ISO/IEC 15408に基づく定理証明とモデル検査による情報セキュリティ仕様の検証技法

森本 祥一　重松 真二郎　後藤 祐一　程 京徳

情報システムの設計・開発において，セキュリティ仕様とその検証は重要な課題となっている．検証を行うには基準が必要であるが，情報システムが備えるべきセキュリティの基準を定めることは難しい．このため本論文では，IT製品や情報システムのセキュリティ評価の国際標準であるISO/IEC15408を基準として採用し，これに基づいた情報セキュリティ仕様の形式手法による検証技法を提案する．本検証技法では，形式的に記述したISO/IEC15408のセキュリティ評価基準を用いて，対象となる情報システムの仕様がISO/IEC15408の基準を満たしているかどうかを定理証明とモデル検査により厳密に検証することができる．

This paper proposes a security specification verification technique based on the international standard ISO/IEC 15408. We formalized the security criteria of ISO/IEC 15408 and developed the verification technique of security specifications based on the formalized criteria with theorