASK_ACTIVATION_START;

procedure TASK_ACTIVATION_COMPLETION is
begin
   if Debug_Mode then
      Put_Line("TASK_ACTIVATION_COMPLETION - " & Image(Current_Task));
   end if;
   --   if TWFG.Get_Master_Class(TWFG.Get_Parent_Master(TWFG.Get_Current_Master(Current_Task))) /= TASKS then
   -- --    Put_Line("No.4");
   --      TWFG.Set_Arc(TWFG.Get_Parent_Master(TWFG.Get_Current_Master(Current_Task)), TWFG.Get_Current_Master(Current_Task),
   --   FINALIZATION_WAITING_ARC);
   --   else

   TWFG.Remove_Arc(TWFG.Get_Current_Master(Current_Task).Parent_Master,
                   TWFG.Get_Current_Master(Current_Task));

   -- end if;
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in TASK_ACTIVATION_COMPLETION!");
   when others =>
      Put_Line("other exception is raised in TASK_ACTIVATION_COMPLETION!");
      raise;
end TASK_ACTIVATION_COMPLETION;

--       procedure TASK_EXECUTION_START is
--       begin
-- -- Put_Line("TASK_EXECUTION_START - " & Image(Current_Task));
--   TWFG.Set_Status(ACTIVATION_COMPLETION);
--   TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
-- -- TWFG.Set_Status(BLOCK_EXECUTION_START);
--   exception
--      when TASKING_ERROR =>
--         Put_Line("!!! tasking error is raised in TASK_EXECUTION_START!");
--       end TASK_EXECUTION_START;

procedure ACCEPT_START(Entry_Name : String; Is_Open_Accept : Boolean) is
begin
   if Debug_Mode then
      Put("ACCEPT_START - " & Image(Current_Task) & "(");
      Put(Entry_Name);
      Put_Line(")");
   end if;
   TWFG.Set_Status(ACCEPTANCE);
   TWFG.Set_Task_Status(Current_Task, ACCEPTING);
   if (not TWFG.Is_Entry_Call(TWFG.Get_Current_Master(Current_Task),
                              Get_String_Buffer(Entry_Name))) and
     Is_Open_Accept then
      TWFG.Set_All_Acceptance_Arc(Get_String_Buffer(Entry_Name));
   end if;
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in ACCEPT_START!");
end ACCEPT_START;

procedure RENDEZVOUS_START(Caller_Task : Task_ID) is
begin
   if Debug_Mode then
      Put_Line("RENDEZVOUS_START - (Caller)" & Image(Caller_Task) & " (Callee)" & Image(Current_Task));
   end if;
```

```
   TWFG.Set_Status(RENDEZVOUS_START);
   TWFG.Set_Task_Status(Current_Task, SUSPENDED_BY_RENDEZVOUS);
   TWFG.Set_Task_Status(Caller_Task, SUSPENDED_BY_RENDEZVOUS);
   if TWFG.Get_Arc_Class(TWFG.Get_Entry_Caller(Caller_Task),
                         TWFG.Get_Current_Master(Current_Task))
     /= ENTRY_CALLING_ARC then
      TWFG.Set_Arc(TWFG.Get_Entry_Caller(Caller_Task),
                   TWFG.Get_Current_Master(Current_Task),
                   ENTRY_CALLING_ARC);
      TWFG.Set_Arc_Label(TWFG.Get_Entry_Caller(Caller_Task),
                         TWFG.Get_Current_Master(Current_Task),
                         Get_Calling_Entry
                           (TWFG.Get_Current_Master(Current_Task)));
   end if;
   TWFG.Set_Entry_Caller(TWFG.Get_Entry_Caller(Caller_Task));
   TWFG.Remove_All_Out_Arc(TWFG.Get_Current_Master(Current_Task),
                           ACCEPTANCE_WAITING_ARC);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in RENDEZVOUS_START!");
end RENDEZVOUS_START;

procedure RENDEZVOUS_END(Caller_Task : Task_ID) is
begin
   if Debug_Mode then
      Put_Line("RENDEZVOUS_END - (Caller)" & Image(Caller_Task) & " (Callee)" & Image(Current_Task));
   end if;
   TWFG.Remove_Arc(TWFG.Get_Entry_Caller(Caller_Task),
                   TWFG.Get_Current_Master(Current_Task));
   TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in RENDEZVOUS_END!");
end RENDEZVOUS_END;

--       procedure TASK_EXECUTION_END is
--       begin
-- -- Put_Line("TASK_EXECUTION_END - " & Image(Current_Task));
--   TWFG.Set_Task_Status(Current_Task, COMPLETED);
--   TWFG.Remove_Node(TWFG.Get_Current_Master(Current_Task));
--   TWFG.Remove_Task;
--   exception
--      when TASKING_ERROR =>
--         Put_Line("!!! tasking error is raised in TASK_EXECUTION_END!");

--       end TASK_EXECUTION_END;

procedure ACCEPT_WITH_TERMINATE_START
  (Entry_Name : String; Is_Open_Accept : Boolean) is
begin
   -- Put_Line("ACCEPT_WITH_TERMINATE_START - " & Image(Current_Task));
   if (not TWFG.Is_Entry_Call(TWFG.Get_Current_Master(Current_Task),
                              Get_String_Buffer(Entry_Name))) and
     Is_Open_Accept then
      TWFG.Remove_All_Out_Arc(TWFG.Get_Current_Master(Current_Task),
                              COMPLETION_WAITING_ARC);
      TWFG.Set_All_Acceptance_Arc(Get_String_Buffer(Entry_Name));
   elsif TWFG.Get_Current_Master(Current_Task).Nextnodes = null then
      TWFG.Set_Completion_Arc;
   end if;
   TWFG.Set_Status(TERMINATION_SELECTION);
   TWFG.Set_Task_Status(Current_Task, TERMINATION_SELECTING);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in ACCEPT_WITH_TERMINATE_START!");
end ACCEPT_WITH_TERMINATE_START;

procedure ASYNCHRONOUS_ENTRY_CALL(Callee_Task : in Task_ID;
                                  Entry_Name : in String) is
begin
   -- Put_Line("ASYNCHRONOUS_ENTRY_CALL - " & Image(Current_Task));
   TWFG.Set_Status(ASYNCHRONOUS_ENTRY_CALL);
   TWFG.Set_Entry_Caller(TWFG.Get_Current_Master(Current_Task));
   TWFG.Set_Asynchronous_State(TWFG.Get_Current_Master(Current_Task),
                               ASYNC_ENTRY_CALL,
                               TWFG.Get_Original_Master(Callee_Task));
   TWFG.Set_Calling_Entry(Callee_Task, Get_String_Buffer(Entry_Name));
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in ASYNCHRONOUS_ENTRY_CALL!");
end ASYNCHRONOUS_ENTRY_CALL;

-- 仮
procedure ASYNCHRONOUS_PROTECTED_ENTRY_CALL(Callee_Task : in Task_ID;
                                            Entry_Name : in String) is
begin
   -- Put_Line("ASYNCHRONOUS_ENTRY_CALL - " & Image(Current_Task));
   TWFG.Set_Status(ASYNCHRONOUS_ENTRY_CALL);
   TWFG.Set_Entry_Caller(TWFG.Get_Current_Master(Current_Task));
   TWFG.Set_Asynchronous_State(TWFG.Get_Current_Master(Current_Task),
                               ASYNC_ENTRY_CALL,
                               TWFG.Get_Original_Master(Callee_Task));
   TWFG.Set_Calling_Entry(Callee_Task, Get_String_Buffer(Entry_Name));
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in ASYNCHRONOUS_PROTECTED_ENTRY_CALL!");
end ASYNCHRONOUS_PROTECTED_ENTRY_CALL;

procedure ASYNCHRONOUS_DELAY_STMT is
begin
   TWFG.Set_Status(ASYNCHRONOUS_DELAY);
   TWFG.Set_Asynchronous_State(TWFG.Get_Current_Master(Current_Task),
                               ASYNC_DELAY_STMT, null);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in ASYNCHRONOUS_DELAY_STMT!");
end ASYNCHRONOUS_DELAY_STMT;

procedure ASYNCHRONOUS_TRIGGER_END is
```

```
   Master : Master_Index_Link := Get_Current_Master(Current_Task);
   Nownode : Node_List_Link := Master.Nextnodes;
   Tmp_Nownode : Node_List_Link;
begin
   while Nownode /= null loop
      if Nownode.Arcclass /= ENTRY_CALLING_ARC and then
        Nownode.Arcclass /= PROTECTED_ENTRY_CALLING_ARC and then
        Nownode.Arcclass /= COMPLETION_WAITING_ARC and then
        Nownode.Arcclass /= PROTECTION_WAITING_ARC and then
        Nownode.Mynode.Master_Class /= TASKS and then
        Nownode.Mynode.Master_Class /= LIBRARY_PACKAGES then
         Tmp_Nownode := Nownode;
         Nownode := Nownode.Next;
         Abort_Node(Tmp_Nownode.Mynode);
      else
         Nownode := Nownode.Next;
      end if;
   end loop;
   --   for I in ID_OF_MASTER loop
   --      if AM(Master_ID, I) /= ENTRY_CALLING_ARC and
   --        AM(Master_ID, I) /= PROTECTED_ENTRY_CALLING_ARC and
   --        AM(Master_ID, I) /= COMPLETION_WAITING_ARC and
   --        AM(Master_ID, I) /= PROTECTION_WAITING_ARC and
   --        Master_ID_List(I).Master_Class /= TASKS and
   --        Master_ID_List(I).Master_Class /= LIBRARY_PACKAGES then
   --         Abort_Node(I);
   --      end if;
   --   end loop;
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in ASYNCHRONOUS_TRIGGER_END!");
end ASYNCHRONOUS_TRIGGER_END;



procedure ASYNCHRONOUS_SELECT_END is
begin
   TWFG.Set_Asynchronous_State(TWFG.Get_Current_Master(Current_Task),
                               NO_ASYNC, null);
   TWFG.Remove_All_Out_Arc(TWFG.Get_Current_Master(Current_Task),
                           ENTRY_CALLING_ARC);
   TWFG.Remove_All_Out_Arc(TWFG.Get_Current_Master(Current_Task),
                           PROTECTED_ENTRY_CALLING_ARC);
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in ASYNCHRONOUS_ASYNCHRONOUS_SELECT_END!");
end ASYNCHRONOUS_SELECT_END;

procedure CONDITIONAL_ENTRY_CALL is
begin
   TWFG.Set_Status(CONDITIONAL_ENTRY_CALL);
   TWFG.Set_Task_Status(Current_Task, CONDITIONAL_ENTRY_CALLING);
   TWFG.Set_Entry_Caller(TWFG.Get_Current_Master(Current_Task));
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in CONDITIONAL_ENTRY_CALL!");
end CONDITIONAL_ENTRY_CALL;

procedure TIMED_ENTRY_CALL is
begin
   TWFG.Set_Status(TIMED_ENTRY_CALL);
   TWFG.Set_Task_Status(Current_Task, TIMED_ENTRY_CALLING);
   TWFG.Set_Entry_Caller(TWFG.Get_Current_Master(Current_Task));
exception
   when TASKING_ERROR =>
      Put_Line("!!! tasking error is raised in TIMED_ENTRY_CALL!");
end TIMED_ENTRY_CALL;

procedure REQUEUE_EXEC(Object_Task : in Task_ID;
                       Entry_Name : in String) is
   Caller_Master : Master_Index_Link;
   Current_Master : Master_Index_Link
     := Get_Current_Master(Current_Task);
begin
   if Get_Task_Status(Current_Task) = SUSPENDED_BY_RENDEZVOUS then
      Caller_Master := Get_Entry_Caller(Current_Task);
   else
      Caller_Master := Current_Master.Parent_Master;
   end if;
   TWFG.Remove_Arc(Caller_Master, Current_Master);
   TWFG.Set_Arc(Caller_Master, TWFG.Get_Original_Master(Object_Task),
                ENTRY_CALLING_ARC);
   TWFG.Set_Arc_Label(Caller_Master,
                      TWFG.Get_Original_Master(Object_Task),
                      Get_String_Buffer(Entry_Name));
   TWFG.Set_Status(SIMPLE_ENTRY_CALL, Caller_Master);
   TWFG.Set_Task_Status(Caller_Master.Running_Task_ID,
                        SIMPLE_ENTRY_CALLING);
   -- conditional, timed, async への対応が必要
   TWFG.Set_Calling_Entry(Get_String_Buffer(Entry_Name),
                          Caller_Master);
   if TWFG.Get_Arc_Class(TWFG.Get_Original_Master(Object_Task),
                         Caller_Master)
     /= ACCEPTANCE_WAITING_ARC or
     TWFG.Get_Arc_Label(TWFG.Get_Original_Master(Object_Task),
                        Caller_Master) /=
     Get_String_Buffer(Entry_Name) then
      TWFG.CHECK_CIRCULAR_DEADLOCKS
        (Caller_Master,
         TWFG.Get_Original_Master(Object_Task));
   end if;
   if Get_Task_Status(Current_Task) = SUSPENDED_BY_RENDEZVOUS then
      TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
   else
      TWFG.BLOCK_EXECUTION_COMPLETION;
   end if;
end REQUEUE_EXEC;

procedure REQUEUE_WITH_ABORT_EXEC(Object_Task : in Task_ID;
```

```
                                Entry_Name : in String) is
   Caller_Master : Master_Index_Link;
   Current_Master : Master_Index_Link
     := Get_Current_Master(Current_Task);
begin
   if Get_Task_Status(Current_Task) = SUSPENDED_BY_RENDEZVOUS then
      Caller_Master := Get_Entry_Caller(Current_Task);
   else
      Caller_Master := Current_Master.Parent_Master;
   end if;
   TWFG.Remove_Arc(Caller_Master, Current_Master);
   if TWFG.Get_Status(Caller_Master) /= CONDITIONAL_ENTRY_CALL and
     TWFG.Get_Status(Caller_Master) /= TIMED_ENTRY_CALL and
     (TWFG.Get_Asynchronous_State(Caller_Master) /= ASYNC_ENTRY_CALL or
        TWFG.Get_Asynchronous_Trigger(Caller_Master) /= Current_Master) then
      TWFG.Set_Arc(Caller_Master, TWFG.Get_Original_Master(Object_Task),
                   ENTRY_CALLING_ARC);
      TWFG.Set_Arc_Label(Caller_Master,
                         TWFG.Get_Original_Master(Object_Task),
                         Get_String_Buffer(Entry_Name));
      TWFG.Set_Status(SIMPLE_ENTRY_CALL, Caller_Master);
      TWFG.Set_Task_Status(Caller_Master.Running_Task_ID,
                           SIMPLE_ENTRY_CALLING);
      -- conditional, timed, async への対応が必要
      TWFG.Set_Calling_Entry(Get_String_Buffer(Entry_Name), Caller_Master);
      if TWFG.Get_Arc_Class(TWFG.Get_Original_Master(Object_Task), Caller_Master)
        /= ACCEPTANCE_WAITING_ARC or
        TWFG.Get_Arc_Label(TWFG.Get_Original_Master(Object_Task),
                           Caller_Master) /=
        Get_String_Buffer(Entry_Name) then
         TWFG.CHECK_CIRCULAR_DEADLOCKS
           (Caller_Master,
            TWFG.Get_Original_Master(Object_Task));
      end if;

   end if;
   if Get_Task_Status(Current_Task) = SUSPENDED_BY_RENDEZVOUS then
      TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
   else
      TWFG.BLOCK_EXECUTION_COMPLETION;
   end if;
end REQUEUE_WITH_ABORT_EXEC;

procedure PROTECTED_REQUEUE_EXEC(Callee_Protected_ID : Protected_ID;
                                 Entry_Name : in String) is
   Caller_Master : Master_Index_Link;
   Current_Master : Master_Index_Link
     := Get_Current_Master(Current_Task);
   Protected_Index : Master_Index_Link
     := Pid_To_Mid(Callee_Protected_ID);
begin
   if Get_Task_Status(Current_Task) = SUSPENDED_BY_RENDEZVOUS then
      Caller_Master := Get_Entry_Caller(Current_Task);
   else
      Caller_Master := Current_Master.Parent_Master;
   end if;

   if Protected_Index = null then
      TWFG.Get_Master_Index(PROTECTED_OBJECTS,
                            Get_String_Buffer(Protected_Name(Callee_Protected_ID)),
                            Caller_Master,
                            Protected_Index);
   end if;

   TWFG.Remove_Arc(Caller_Master, Current_Master);
   Put_Line("prca3");
   TWFG.Set_Arc(Caller_Master, Protected_Index,
                PROTECTED_ENTRY_CALLING_ARC);
   TWFG.Set_Arc_Label(Caller_Master, Protected_Index,
                      Get_String_Buffer(Entry_Name));
   TWFG.Set_Status(PROTECTED_ENTRY_CALL, Caller_Master);
   TWFG.Set_Task_Status(Caller_Master.Running_Task_ID,
                        PROTECTED_ENTRY_CALLING);
   TWFG.CHECK_CIRCULAR_DEADLOCKS(Caller_Master, Protected_Index);
   TWFG.Set_Current_Master(Protected_Index,
                           Caller_Master.Running_Task_ID);
   TWFG.Set_Entry_Caller(Caller_Master);
   if Get_Task_Status(Current_Task) = SUSPENDED_BY_RENDEZVOUS then
      TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
   else
      TWFG.BLOCK_EXECUTION_COMPLETION;
   end if;
end PROTECTED_REQUEUE_EXEC;

procedure PROTECTED_REQUEUE_WITH_ABORT_EXEC
  (Callee_Protected_ID : Protected_ID;
   Entry_Name : in String) is
   Caller_Master : Master_Index_Link;
   Current_Master : Master_Index_Link
     := Get_Current_Master(Current_Task);
   Protected_Index : Master_Index_Link
     := Pid_To_Mid(Callee_Protected_ID);
begin
   if Get_Task_Status(Current_Task) = SUSPENDED_BY_RENDEZVOUS then
      Caller_Master := Get_Entry_Caller(Current_Task);
   else
      Caller_Master := Current_Master.Parent_Master;
   end if;

   if Protected_Index = null then
      TWFG.Get_Master_Index
        (PROTECTED_OBJECTS,
         Get_String_Buffer(Protected_Name(Callee_Protected_ID)),
         Caller_Master,
         Protected_Index);
   end if;

   TWFG.Remove_Arc(Caller_Master, Current_Master);
   if TWFG.Get_Status(Caller_Master) /= CONDITIONAL_ENTRY_CALL and
```

```
     TWFG.Get_Status(Caller_Master) /= TIMED_ENTRY_CALL and
     (TWFG.Get_Asynchronous_State(Caller_Master) /= ASYNC_ENTRY_CALL or
        TWFG.Get_Asynchronous_Trigger(Caller_Master) /= Current_Master) then
      Put_Line("prca4");
      TWFG.Set_Arc(Caller_Master, Protected_Index,
                   PROTECTED_ENTRY_CALLING_ARC);
      TWFG.Set_Arc_Label(Caller_Master, Protected_Index,
                         Get_String_Buffer(Entry_Name));
      TWFG.Set_Status(PROTECTED_ENTRY_CALL, Caller_Master);
      TWFG.Set_Task_Status(Caller_Master.Running_Task_ID,
                           PROTECTED_ENTRY_CALLING);
      TWFG.CHECK_CIRCULAR_DEADLOCKS(Caller_Master, Protected_Index);
      TWFG.Set_Current_Master(Protected_Index,
                              Caller_Master.Running_Task_ID);
      TWFG.Set_Entry_Caller(Caller_Master);

   end if;
   if Get_Task_Status(Current_Task) = SUSPENDED_BY_RENDEZVOUS then
      TWFG.Set_Task_Status(Current_Task, WORKING_FOR_INTERNAL_AFFAIRS);
   else
      TWFG.BLOCK_EXECUTION_COMPLETION;
   end if;
end PROTECTED_REQUEUE_WITH_ABORT_EXEC;

procedure ENQUEUE_CALL_START(Caller_Task : Task_ID;
                             Queue_Name: in String) is
begin
   put("ENQUEUE_CALL_START");
end ENQUEUE_CALL_START;


procedure Get_Master_Index(U_Class : in UNIT_CLASS;
                           U_Name : in STRING_BUFFER;
                           Parent_Master : in Master_Index_Link;
                           New_Master : out Master_Index_link) is
   -- Temp_Master_ID : ID_OF_MASTER := Null_Master_ID + 1;
begin
   if Debug_Mode then
      Put("get_master_id:");
      Put_String(U_Name);
      New_Line;
   end if;
   --   while Temp_Master_ID <= Maximum_Master loop
   --      exit when Master_ID_List(Temp_Master_ID).Master_Class = N_U_L_L;
   --      Temp_Master_ID := Temp_Master_ID + 1;
   --   end loop;
   --   if Temp_Master_ID <= Maximum_Master then
   New_Master := new Master_Index;
   New_Master.Master_Class := U_Class;
   New_Master.Parent_Master := Parent_Master;
   New_Master.Master_Name := U_Name;
   New_Master.Running_Task_ID := Current_Task;
   New_Master.Asyncstate.Status := NO_ASYNC;
   New_Master.Master_ID := Master_ID_Counter;
   Master_ID_Counter := Master_ID_Counter + 1;

   if Master_List = null then
      Master_List := new Node_List'(null, New_Master,
                                    NO_ARC, Null_String_Buffer);
      Now_Master_List := Master_List;
   else
      Now_Master_List.Next := new Node_List'
        (null, New_Master,
         NO_ARC, Null_String_Buffer);
      Now_Master_List := Now_Master_List.Next;
   end if;
   --    Int_IO.Put(Integer(Temp_Master_ID));
   --    Put_String(U_Name);
   -- New_Master_ID := Temp_Master_ID;
   --   else
   --      Put_Line("Master ID Pool is full!");
   --      New_Master_ID := Null_Master_ID;
   --   end if;
end Get_Master_Index;

procedure Set_Current_Master(Master : in Master_Index_Link) is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                  Current_Task);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   end if;
   -- if Temp_Task_Index = Null_Task_Index then
   --    Put_Line("null?");
   -- else
   --    Put("Set_Current_Master で得た Master_ID -- ");
   --    Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
   --    New_Line;
   -- end if;
   if Master /= null then
      Temp_Task_Index.Current_Master := Master;
   end if;
   -- Put("Set_Current_Master 設定しようとしてる Master_ID -- ");
   -- Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
   -- New_Line;
   --   Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   --   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
   --Current_Task);
   -- Put_Line(Image(Current_Task));
   -- Put("Set_Current_Master 設定した Master_ID -- ");
   -- Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
   -- New_Line;

end Set_Current_Master;

-- new
procedure Set_Source_Unit_ID(ID : Unit_ID) is
```

```
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                  Current_Task);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   end if;
   -- if Temp_Task_Index = Null_Task_Index then
   --    Put_Line("null?");
   -- else
   --    Put("Set_Current_Master で得た Master_ID -- ");
   --    Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
   --    New_Line;
   -- end if;
   Temp_Task_Index.Source_Unit_ID := ID;
   -- Put("Set_Current_Master 設定しようとしてる Master_ID -- ");
   -- Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
   -- New_Line;
   --   Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   --   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
   --  Current_Task);
   -- Put_Line(Image(Current_Task));
   -- Put("Set_Current_Master 設定した Master_ID -- ");
   -- Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
   -- New_Line;

end Set_Source_Unit_ID;

procedure Set_Current_Master(Master : in Master_Index_Link;
                             ID : in Task_ID) is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                  ID);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, ID, Temp_Task_Index);
   end if;
   -- if Temp_Task_Index = Null_Task_Index then
   --    Put_Line("null?");
   -- else
   --    Put("Set_Current_Master で得た Master_ID -- ");
   --    Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
   --    New_Line;
   -- end if;
   Temp_Task_Index.Current_Master := Master;
   -- Put("Set_Current_Master 設定しようとしてる Master_ID -- ");
   -- Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
   -- New_Line;
   -- Task_Index_List.SET_NODE(All_Task_ID, ID, Temp_Task_Index);
   -- Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
   --        ID);
   -- Put_Line(Image(Current_Task));
   -- Put("Set_Current_Master 設定した Master_ID -- ");
   -- Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
   -- New_Line;

end Set_Current_Master;

procedure Set_Original_Master(Master : in Master_Index_Link) is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                  Current_Task);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   end if;
   if Master /= null then
      Temp_Task_Index.Original_Master := Master;
   end if;
   -- Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
end Set_Original_Master;

procedure Set_Original_Master(Master : in Master_Index_Link;
                              ID : Task_ID) is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                  ID);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, ID, Temp_Task_Index);
   end if;
   Temp_Task_Index.Original_Master := Master;
   -- Task_Index_List.SET_NODE(All_Task_ID, ID, Temp_Task_Index);
end Set_Original_Master;

procedure Set_Completed_Master(Master : in Master_Index_Link) is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                  Current_Task);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   end if;
   if Master /= null then
      Temp_Task_Index.Completed_Master := Master;
   end if;
   -- Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
end Set_Completed_Master;

procedure Set_Completed_Master(Master : in Master_Index_Link;
                               ID : Task_ID) is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
```

```
                                                ID);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, ID, Temp_Task_Index);
   end if;
   if Master /= null then
      Temp_Task_Index.Completed_Master := Master;
   end if;
   -- Task_Index_List.SET_NODE(All_Task_ID, ID, Temp_Task_Index);
end Set_Completed_Master;

procedure Set_Asynchronous_State(Master : in Master_Index_Link;
                                 State : in Async_Type;
                                 Trigger : in Master_Index_Link) is
begin
   if Debug_Mode then
      Put("Set_Asynchronous_State -- ");
      Put_String(Get_Master_Name(Master));
      New_Line;
      if State = ASYNC_ENTRY_CALL then
         Put_Line("ASYNC_ENTRY_CALL!");
      end if;
   end if;
   Master.Asyncstate.Status := State;
   Master.Asyncstate.Trigger_Master := Trigger;
end Set_Asynchronous_State;

function Get_Asynchronous_State(Master : Master_Index_Link)
                                return Async_Type is
begin
   return Master.Asyncstate.Status;
end Get_Asynchronous_State;

function Get_Asynchronous_Trigger(Master : Master_Index_Link)
                                  return Master_Index_Link is
begin
   return Master.Asyncstate.Trigger_Master;
end Get_Asynchronous_Trigger ;


procedure Set_Entry_Caller(Master : in Master_Index_Link) is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                  Current_Task);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   end if;
   if Master /= null then
      Temp_Task_Index.Entry_Caller_Master := Master;
   end if;
   -- Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
end Set_Entry_Caller;

function Get_Entry_Caller(ID: Task_ID) return Master_Index_Link is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, ID);
   if Temp_Task_Index = null then
      return null;
   end if;
   return Temp_Task_Index.Entry_Caller_Master;
end Get_Entry_Caller;


--       function Get_Current_Master return ID_OF_MASTER is
--   Temp_Task_Index : Task_Index;
--       begin
--   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, Current_Task);
-- -- Put_Line(Image(Current_Task));
-- -- Put("Get_Current_Master で取得した ID - ");
-- -- Int_IO.Put(Integer(Temp_Task_Index.Current_Master));
-- -- New_Line;
--   return Task_Index_List.SEARCH_NODE(All_Task_ID, Current_Task).Current_Master ;
--       end Get_Current_Master;

function Get_Current_Master(ID : Task_ID := Current_Task)
                            return Master_Index_Link is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, ID);
   if Temp_Task_Index = null then
      return null;
   end if;
   return Temp_Task_Index.Current_Master ;
end Get_Current_Master;

function Get_Original_Master return Master_Index_Link is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, Current_Task);
   if Temp_Task_Index = null then
      return null;
   end if;
   return Temp_Task_Index.Original_Master ;
end Get_Original_Master;

function Get_Original_Master(ID : Task_ID) return Master_Index_Link is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, ID);
   if Temp_Task_Index = null then
      return null;
   end if;
   return Temp_Task_Index.Original_Master ;
end Get_Original_Master;

function Get_Completed_Master(ID : Task_ID) return Master_Index_Link is
```

```
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, ID);
   if Temp_Task_Index = null then
      return null;
   end if;
   return Temp_Task_Index.Completed_Master;
end Get_Completed_Master;

--       function Get_Parent_Master return ID_OF_MASTER is
--       begin
--   return Master_ID_List(Task_Index_List.SEARCH_NODE(All_Task_ID, Current_Task).Current_Master).Parent_Master_ID;
--       end Get_Parent_Master;

--       function Get_Parent_Master(Master_ID : ID_OF_MASTER)
--   return ID_OF_MASTER is
--       begin
--   return Master_ID_List(Master_ID).Parent_Master_ID;
--       end Get_Parent_Master;

--       procedure Set_Running_Task(Master_ID : in ID_OF_MASTER;
--  Running_Task_ID : in Task_ID) is
--       begin
--          Master_ID_List(Master_ID).Running_Task_ID := Running_Task_ID;
--       end Set_Running_Task;

--       function Get_Running_Task(Master_ID : ID_OF_MASTER) return Task_ID is
--       begin
--          return Master_ID_List(Master_ID).Running_Task_ID;
--       end Get_Running_Task;

procedure Set_Arc(Source , Destination : in Master_Index_Link;
                  Arc : in Arc_Class) is
   procedure Add_Next is
      Nownode : Node_List_Link;
   begin
      if Source.Nextnodes = null then
         Source.Nextnodes := new Node_List'(null, Destination,
                                            Arc, Null_String_Buffer);
      else
         Nownode := Source.Nextnodes;
         if Nownode.Mynode = Destination then
            Nownode.Arcclass := Arc;
         else
            while Nownode.Next /= null loop
               Nownode := Nownode.Next;
               if Nownode.Mynode = Destination then
                  Nownode.Arcclass := Arc;
                  exit;
               end if;
            end loop;
            if Nownode.Next = null and Nownode.Mynode /= Destination then
               Nownode.Next := new Node_List'(null, Destination,
                                              Arc, Null_String_Buffer);
            end if;
         end if;
      end if;
   end Add_Next;

   procedure Add_Prev is
      Nownode : Node_List_Link;
   begin
      if Destination.Prevnodes = null then
         Destination.Prevnodes := new Node_List'(null, Source,
                                                 Arc, Null_String_Buffer);
      else
         Nownode := Destination.Prevnodes;
         if Nownode.Mynode = Source then
            Nownode.Arcclass := Arc;
         else
            while Nownode.Next /= null loop
               Nownode := Nownode.Next;
               if Nownode.Mynode = Source then
                  Nownode.Arcclass := Arc;
                  exit;
               end if;
            end loop;
            if Nownode.Next = null and Nownode.Mynode /= Source then
               Nownode.Next := new Node_List'(null, Source,
                                              Arc, Null_String_Buffer);
            end if;
         end if;
      end if;
   end Add_Prev;

begin
   if Source = null then
      if Debug_Mode then
         Put_Line("Set_Arc: warning: Source is null.");
      end if;
      return;
   end if;
   if Destination = null then
      if Debug_Mode then
         Put_Line("Set_Arc: warning: Destination is null.");
         Put_Line("(Source: " & Integer'IMAGE(Source.Master_ID) & " : "
                  & Arc_Class'IMAGE(Arc) & ")" );
      end if;
      return;
   end if;

   if Debug_Mode then
      Put("Set_Arc (" & Image(Current_Task) & ") : from ");
      Int_IO.Put(Source.Master_ID);
      --    Int_IO.Put(Integer(Source));
      Put(" to ");
      Int_IO.Put(Destination.Master_ID);
      --    Int_IO.Put(Integer(Destination));
      Put(" : ");
```

```
      Put(Arc_Class'IMAGE(Arc));
      New_Line;
   end if;
   Add_Next;
   Add_Prev;
   Set_Count := Set_Count + 1;
   -- if Set_Count = 28 then TWFG.Dump_Master_List; end if;
   Flush;
end Set_Arc;

procedure Remove_Arc(Source, Destination : Master_Index_Link) is
   procedure Remove_Next is
      Nownode : Node_List_Link;
   begin
      if Source.Nextnodes /= null then
         if Source.Nextnodes.Mynode = Destination then
            Source.Nextnodes := Source.Nextnodes.Next;
         else
            Nownode := Source.Nextnodes;
            while Nownode.Next /= null loop
               if Nownode.Next.Mynode = Destination then
                  Nownode.Next := Nownode.Next.Next;
                  exit;
               end if;
               Nownode := Nownode.Next;
            end loop;
         end if;
      end if;
   end Remove_Next;
   procedure Remove_Prev is
      Nownode : Node_List_Link;
   begin
      if Destination.Prevnodes /= null then
         if Destination.Prevnodes.Mynode = Source then
            Destination.Prevnodes := Destination.Prevnodes.Next;
         else
            Nownode := Destination.Prevnodes;
            while Nownode.Next /= null loop
               if Nownode.Next.Mynode = Source then
                  Nownode.Next := Nownode.Next.Next;
                  exit;
               end if;
               Nownode := Nownode.Next;
            end loop;
         end if;
      end if;
   end Remove_Prev;
begin
   if Debug_Mode and then Source /= null and then Destination /= null
   then
      Put("Remove_Arc : from ");
      Int_IO.Put(Source.Master_ID);
      Put(" to ");
      Int_IO.Put(Destination.Master_ID);
      New_Line;
   end if;

   Remove_Next;
   Remove_Prev;
end Remove_Arc;

procedure Remove_All_Out_Arc(Master : in Master_Index_Link) is
   procedure Remove_Prev(Destination : in Master_Index_Link) is
      Nownode : Node_List_Link;
   begin
      if Destination.Prevnodes /= null then
         if Destination.Prevnodes.Mynode = Master then
            Destination.Prevnodes := Destination.Prevnodes.Next;
         else
            Nownode := Destination.Prevnodes;
            while Nownode.Next /= null loop
               if Nownode.Next.Mynode = Master then
                  Nownode.Next := Nownode.Next.Next;
                  exit;
               end if;
               Nownode := Nownode.Next;
            end loop;
         end if;
      end if;
   end Remove_Prev;
   Nownode : Node_List_Link := Master.Nextnodes;
begin
   if Debug_Mode and then Master /= null then
      Put("Remove_All_Out_Arc : ");
      Int_IO.Put(Master.Master_ID);
      New_Line;
   end if;
   while Nownode /= null loop
      Remove_Prev(Nownode.Mynode);
      Nownode := Nownode.Next;
   end loop;
   Master.Nextnodes := null;
end Remove_All_Out_Arc;

procedure Remove_All_Out_Arc(Master : in Master_Index_Link;
                             Arc : in Arc_Class) is
   Nownode : Node_List_Link := Master.Nextnodes;
   Tmp_Nownode : Node_List_Link;
begin
   while Nownode /= null loop
      if Nownode.Arcclass = Arc then
         Tmp_Nownode := Nownode;
         Nownode := Nownode.Next;
         Remove_Arc(Master, Tmp_Nownode.Mynode);
      else
         Nownode := Nownode.Next;
      end if;
   end loop;
end Remove_All_Out_Arc;
```

```
function Search_Protected_Object(U_Name : STRING_BUFFER) return Master_Index_Link is
   Nownode : Node_List_Link := Protected_Object_List;
begin
   while Nownode /= null loop
      if Nownode.Mynode.Master_Class = PROTECTED_OBJECTS and then
        Nownode.Mynode.Master_Name = U_Name then
         return Nownode.Mynode;
      end if;
      Nownode := Nownode.Next;
   end loop;
   return null;
end Search_Protected_Object;

function Get_An_Out_Arc_Destination(Source : Master_Index_Link) return Master_Index_Link is
begin
   if Source.Nextnodes /= null then
      return Source.Nextnodes.Mynode;
   else
      return null;
   end if;
end Get_An_Out_Arc_Destination;

procedure Set_Master_Name(Master : in Master_Index_Link;
                          U_Name : in STRING_BUFFER) is
begin
   Master.Master_Name := U_Name;
end Set_Master_Name;

function Get_Master_Name(Master : in Master_Index_Link)
                         return STRING_BUFFER is
begin
   return Master.Master_Name;
end Get_Master_Name;


--       procedure Set_Server_ID(ID : in Task_ID; T_Name : in STRING_BUFFER) is
--   Temp_Task_Index : Task_Index;
--       begin
--   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, ID);
--   Temp_Task_Index.Server_ID := Search_Server_ID(T_Name);
--   Task_Index_List.SET_NODE(All_Task_ID, ID, Temp_Task_Index);
--       end Set_Server_ID;

--       function Get_Server_ID(ID : Task_ID) return Natural is
--       begin
--   return Task_Index_List.SEARCH_NODE(All_Task_ID, ID).Server_ID ;
--       end Get_Server_ID;

procedure Set_Task_Status(ID : in Task_ID;
                          Status : in Status_Of_Task) is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, ID);
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, ID, Temp_Task_Index);
   end if;
   Temp_Task_Index.Task_Status := Status;
   -- Task_Index_List.SET_NODE(All_Task_ID, ID, Temp_Task_Index);
end Set_Task_Status;

function Get_Task_Status(ID : Task_ID) return Status_Of_Task is
   Temp_Task_Index : Task_Index_Link;
begin
   Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, ID);
   if Temp_Task_Index = null then
      return ELABORATING;
   end if;
   return Temp_Task_Index.Task_Status;
end Get_Task_Status;

procedure Set_Calling_Entry(Object_Task : in Task_ID;
                            E_Name : in STRING_BUFFER) is
   Tmp_Master : Master_Index_Link;
begin
   -- Put(Integer(Task_Index_List.SEARCH_NODE(All_Task_ID, Object_Task).Server_ID));
   -- New_Line;
   --   Master_ID_List(Get_Current_Master(Current_Task)).Calling_Entry :=
   --     Search_Entry_ID(Task_Index_List.SEARCH_NODE(All_Task_ID, Object_Task).Server_ID, E_Name);
   Tmp_Master := Get_Current_Master(Current_Task);
   if Tmp_Master /= null then
      Tmp_Master.Calling_Entry := E_Name;
   end if;
exception
   when Constraint_Error =>
      Put_Line("!! constraint error raised in Set_Calling_Entry !!");
   when Program_Error =>
      Put_Line("!! program error raised in Set_Calling_Entry !!");
   when Storage_Error =>
      Put_Line("!! storage error raised in Set_Calling_Entry !!");
   when Tasking_Error =>
      Put_Line("!! tasking error raised in Set_Calling_Entry !!");
end Set_Calling_Entry;

procedure Set_Calling_Entry(E_Name : in STRING_BUFFER;
                            Master : in Master_Index_Link) is
begin
   if Master /= null then
      Master.Calling_Entry := E_Name;
   end if;
end Set_Calling_Entry;

function Get_Calling_Entry(Master : Master_Index_Link) return String_Buffer is
begin
   if Master /= null then
      return Master.Calling_Entry;
   end if;
   return Null_String_Buffer;
```

```
end Get_Calling_Entry;

function Get_Arc_Class(Source, Destination : Master_Index_Link)
                       return ARC_CLASS is
   Nownode : Node_List_Link := Source.Nextnodes;
begin
   while Nownode /= null loop
      if Nownode.Mynode = Destination then
         return Nownode.Arcclass;
      end if;
      Nownode := Nownode.Next;
   end loop;
   return NO_ARC;
end Get_Arc_Class;

function Get_Master_Class(Master : Master_Index_Link)
                          return UNIT_CLASS is
begin
   if Master /= null then
      return Master.Master_Class;
   end if;
   return N_U_L_L;
end Get_Master_Class;

procedure Remove_Node(Master : in Master_Index_Link) is
   Top_Nodelist, Tmp_Nodelist : Node_List_Link := null;
   Nownode : Node_List_Link := Master.Nextnodes;
   Tmp_Nownode : Node_List_Link;
begin
   while Nownode /= null loop
      if Nownode.Arcclass = ACCEPTANCE_WAITING_ARC then
         if Top_Nodelist = null then
            Top_Nodelist := new Node_List'(null, Nownode.Mynode,
                                           NO_ARC, Null_String_Buffer);
            Nownode := Top_Nodelist;
         else
            Nownode.Next := new Node_List'(null, Nownode.Mynode,
                                           NO_ARC, Null_String_Buffer);
            Nownode := Nownode.Next;
         end if;
      end if;
      Remove_Arc(Nownode.Mynode, Master);
      Nownode := Nownode.Next;
   end loop;
   Remove_All_Out_Arc(Master);
   Nownode := Top_Nodelist;
   while Nownode /= null loop
      Tmp_Nownode := Nownode.Next;
      while Tmp_Nownode /= null loop
         TWFG.CHECK_CIRCULAR_DEADLOCKS(Nownode.Mynode,
                                       Tmp_Nownode.Mynode);
         Tmp_Nownode := Tmp_Nownode.Next;
      end loop;
      Nownode := Nownode.Next;
   end loop;
   Master.Master_Class := COMPLETED;


   --   for I in ID_OF_MASTER loop
   --      if AM(I, MASTER_ID) = ACCEPTANCE_WAITING_ARC then
   --         NUMBER := NUMBER + 1;
   --         ACC_TASK(NUMBER) := I;
   --      end if;
   --      AM(Master_ID, I) := NO_ARC;
   --      AM(I, Master_ID) := NO_ARC;
   --   end loop;
   --   for I in 1..NUMBER
   --   loop
   --              for J in ID_OF_MASTER
   --              loop
   --         if AM(ACC_TASK(I), J) /= NO_ARC then
   --    TWFG.CHECK_CIRCULAR_DEADLOCKS(I, J);
   --         end if;
   --             end loop;
   --   end loop;
   -- -- Master_ID_List(Master_ID) := (N_U_L_L, Null_Master_ID,
   -- --       Null_String_Buffer, NOT_EXIST,
   -- --       Null_Entry_ID);
   --         Master.Master_Class := COMPLETED;
end Remove_Node;

procedure Remove_All_Out_Node(Master : in Master_Index_Link;
                              U_Class : in UNIT_CLASS) is
   Nownode : Node_List_Link := Master.Nextnodes;
   Tmp_Nownode : Node_List_Link;
begin
   while Nownode /= null loop
      if Nownode.Mynode.Master_Class = U_Class then
         Tmp_Nownode := Nownode;
         Nownode := Nownode.Next;
         Remove_Node(Tmp_Nownode.Mynode);
      else
         Nownode := Nownode.Next;
      end if;
   end loop;
   --   for I in ID_OF_MASTER loop
   --      if AM(Master_ID, I) /= NO_ARC and then
   --        Master_ID_List(Master_ID).Master_Class = U_Class then
   --         Remove_Node(I);
   --      end if;
   --   end loop;
end Remove_All_Out_Node;

function Check_CDT(Server_Task : Task_ID; Task_CDT : CDT; Entry_Name : in String_Buffer) return Boolean is
begin
   for I in Task_CDT'RANGE loop
      if Task_CDT(I).Callee_Task = Server_Task and
        Get_String_Buffer(Task_CDT(I).Entry_Name(1..Task_CDT(I).Last)) =
        Entry_Name then
```

```
         return True;
      end if;
   end loop;
   return False;
end Check_CDT;

function Check_CDT_For_TT(Server_Unit_ID : Unit_ID; Task_CDT_For_TT : CDT_For_TT; Entry_Name : in String_Buffer) return Boolean is
begin
   for I in Task_CDT_For_TT'RANGE loop
      if Task_CDT_For_TT(I).Callee_Unit_ID = Server_Unit_ID and
        Get_String_Buffer(Task_CDT_For_TT(I).Entry_Name
                          (1..Task_CDT_For_TT(I).Last)) =
        Entry_Name then
         return True;
      end if;
   end loop;
   return False;
end Check_CDT_For_TT;

procedure Set_All_Acceptance_Arc(Entry_Name : in STRING_BUFFER) is
   -- Server_ID, Entry_ID, Client_ID : Natural;
   -- MARKS   : array(ID_OF_MASTER) of BOOLEAN := (others => FALSE);
   Top_Marks, Now_Marks : Node_List_Link := null;
   Iter : Task_Index_List.Node := Task_Index_List.Null_node;
   Temp_Task_Index : Task_Index_Link := Task_Index_List.SEARCH_NODE
     (All_Task_ID, Current_Task);
begin
   if Temp_Task_Index = null then
      Temp_Task_Index := new Task_Index;
      Task_Index_List.SET_NODE(All_Task_ID, Current_Task, Temp_Task_Index);
   end if;
   if not Task_Index_List.IsNULL(All_Task_ID) then
      Iter := All_Task_ID.all;
      while not Task_Index_List."="(Iter, Task_Index_List.Null_Node) loop
         if (Check_CDT(Current_Task, Iter.Object.Task_CDT, Entry_Name) or else
               (Temp_Task_Index.Source_Unit_ID /= 0 and then
                  Check_CDT_For_TT(Temp_Task_Index.Source_Unit_ID,
                                   Iter.Object.Task_CDT_For_TT, Entry_Name)))
           and then Iter.Object.Task_Status /= BLOCK_COMPLETED
         then
            --  MARKS(TWFG.Get_Original_Master(Iter.Index)) := True;
            if Top_Marks = null then
               Top_Marks := new Node_List'
                 (null, TWFG.Get_Original_Master(Iter.Index), NO_ARC,
                  Null_String_Buffer);
               Now_Marks := Top_Marks;
            else
               Now_Marks.Next := new Node_List'
                 (null, TWFG.Get_Original_Master(Iter.Index), NO_ARC,
                  Null_String_Buffer);
               Now_Marks := Now_Marks.Next;
            end if;
            TWFG.Set_Arc(TWFG.Get_Current_Master(Current_Task),
                         TWFG.Get_Original_Master(Iter.Index),
                         ACCEPTANCE_WAITING_ARC);
            TWFG.Set_Arc_Label(TWFG.Get_Current_Master(Current_Task),
                               TWFG.Get_Original_Master(Iter.Index),
                               Entry_Name);
         end if;
         Iter := Task_Index_List.Get_Next_Node(Iter);
      end loop;
      Get_Current_Master(Current_Task).Calling_Entry :=
        Entry_Name;
      --      for I in ID_OF_MASTER loop
      --         if MARKS(I) then
      Now_Marks := Top_Marks;
      while Now_Marks /= null loop
         TWFG.CHECK_CIRCULAR_DEADLOCKS(Get_Current_Master(Current_Task),
                                       Now_Marks.Mynode);
         Now_Marks := Now_Marks.Next;
      end loop;
      --       end if;
      --    end loop;
   end if;
end Set_All_Acceptance_Arc;

procedure Set_All_Acceptance_Arc(Master : Master_Index_Link) is
   -- Server_Master_ID : ID_OF_MASTER;
   Server_Master : Master_Index_Link;
   Current_Task_Index : Task_Index_Link := Task_Index_List.SEARCH_NODE
     (All_Task_ID, Current_Task);
   Current_Node : Task_Index_List.NODE;
   Current_Node_Master : Master_Index_Link;
begin

   if not Task_Index_List.IsNULL(All_Task_ID) then
      Current_Node := All_Task_ID.all;

      if Current_Task_Index = null then
         Current_Task_Index := new Task_Index;
         Task_Index_List.SET_NODE(All_Task_ID, Current_Task,
                                  Current_Task_Index);
      end if;

      loop
         Server_Master :=
           TWFG.Get_Original_Master(Current_Node.Index);
         --       Server_Master_ID :=
         -- TWFG.Get_Original_Master(Current_Node.Index);
         --       Server_ID := Current_Node.Object.Server_ID;
         --       Put("Server_ID - ");
         --       Int_IO.Put(Integer(Server_ID));
         --       New_Line;
         --     if Server_ID /= 0 then
         --     for I in 1..Server_Entry_Count(Server_ID) loop
         --     Server_Master_ID :=
         --       TWFG.Get_Original_Master(Current_Node.Index);
         Current_Node_Master := Get_Current_Master(Current_Node.Index);
         if Current_Node_Master /= null then
```

```
            if Check_CDT
              (Current_Node.Index, Current_Task_Index.Task_CDT,
               Current_Node_Master.Calling_Entry)
              and then TWFG.Get_Status(Server_Master) = ACCEPTANCE
            then
               TWFG.Set_Arc(TWFG.Get_Original_Master(Current_Node.Index),
                            Master,
                            ACCEPTANCE_WAITING_ARC);
               TWFG.Set_Arc_Label
                 (TWFG.Get_Original_Master(Current_Node.Index),
                  Master,
                  Get_Current_Master
                    (Current_Node.Index).Calling_Entry);
            end if;
         end if;
         Current_Node := Task_Index_List.GET_NEXT_NODE(Current_Node);
         exit when Task_Index_List."="(Current_Node, Task_Index_List.Null_Node);
      end loop;
   end if;
end Set_All_Acceptance_Arc;

procedure Set_All_Finalization_Arc(Master : Master_Index_Link) is
   Current_Node : Task_Index_List.NODE;
begin
   if not Task_Index_List.IsNULL(All_Task_ID) then
      Current_Node := All_Task_ID.all;
      loop
         if Get_Original_Master(Current_Node.Index).Parent_Master
           = Master then
            Set_Arc(Master,
                    Get_Original_Master(Current_Node.Index),
                    --  Get_Parent_Master(Get_Current_Master(Current_Node.Index)),
                    FINALIZATION_WAITING_ARC);
            TWFG.Check_Circular_Deadlocks
              (Master,
               Get_Original_Master(Current_Node.Index));
             --  Get_Parent_Master(Get_Current_Master(Current_Node.Index)));
         end if;
         Current_Node := Task_Index_List.GET_NEXT_NODE(Current_Node);
         exit when Task_Index_List."="(Current_Node, Task_Index_List.Null_Node);
      end loop;
   end if;
end Set_All_Finalization_Arc;

procedure Set_Arc_Label(Source , Destination : in Master_Index_Link;
                        Arc_Label : in STRING_BUFFER) is
   Nownode : Node_List_Link;
begin
   if Source = null then
      if Debug_Mode then
         Put_Line("Set_Arc_Label: Warinig: source is null.");
      end if;
      return;
   end if;
   Nownode := Source.Nextnodes;
   if Destination = null then
      if Debug_Mode then
         Put_Line("Set_Arc_Label: Warinig: destination is null.");
      end if;
      return;
   end if;
   while Nownode /= null loop
      if Nownode.Mynode = Destination then
         Nownode.Arclabel := Arc_Label;
         exit;
      end if;
      Nownode := Nownode.Next;
   end loop;
   Nownode := Destination.Prevnodes;
   while Nownode /= null loop
      if Nownode.Mynode = Source then
         Nownode.Arclabel := Arc_Label;
         exit;
      end if;
      Nownode := Nownode.Next;
   end loop;
   -- AL(Source, Destination) := Arc_Label;
end Set_Arc_Label;

function Get_Arc_Label(Source , Destination : Master_Index_Link)
                       return STRING_BUFFER is
   Nownode : Node_List_Link := Source.Nextnodes;
begin
   while Nownode /= null loop
      if Nownode.Mynode = Destination then
         return Nownode.Arclabel;
      end if;
      Nownode := Nownode.Next;
   end loop;
   return Null_String_Buffer;
   -- return AL(Source, Destination);
end Get_Arc_Label;


procedure Remove_Task is
begin
   Task_Index_List.DELETE_NODE(All_Task_ID, Current_Task);
end Remove_Task;

procedure Set_Status(Master_Status : in Status_Of_Master) is
begin
   TWFG.Get_Current_Master(Current_Task).Master_Status
     := Master_Status;
end Set_Status;

procedure Set_Status(Master_Status : in Status_Of_Master;
                     Master : in Master_Index_Link) is
begin
   Master.Master_Status := Master_Status;
```

```
end Set_Status;

function Get_Status(Master : Master_Index_Link)
                  return Status_Of_Master is
begin
   return Master.Master_Status;
end Get_Status;

procedure Set_Completion_Arc is
   Current_Node : Task_Index_List.NODE;
   Current_Master, Parent_Master, Object_Master : Master_Index_Link;
   -- MARKS   : array(ID_OF_MASTER) of BOOLEAN := (others => FALSE);
   Top_Marks, Now_Marks : Node_List_Link := null;
begin
   Current_Master := TWFG.Get_Current_Master(Current_Task);
   Parent_Master := Current_Master.Parent_Master;
   if not Task_Index_List.IsNULL(All_Task_ID) then
      Current_Node := All_Task_ID.all;
      loop
         Object_Master := TWFG.Get_Original_Master(Current_Node.Index);
         if Object_Master.Parent_Master = Parent_Master
           and then TWFG.Get_Status(Object_Master) /=
           TERMINATION_SELECTION then
            TWFG.Set_Arc(Current_Master, Object_Master,
                         COMPLETION_WAITING_ARC);
            --MARKS(Object_Master) := True;
            if Top_Marks = null then
               Top_Marks := new Node_List'(null, Object_Master,
                                           NO_ARC, Null_String_Buffer);
               Now_Marks := Top_Marks;
            else
               Now_Marks.Next := new Node_List'
                 (null, Object_Master,
                  NO_ARC, Null_String_Buffer);
               Now_Marks := Now_Marks.Next;
            end if;
         end if;
         Current_Node := Task_Index_List.GET_NEXT_NODE(Current_Node);
         exit when Task_Index_List."="(Current_Node, Task_Index_List.Null_Node);
      end loop;
      Now_Marks := Top_Marks;
      while Now_Marks /= null loop
         CHECK_CIRCULAR_DEADLOCKS(Current_Master, Now_Marks.Mynode);
         Now_Marks := Now_Marks.Next;
      end loop;
      --             for I in ID_OF_MASTER
      --             loop
      --          if MARKS(I) then
      --    CHECK_CIRCULAR_DEADLOCKS(Current_Master, I);
      --               end if;
      --             end loop;
   end if;
end Set_Completion_Arc;

procedure Set_Reverse_Completion_Arc is
   Current_Node : Task_Index_List.NODE;
   Current_Master, Parent_Master, Object_Master : Master_Index_Link;
begin
   Current_Master := TWFG.Get_Current_Master(Current_Task);
   Parent_Master := Current_Master.Parent_Master;
   if not Task_Index_List.isNULL(All_Task_ID) then
      Current_Node := All_Task_ID.all;
      loop
         Object_Master := TWFG.Get_Original_Master(Current_Node.Index);
         if Object_Master /= null
           and then Object_Master.Parent_Master = Parent_Master
           and then TWFG.Get_Status(Object_Master)
           = TERMINATION_SELECTION then
            TWFG.Set_Arc(Object_Master, Current_Master,
                         COMPLETION_WAITING_ARC);
         end if;
         Current_Node := Task_Index_List.GET_NEXT_NODE(Current_Node);
         exit when Task_Index_List."="(Current_Node, Task_Index_List.Null_Node);
      end loop;
   end if;
end Set_Reverse_Completion_Arc;

function Is_Entry_Call(Master : Master_Index_Link;
                       Entry_Name : STRING_BUFFER) return Boolean is
   Nownode : Node_List_Link := Master.Prevnodes;
begin
   while Nownode /= null loop
      if Nownode.Arcclass = ENTRY_CALLING_ARC and then
        Nownode.Arclabel = Entry_Name then
         return True;
      end if;
      Nownode := Nownode.Next;
   end loop;
   --  for I in ID_OF_MASTER loop
   --     if AM(I, Master_ID) = ENTRY_CALLING_ARC and then
   --       AL(I, Master_ID) = Entry_Name then
   --        return True;
   --     end if;
   --  end loop;
   return False;
end Is_Entry_Call;

procedure Dump_Master_List is
   Now_Node : Node_List_Link := Master_List;
begin
   while Now_Node /= null loop
      -- for I in ID_OF_MASTER loop
      --    if Master_ID_List(I).Master_Class /= N_U_L_L then
      if Now_Node.Mynode.Master_Class /= N_U_L_L then
         --          Put("ID : ");
         --          Int_IO.Put(Integer(I));
         Put("  Master_Name : ");
         --       Put_String(Master_ID_List(I).Master_Name);
         Put_String(Now_Node.Mynode.Master_Name);
```

```
         if Now_Node.Mynode.Parent_Master /= null then
            Put(" Parent_Master_Name : ");
            Put_String(Now_Node.Mynode.Parent_Master.Master_Name);
         end if;
         New_Line;
      end if;
      Now_Node := Now_Node.Next;
   end loop;
end Dump_Master_List;


procedure EXIST_A_PATH(
                      START: in Master_Index_Link;
                      TERMINAL: in Master_Index_Link;
                      Result : out boolean) is
   Nownode : Node_List_Link;
   procedure Process_Node(Node : in Master_Index_Link;
                         Seq : in Integer) is
      Tmp_Nownode : Node_List_Link := Node.Prevnodes;
   begin
      while Tmp_Nownode /= null loop
         if Tmp_Nownode.Mynode.Seq = 0 or Tmp_Nownode.Mynode.Seq > Seq
         then
            Tmp_Nownode.Mynode.Seq := Seq;
            if Tmp_Nownode.Mynode /= TERMINAL then
               Process_Node(Tmp_Nownode.Mynode, Seq + 1);
            end if;
         end if;
         Tmp_Nownode := Tmp_Nownode.Next;
      end loop;
   end Process_Node;

begin -- EXIST_A_PATH
   -- フラグのクリア
   Nownode := Master_List;
   while Nownode /= null loop
      Nownode.Mynode.Seq := 0;
      Nownode := Nownode.Next;
   end loop;
   -- 再帰で到達可能な範囲を調べる
   -- TERMINAL.Seq := -1;
   Process_Node(TERMINAL, 1);
   if Debug_Mode then
      Put_Line("exist_a_path from " & Integer'IMAGE(START.Master_ID) &
               " to " & Integer'IMAGE(TERMINAL.Master_ID));
      Nownode := Master_List;
      while Nownode /= null loop
         Put_Line("ID: " & Integer'IMAGE(Nownode.Mynode.Master_ID)
                  & " Seq: "
                  & Integer'IMAGE(Nownode.Mynode.Seq));
         Nownode := Nownode.Next;
      end loop;
   end if;

   Result := (START.Seq /= 0);
   --  for I in ID_OF_MASTER
   --  loop
   --     for J in ID_OF_MASTER
   --     loop
   --        --  case AM(I, J) is
   --        --  when NO_ARC => null;
   --        --  when ACTIVATION_WAITING_ARC => PUT(I, 3); PUT(J, 3); PUT(":Act  ");
   --        --  when ACCEPTANCE_WAITING_ARC => PUT(I, 3); PUT(J, 3); PUT(":Acc  ");
   --        --  when ENTRY_CALLING_ARC => PUT(I, 3); PUT(J, 3); PUT(":E  ");
   --        --  when DEPENDENCE_ARC => PUT(I, 3); PUT(J, 3); PUT(":D  ");
   --        --  when TERMINATION_WAITING_ARC => PUT(I, 3); PUT(J, 3); PUT(":T  ");
   --        --  end case;

   --        if AM(I, J) /= NO_ARC then
   --   PM(I, J) := 1;
   --        end if;
   --     end loop;
   --  end loop;
   --  for I in ID_OF_MASTER
   --  loop
   --     for J in ID_OF_MASTER
   --     loop
   --        if PM(J, I) = 1 then
   --   for K in ID_OF_MASTER
   --     loop
   --        if PM(I, K) = 1 then
   --   PM(J, K) := 1;
   --        end if;
   --     end loop;
   --        end if;
   --     end loop;
   --  end loop;
   --  if PM(START, TERMINAL) = 1 then
   --     Result := TRUE;
   --  else
   --     Result := FALSE;
   --  end if;

end EXIST_A_PATH;

-- 必ず EXIST_A_PATH の後に実行する
procedure FIND_A_PATH(START: in Master_Index_Link;
                      TERMINAL: in Master_Index_Link;
                      PATH: out Node_List_Link) is

   Top_Path, Now_Path : Node_List_Link := null;
   -- PATHS  : array(ID_OF_MASTER) of PATH_ARRAY;
   -- LEN    : NATURAL range 0..Maximum_Master := 0;
   Seq : Integer;
   Nownode : Node_List_Link;
begin -- FIND_A_PATH

   Top_Path := new Node_List'
     (null, START, NO_ARC, Null_String_Buffer);
```

```
   Now_Path := Top_Path;
   Seq := START.Seq;
   --   PATHS(START).NODE(1) := START;
   --   PATHS(START).LENGTH  := 1;
   --   LEN := 0;

   -- タスクが自身のエントリを呼び出す場合 -- by T.E
   PATH := Top_Path;  -- by T.E
   Find_A_Path_Loop:
   loop
      Nownode := Now_Path.Mynode.Nextnodes;
      loop
         if Nownode = null then
            Put_Line("No path! from:" & Integer'IMAGE(START.Master_ID)
                     & " to: " & Integer'IMAGE(TERMINAL.Master_ID) &
                       " at: " & Integer'IMAGE(Now_Path.Mynode.Master_ID));
            raise Program_Error;
         end if;
         if Nownode.Mynode = TERMINAL then
            Now_Path.Next := new Node_List'
              (null, TERMINAL, Nownode.Arcclass, Null_String_Buffer);
            exit Find_A_Path_Loop;
         end if;
         if Nownode.Mynode.Seq = Seq - 1 and Nownode.Mynode.Seq > 0 then
            Now_Path.Next := new Node_List'
              (null, Nownode.Mynode, Nownode.Arcclass,
               Null_String_Buffer);
            Now_Path := Now_Path.Next;
            Seq := Seq - 1;
            exit;
         end if;
         Nownode := Nownode.Next;
      end loop;

      PATH := Top_Path;
   end loop Find_A_Path_Loop;
   --      for I in ID_OF_MASTER
   --      loop
   --         if PATHS(I).LENGTH = LEN then
   --    if I = TERMINAL then
   --       PATH.NODE   := PATHS(I).NODE;
   --       PATH.LENGTH := PATHS(I).LENGTH;
   --       return;
   --    else
   --       for J in ID_OF_MASTER
   --       loop
   --  if AM(PATHS(I).NODE(LEN), J) /= NO_ARC then
   --     if ( PATHS(J).LENGTH > LEN + 1 ) or
   --       ( PATHS(J).LENGTH = 0 ) then
   --        PATHS(J).NODE := PATHS(I).NODE;
   --        PATHS(J).NODE(LEN+1) := J;
   --        PATHS(J).LENGTH := LEN + 1;
   --     end if;
   --  end if;
   --       end loop;
   --    end if;
   --         end if;
   --      end loop;
   --   end loop;

end FIND_A_PATH;

procedure SATISFY_CONDITION_OF_DEADLOCK(
                                        PATH: in Node_List_Link;
                                        Result : out Boolean;
                                        Async_Result : out Boolean;
                                        Async_Depend : out Node_List_Link)
is


   SUB_PATH : Node_List_Link;
   Head     : Master_Index_Link;
   Tail     : Master_Index_Link;
   Arcclass : ARC_CLASS;
   FLAG     : BOOLEAN := TRUE;
   Async_Flag : Boolean := False;
   Nownode, Tmp_Nownode, Sub_Nownode : Node_List_Link;
   Tmp_Result1, Tmp_Result2 : Boolean;
   Top_Depend, Now_Depend : Node_List_Link := null;
begin
   if PATH = null then
      Result := False;
      return;
   end if;
   Nownode := PATH;
   while Nownode /= null loop
      Tail := Nownode.Mynode;
      if Nownode.Next = null then
         Head := PATH.Mynode;
         Arcclass := Get_Arc_Class(Tail, Head);
      else
         Head := Nownode.Next.Mynode;
         Arcclass := Nownode.Next.Arcclass;
      end if;
      if Debug_Mode then
         Put_Line("Tail: " & Integer'IMAGE(Tail.Master_ID) &
                  " Head: " & Integer'IMAGE(Head.Master_ID) &
                  " arc_class: " & ARC_CLASS'IMAGE(Nownode.Arcclass));
      end if;
      if Get_Asynchronous_State(Head) = ASYNC_DELAY_STMT then
         Result := False;
         return;
      end if;
      if Get_Asynchronous_State(HEAD) = ASYNC_ENTRY_CALL then
         if Debug_Mode then
            Put("!!! ASYNC_ENTRY_CALL is found! -- ");
            Put_String(Get_Master_Name(HEAD));
            New_Line;
         end if;
```

```
      Async_Flag := True;
      if Top_Depend = null then
         Top_Depend := new Node_List'
           (null, Get_Asynchronous_Trigger(Head), NO_ARC,
            Null_String_Buffer);
         Now_Depend := Top_Depend;
      else
         Now_Depend.Next := new Node_List'
           (null, Get_Asynchronous_Trigger(Head), NO_ARC,
            Null_String_Buffer);
         Now_Depend := Now_Depend.Next;
      end if;
   end if;
   if Arcclass = ACCEPTANCE_WAITING_ARC
     or Arcclass = COMPLETION_WAITING_ARC then
      Tmp_Nownode := Tail.Nextnodes;
      while Tmp_Nownode /= null loop
         if Tmp_Nownode.Arcclass = ACCEPTANCE_WAITING_ARC then
            Tmp_Result2 := False;
            Exist_A_Path(Tmp_Nownode.Mynode, Tail, Tmp_Result1);
            if not Tmp_Result1 then
               Exist_A_Path(Tmp_Nownode.Mynode, Tmp_Nownode.Mynode,
                            Tmp_Result2);
               if not Tmp_Result2 then
                  Result := False;
                  return;
               end if;
            end if;
            if Tmp_Result1 then
               Find_A_Path(Tmp_Nownode.Mynode, Tail, SUB_PATH);
               Sub_Nownode := SUB_PATH;
               while Sub_Nownode /= null and then
                 Sub_Nownode.Next /= null loop
                  if Sub_Nownode.Arcclass = ACCEPTANCE_WAITING_ARC
                  then
                     Result := False;
                     return;
                  end if;
                  Sub_Nownode := Sub_Nownode.Next;
               end loop;
            end if;
            Exist_A_Path(Tmp_Nownode.Mynode, Tmp_Nownode.Mynode,
                         Tmp_Result2);
            if Tmp_Result2 then
               Find_A_Path(Tmp_Nownode.Mynode, Tmp_Nownode.Mynode,
                           SUB_PATH);
               Sub_Nownode := SUB_PATH;
               while Sub_Nownode /= null and then
                 Sub_Nownode.Next /= null loop
                  if Sub_Nownode.Arcclass = ACCEPTANCE_WAITING_ARC
                  then
                     Result := False;
                     return;
                  end if;
                  Sub_Nownode := Sub_Nownode.Next;
               end loop;
            end if;
         end if;
         Tmp_Nownode := Tmp_Nownode.Next;
      end loop;
   end if;
   Nownode := Nownode.Next;
end loop;
Result := True;
Async_Result := Async_Flag;
Async_Depend := Top_Depend;

--   for I in 1..PATH.LENGTH
--   loop
--      if I = PATH.LENGTH then
--         HEAD := PATH.NODE(1);
--         TAIL := PATH.NODE(PATH.LENGTH);
--      else
--         HEAD := PATH.NODE(I+1);
--         TAIL := PATH.NODE(I);
--      end if;
--      if Get_Asynchronous_State(HEAD) = ASYNC_DELAY_STMT then
--         Result := False;
--         return;
--      end if;
--      if Get_Asynchronous_State(HEAD) = ASYNC_ENTRY_CALL then
--         if Debug_Mode then
--    Put("!!! ASYNC_ENTRY_CALL is found! -- ");
--    Put_String(Get_Master_Name(HEAD));
--    New_Line;
--         end if;
--         Async_Flag := True;
--         Async_Deps(Get_Asynchronous_Trigger(HEAD)) := True;
--      end if;
--      if AM(TAIL, HEAD) = ACCEPTANCE_WAITING_ARC then
--         for J in ID_OF_MASTER
--   loop
--      if J /= HEAD then
--         if AM(TAIL, J) = ACCEPTANCE_WAITING_ARC then
--    if PM(J, TAIL) = 0 and PM(J, J) = 0 then
--       Result := FALSE;
--       return;
--    else
--       if PM(J, TAIL) /= 0 then
--  FIND_A_PATH(J, TAIL, SUB_PATH);
--  for K in 1..SUB_PATH.LENGTH-1
--  loop
--     if AM(PATH.NODE(K), PATH.NODE(K+1))
--       = ACCEPTANCE_WAITING_ARC then
--        Result := FALSE;
--        return;
--     end if;
--  end loop;
--       end if;
```

```
   --       if PM(J, J) /= 0 then
   --  FIND_A_PATH(J, J, SUB_PATH);
   --  for K in 1..SUB_PATH.LENGTH-1
   --  loop
   --     if AM(PATH.NODE(K), PATH.NODE(K+1))
   --       = ACCEPTANCE_WAITING_ARC then
   --        Result := FALSE;
   --        return;
   --     end if;
   --  end loop;
   --       end if;
   --    end if;
   --         end if;
   --      end if;
   --   end loop;
   --      end if;
   --   end loop;
   --   Result := TRUE;
   --   Async_Result := Async_Flag;
   --   Async_Depend := Async_Deps;
end SATISFY_CONDITION_OF_DEADLOCK;

procedure CHECK_CIRCULAR_DEADLOCKS(
                                   TAIL_MASTER: in Master_Index_Link;
                                   HEAD_MASTER: in Master_Index_Link
                                  ) is

   PATH        : Node_List_Link;
   Result : Boolean;
   HEAD, TAIL  : Master_Index_Link;
   Async_Result : Boolean;
   Async_Depend : Node_List_Link;
   Nownode : Node_List_Link;

begin -- CHECK_CIRCULAR_DEADLOCKS
   if Debug_Mode then
      Put("## CHECK_CIRCULAR_DEADLOCKS (HEAD --");
      Put_String(Get_Master_Name(HEAD_MASTER));
      Put("(" & Integer'IMAGE(HEAD_MASTER.Master_ID) & ")  TAIL --");
      Put_String(Get_Master_Name(TAIL_MASTER));
      Put_Line("(" & Integer'IMAGE(TAIL_MASTER.Master_ID) & ")) ## ");
   end if;
   Exist_A_Path(HEAD_MASTER, TAIL_MASTER, Result );
   if Result then
      FIND_A_PATH(HEAD_MASTER,
                  TAIL_MASTER,
                  PATH);
   else
      return;
   end if;
   SATISFY_CONDITION_OF_DEADLOCK(PATH, Result, Async_Result,
                                 Async_Depend);

   if Result then
      Put_Line("Tasking deadlock will occur in this program!");
      Put("* At ");
      Duration_IO.Put(Clock - Start_Time);
      Put_Line(" sec after execution start,");
      PUT_LINE("  there is such a situation as follows :");

      Nownode := PATH;
      while Nownode /= null loop
         Tail := Nownode.Mynode;
         if Nownode.Next = null then
            Head := PATH.Mynode;
         else
            Head := Nownode.Next.Mynode;
         end if;
         --              for L in 2..PATH.LENGTH+1
         --              loop
         --         if L /= PATH.LENGTH+1 then
         --    TAIL := PATH.NODE(L-1);
         --    HEAD := PATH.NODE(L);
         --         else
         --    TAIL := TAIL_MASTER;
         --    HEAD := HEAD_MASTER;
         --         end if;
         case TWFG.Get_Arc_Class(TAIL, HEAD) is
         when ACTIVATION_WAITING_ARC =>
            NEW_LINE;
            Put("*");
            Put_Master_Class(TAIL);
            Put_String(Get_Master_Name(TAIL));
            --                    N := NAME_OF(TASK_ID_OF(TAIL));
            --                    PUT(N.NAME(1..N.LENGTH));
            --                    if N.INDEX /= NULL_INDEX then
            --                      PUT("(");
            --                      PUT(N.INDEX, WIDTH_OF_INDEX);
            --                      PUT(")");
            --                    end if;
            PUT(" is waiting for activation completion of task ");
            --                    N := NAME_OF(TASK_ID_OF(HEAD));
            --                    PUT(N.NAME(1..N.LENGTH));
            --                    if N.INDEX /= NULL_INDEX then
            --                      PUT("(");
            --                      PUT(N.INDEX, WIDTH_OF_INDEX);
            --                      PUT(")");
            --                    end if;
            Put_String(Get_Master_Name(HEAD));
         when ACCEPTANCE_WAITING_ARC =>
            NEW_LINE;
            PUT("* task ");
            Put_String(Get_Master_Name(TAIL));
            --                    N := NAME_OF(WAIT_ENTRY(TASK_ID_OF(TAIL)).TASK_ID);
            --                    PUT(N.NAME(1..N.LENGTH));
            --                    if N.INDEX /= NULL_INDEX then
            --                      PUT("(");
            --                      PUT(N.INDEX, WIDTH_OF_INDEX);
            --                      PUT(")");
```

```
   --                   end if;
   PUT(" is waiting for accepting a call to its entry ");
   --                   N := NAME_OF(WAIT_ENTRY(TASK_ID_OF(TAIL)));
   --                   PUT(N.NAME(1..N.LENGTH));
   --                   if N.INDEX /= NULL_INDEX then
   --                     PUT("(");
   --                     PUT(N.INDEX, WIDTH_OF_INDEX);
   --                     PUT(")");
   --                   end if;
   --     Put_String(Server_Entry_Names(Get_Server_ID(Get_Running_Task(TAIL)), Get_Calling_Entry(TAIL)));
   Put_String(Get_Arc_Label(TAIL, HEAD));
   PUT(" from");
   Put_Master_Class(HEAD);
   --                   N := NAME_OF(TASK_ID_OF(HEAD));
   --                   PUT(N.NAME(1..N.LENGTH));
   --                   if N.INDEX /= NULL_INDEX then
   --                     PUT("(");
   --                     PUT(N.INDEX, WIDTH_OF_INDEX);
   --                     PUT(")");
   --                   end if;
   Put_String(Get_Master_Name(HEAD));
when ENTRY_CALLING_ARC =>
   NEW_LINE;
   PUT("*");
   Put_Master_Class(TAIL);
   --                   N := NAME_OF(TASK_ID_OF(TAIL));
   --                   PUT(N.NAME(1..N.LENGTH));
   --                   if N.INDEX /= NULL_INDEX then
   --                     PUT("(");
   --                     PUT(N.INDEX, WIDTH_OF_INDEX);
   --                     PUT(")");
   --                   end if;
   Put_String(Get_Master_Name(TAIL));
   PUT(" is calling entry ");
   --                   N := NAME_OF(CALLED_BY(TASK_ID_OF(TAIL)));
   --                   PUT(N.NAME(1..N.LENGTH));
   --                   if N.INDEX /= NULL_INDEX then
   --                     PUT("(");
   --                     PUT(N.INDEX, WIDTH_OF_INDEX);
   --                     PUT(")");
   --                   end if;
   --     Put_String(Server_Entry_Names(Get_Server_ID(Get_Running_Task(HEAD)), Get_Calling_Entry(TAIL)));
   Put_String(Get_Arc_Label(TAIL, HEAD));
   PUT(" of task ");
   --                   N := NAME_OF(CALLED_BY(TASK_ID_OF(TAIL)).TASK_ID);
   --                   PUT(N.NAME(1..N.LENGTH));
   --                   if N.INDEX /= NULL_INDEX then
   --                     PUT("(");
   --                     PUT(N.INDEX, WIDTH_OF_INDEX);
   --                     PUT(")");
   --                   end if;
   Put_String(Get_Master_Name(HEAD));
when FINALIZATION_WAITING_ARC =>
   NEW_LINE;
   Put("*");
   Put_Master_Class(TAIL);
   Put_String(Get_Master_Name(TAIL));
   --                   case Get_Master_Class(TAIL) is
   --                   when TASKS =>
   --                        SET_COL(TO);
   --                        PUT("* task ");
   --                        N := NAME_OF(TASK_ID_OF(TAIL));
   --                        PUT(N.NAME(1..N.LENGTH));
   --                        if N.INDEX /= NULL_INDEX then
   --                          PUT("(");
   --                          PUT(N.INDEX, WIDTH_OF_INDEX);
   --                          PUT(")");
   --                        end if;
   --                   when SUBPROGRAMS =>
   --                        SET_COL(TO);
   --                        PUT("* subprogram ");
   --                        SN := NAME_OF(TAIL);
   --                        PUT(SN.NAME(1..SN.LENGTH));
   --                   when BLOCKS =>
   --                        SET_COL(TO);
   --                        PUT("* block ");
   --                        SN := NAME_OF(TAIL);
   --                        PUT(SN.NAME(1..SN.LENGTH));
   --                   when others =>
   --                        null;
   --                   end case;
   case Get_Master_Class(HEAD) is
   when TASKS =>
      PUT(" created task ");
      --                        SN := NAME_OF(HEAD);
      --                        PUT(SN.NAME(1..SN.LENGTH));
      Put_String(Get_Master_Name(HEAD));
      PUT(" and is waiting for termination of task ");
      Put_String(Get_Master_Name(HEAD));
      --                        PUT(SN.NAME(1..SN.LENGTH));
   when PROCEDURES|FUNCTIONS|MAIN_PROCEDURE =>
      PUT(" is calling subprogram ");
      --                        SN := NAME_OF(HEAD);
      --                        PUT(SN.NAME(1..SN.LENGTH));
      Put_String(Get_Master_Name(HEAD));
   when BLOCKS =>
      PUT(" is executing block ");
      Put_String(Get_Master_Name(HEAD));
      --                        SN := NAME_OF(HEAD);
      --                        PUT(SN.NAME(1..SN.LENGTH));
   when PROTECTED_OBJECTS =>
      Put(" is waiting protected object ");
      Put_String(Get_Master_Name(HEAD));
   when PROTECTED_PROCEDURES | PROTECTED_FUNCTIONS =>
      Put(" is caling protected subprogram ");
      Put_String(Get_Master_Name(HEAD));
   when others =>
      null;
   end case;
```

```
            when COMPLETION_WAITING_ARC =>
               NEW_LINE;
               PUT("* task ");
               --                    N := NAME_OF(TASK_ID_OF(TAIL));
               --                    PUT(N.NAME(1..N.LENGTH));
               --                    if N.INDEX /= NULL_INDEX then
               --                      PUT("(");
               --                      PUT(N.INDEX, WIDTH_OF_INDEX);
               --                      PUT(")");
               --                    end if;
               Put_String(Get_Master_Name(TAIL));
               PUT(" is waiting for terminating together with task ");
               --                    N := NAME_OF(TASK_ID_OF(HEAD));
               --                    PUT(N.NAME(1..N.LENGTH));
               --                    if N.INDEX /= NULL_INDEX then
               --                      PUT("(");
               --                      PUT(N.INDEX, WIDTH_OF_INDEX);
               --                      PUT(")");
               --                    end if;
               Put_String(Get_Master_Name(HEAD));
            when PROTECTION_WAITING_ARC =>
               New_Line;
               Put("*");
               Put_Master_Class(TAIL);
               Put_String(Get_Master_Name(TAIL));
               Put(" is waiting to get protection of protected object ");
               Put_String(Get_Master_Name(HEAD));
            when PROTECTED_ENTRY_CALLING_ARC =>
               New_Line;
               Put("*");
               Put_Master_Class(TAIL);
               Put_String(Get_Master_Name(TAIL));
               Put(" is calling entry ");
               Put_String(Get_Arc_Label(TAIL, HEAD));
               Put(" of protected object ");
               Put_String(Get_Master_Name(HEAD));
            when NO_ARC =>
               null;
            end case;
            Nownode := Nownode.Next;
         end loop;
         NEW_LINE;

         if Async_Result then
            Put_Line("* But, this deadlock is possible to be removed by trigger part of asynchronous entry call.
             If follow tasks is going to be deadlocked or deadlock blocked, the possibility will disappear.");
            Put("* deadlock dependent tasks =>");
            declare
               Nownode : Node_List_Link := Async_Depend;
            begin
               while Nownode /= null loop
                  Put_String(Get_Master_Name(Nownode.Mynode));
                  Put(" ");
                  Nownode := Nownode.Next;
               end loop;
            end;
            New_Line;
         end if;

         if Debug_Mode then
            TWFG.Dump_Master_List;
         end if;
         TWFG.Dump_Task_Status;
         Flush;

      end if;
   end CHECK_CIRCULAR_DEADLOCKS;

   procedure Dump_Task_Status is
      Current_Node : Task_Index_List.NODE;
      Task_Master : Master_Index_Link;
      Task_Status : Status_Of_Task;
      Current_Master : Master_Index_Link;
      Communicate_Task : Master_Index_Link;
   begin
      if not Task_Index_List.IsNULL(All_Task_ID) then
         New_Line;
         Put_Line("* Status of all tasks are follows :");
         New_Line;
         Current_Node := All_Task_ID.all;
         loop
            Task_Master := TWFG.Get_Original_Master(Current_Node.Index);
            Current_Master := TWFG.Get_Current_Master(Current_Node.Index);
            Put("* task name                => ");
            if TWFG.Get_Master_Class(Task_Master) = MAIN_PROCEDURE then
               Put_Line("MAIN_TASK");
            else
               Put_String(Task_Master.Master_Name);
               New_Line;
               Put("* parent of the task     => ");
               Put_String(Task_Master.Parent_Master.Master_Name);
               New_Line;
            end if;
            Task_Status := TWFG.Get_Task_Status(Current_Node.Index);
            Put("* state of the task      => ");
            Task_State_IO.Put(Task_Status);
            New_Line;
            case Task_Status is
            when SIMPLE_ENTRY_CALLING =>
               Put("* communication entry    => ");
               Communicate_Task := TWFG.Get_Communicate_Task(Current_Master);
               Put_String(Communicate_Task.Master_Name);
               Put(".");
               Put_String(TWFG.Get_Arc_Label(Current_Master,
                 Communicate_Task));
               New_Line;
            when BLOCK_ELABORATING|BLOCK_EXECUTION_WAITING =>
               Put("* calling subprogram     => ");
               Put_String(Get_Master_Name(Current_Master));
```

```
            New_Line;
         when BLOCK_COMPLETED =>
            Put("* calling subprogram     => ");
            Put_String(Get_Master_Name(Get_Completed_Master(Current_Node.Index)));
            New_Line;
         when PROTECTED_SUBPROGRAM_ELABORATING|
              PROTECTED_SUBPROGRAM_EXECUTION_WAITING =>
            Put("* calling protected subprogram => ");
            Put_String(Get_Master_Name(Get_Completed_Master(Current_Node.Index)));
            New_Line;
         when PROTECTED_SUBPROGRAM_COMPLETED =>
            Put("* calling protected subprogram => ");
            Put_String(Get_Master_Name(Current_Master));
            New_Line;
         when PROTECTED_ENTRY_CALLING|
              PROTECTED_ENTRY_BLOCK_ELABORATING =>
            Put("* calling protected entry => ");
            Put_String(Get_Master_Name(Current_Master));
            Put(".");
            Put_String(Get_Arc_Label(Current_Master.Parent_Master,
              Current_Master));
            New_Line;
         when ACCEPTING =>
            Put("* communication entry    => ");
            Put_String(Current_Master.Master_Name);
            Put(".");
            Put_String(Get_Calling_Entry(Current_Master));
            New_Line;
         when BLOCKED_BY_PROTECTION =>
            Put("* calling subprogram     => ");
            Put_String(Get_Master_Name(Get_An_Out_Arc_Destination(Current_Master)));
            Put(".");
            Put_String(Get_Master_Name(Current_Master));
            New_Line;
         when others => null;
         end case;
         New_Line;
         Current_Node := Task_Index_List.GET_NEXT_NODE(Current_Node);
         exit when Task_Index_List."="(Current_Node, Task_Index_List.Null_Node);
      end loop;
   end if;
   Flush;
end Dump_Task_Status;

function Get_Communicate_Task(Master : Master_Index_Link)
                             return Master_Index_Link is
   Nownode : Node_List_Link := Master.Nextnodes;
begin
   while Nownode /= null loop
      if Nownode.Arcclass = ENTRY_CALLING_ARC then
         return Nownode.Mynode;
      end if;
      Nownode := Nownode.Next;
   end loop;
   return null;
   --   for I in ID_OF_MASTER loop
   --      if AM(Master_ID, I) = ENTRY_CALLING_ARC then
   --         return I;
   --      end if;
   --   end loop;
   --   return Null_Master_ID;
end Get_Communicate_Task;

procedure Abort_Node(Master : in Master_Index_Link) is
   Nownode : Node_List_Link := Master.Nextnodes;
begin
   while Nownode /= null loop
      if Nownode.Arcclass /= ENTRY_CALLING_ARC and
        Nownode.Arcclass /= PROTECTED_ENTRY_CALLING_ARC and
        Nownode.Arcclass /= COMPLETION_WAITING_ARC and
        Nownode.Arcclass /= PROTECTION_WAITING_ARC and
        Nownode.Mynode.Master_Class /= TASKS and
        Nownode.Mynode.Master_Class /= LIBRARY_PACKAGES then
         Abort_Node(Nownode.Mynode);
      end if;
      Nownode := Nownode.Next;
   end loop;
   Remove_Node(Master);
   --   for I in ID_OF_MASTER loop
   --      if AM(Master_ID, I) /= ENTRY_CALLING_ARC and
   --        AM(Master_ID, I) /= PROTECTED_ENTRY_CALLING_ARC and
   --        AM(Master_ID, I) /= COMPLETION_WAITING_ARC and
   --        AM(Master_ID, I) /= PROTECTION_WAITING_ARC and
   --        Master_ID_List(I).Master_Class /= TASKS and
   --        Master_ID_List(I).Master_Class /= LIBRARY_PACKAGES then
   --         Abort_Node(I);
   --      end if;
   --      Remove_Node(Master_ID);
   --   end loop;
end Abort_Node;

procedure Put_Master_Class(Master : in Master_Index_Link) is
begin
   Put(" ");
   Put(Unit_Class'IMAGE(Get_Master_Class(Master)));
   Put(" ");
end Put_Master_Class;

procedure Set_Standby_Arc(Callee_Task : in Task_ID;
                          Source : in Master_Index_Link;
                          Arc : in Arc_Class;
                          Label : in STRING_BUFFER) is

   Temp_Task_Index : Task_Index_Link;

   procedure Add_Standby is
      Nownode : Node_List_Link;
   begin
      if Temp_Task_Index.Standby_Arcs = null then
```

```
               Temp_Task_Index.Standby_Arcs
                 := new Node_List'(null, Source,
                                   Arc, Label);
            else
               Nownode := Temp_Task_Index.Standby_Arcs;
               if Nownode.Mynode = Source then
                  Nownode.Arcclass := Arc;
                  Nownode.Arclabel := Label;
               else
                  while Nownode.Next /= null loop
                     Nownode := Nownode.Next;
                     if Nownode.Mynode = Source then
                        Nownode.Arcclass := Arc;
                        Nownode.Arclabel := Label;
                        exit;
                     end if;
                  end loop;
                  if Nownode.Next = null and Nownode.Mynode /= Source then
                     Nownode.Next := new Node_List'(null, Source,
                                                    Arc, Label);
                  end if;
               end if;
            end if;
         end Add_Standby;

      begin
         Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID,
                                                        Callee_Task);
         if Temp_Task_Index = null then
            Temp_Task_Index := new Task_Index;
            Task_Index_List.SET_NODE(All_Task_ID, Callee_Task, Temp_Task_Index);
         end if;
         if Source /= null then
            Add_Standby;
         end if;
      end Set_Standby_Arc;

      function Get_Standby_Arc(ID : Task_ID := Current_Task)
                               return Node_List_Link is
         Temp_Task_Index : Task_Index_Link;
      begin
         Temp_Task_Index := Task_Index_List.SEARCH_NODE(All_Task_ID, ID);
         if Temp_Task_Index = null then
            return null;
         end if;
         return Temp_Task_Index.Standby_Arcs;
      end Get_Standby_Arc;

   end TWFG;

begin
   if Debug_Mode then
      Put_Line ("Event_Driven_Execution_Monitor elabolated.");
   end if;
   --    Init_Arc_Class_Name;
   --    Init_Unit_Class_Name;
   --    Input_ECT;
   Start_Time := Clock;

end Task_Wait_For_Graph_Manager;
```

C.2.8 V_Strings Package

```
-- $Id: v_strings.ads,v 1.2 2000/04/07 05:21:20 yusuke Exp $
--
-- Variable-length strings handler package specification.
--
--  by Yoshiaki Kasahara, 1992, 1993.

-- Modified by Yusuke Nonaka, 1999, 2000.

package V_Strings is

type v_string is private;

null_str: constant v_string;

function to_v(s: in string) return v_string;
function to_s(s: in v_string) return string;
function to_v(s: in Wide_string) return v_string;
function to_s(s: in v_string) return Wide_string;

function equal(s1, s2: v_string) return boolean;
function Equal_Insensitive(s1, s2: v_string) return Boolean;
function "&"(s1: v_string;  s2: v_string ) return v_string;
function "&"(s1: v_string;  s2: string   ) return v_string;
function "&"(s1: string;    s2: v_string ) return v_string;
function "&"(s1: v_string;  s2: character) return v_string;
function "&"(s1: character; s2: v_string ) return v_string;

function Match_Wildcard(S1, S2 : V_String) return Boolean;
function Length(S1 : V_String) return Natural;

private

MAX_LENGTH: constant natural := 1024;

type v_string is
record
str: string(1 .. MAX_LENGTH)
 := (1 .. MAX_LENGTH => ascii.nul);
len: natural range 0 .. MAX_LENGTH:= 0;
end record;
```

```
null_str: constant v_string:=( (1 .. MAX_LENGTH => ascii.nul), 0);

end V_Strings;



-- $Id: v_strings.adb,v 1.2 2000/04/07 05:21:29 yusuke Exp $
--
-- Variable-length strings handler package body.
--
--  by Yoshiaki Kasahara, 1992, 1993.
--  modified by Yusuke Nonaka, 1998, 1999.
with Ada.Characters.Handling;
use Ada.Characters.Handling;
package body V_Strings is

function to_v(s: in string) return v_string is
buffer: v_string;
begin
buffer.len := s'Last - s'First + 1;
buffer.str(1..buffer.len) := s;
return buffer;
end to_v;

function to_v(s: in Wide_string) return v_string is
buffer: v_string;
begin
buffer.len := s'Last - s'First + 1;
buffer.str(1..buffer.len) := To_String(S);
return buffer;
end to_v;

function to_s(s: in v_string) return string is
begin
return s.str(1..s.len);
end to_s;

function to_s(s: in v_string) return Wide_string is
begin
return To_Wide_String(s.str(1..s.len));
end to_s;

function equal(s1, s2: v_string) return boolean is
begin
return ((s1.str = s2.str) and (s1.len = s2.len));
end equal;

function Equal_Insensitive(s1, s2: v_string) return boolean is
begin
return ((To_Lower(s1.Str) = To_Lower(s2.Str)) and (s1.len = s2.len));
end Equal_Insensitive;

function "&"(s1: v_string; s2: v_string) return v_string is
buffer: v_string;
begin
buffer.len:= s1.len + s2.len;
buffer.str(1..buffer.len)
:= s1.str(1..s1.len)&s2.str(1..s2.len);
return buffer;
end "&";

function "&"(s1: v_string; s2: string) return v_string is
buffer: v_string;
begin
buffer.len:= s1.len + s2'length;
buffer.str(1..buffer.len):= s1.str(1..s1.len)&s2;
return buffer;
end "&";

function "&"(s1: string; s2: v_string) return v_string is
buffer: v_string;
begin
buffer.len:= s1'length + s2.len;
buffer.str(1..buffer.len):= s1&s2.str(1..s2.len);
return buffer;
end "&";

function "&"(s1: v_string; s2: character) return v_string is
buffer: v_string;
begin
buffer.len:= s1.len + 1;
buffer.str(1..buffer.len):= s1.str(1..s1.len)&s2;
return buffer;
end "&";

function "&"(s1: character; s2: v_string) return v_string is
buffer: v_string;
begin
buffer.len:= 1 + s2.len;
buffer.str(1..buffer.len):= s1&s2.str(1..s2.len);
return buffer;
end "&";

function Match_Wildcard(S1, S2 : V_String) return Boolean is
   S1_L : String := To_Lower(S1.Str);
   S2_L : String := To_Lower(S2.Str);
begin
--     if S2.Len < S1.Len then
--        return False;
--     end if;
   for I in 1..S1.Len loop
      if I > S2.Len or else S1_L(I) /= S2_L(I) then
 case S1_L(I) is
    when '*' =>
       for J in I..S2.Len loop
  if S2_L(J) = '.' then
     return False;
  end if;
       end loop;
```

```
         return True;
      when '+' =>
         return True;
      when others =>
         return False;
   end case;
        end if;
     end loop;

     return (S1.Len = S2.Len);
end Match_Wildcard;

function Length(S1 : V_String) return Natural is
begin
   return S1.Len;
end Length;


end V_Strings;
```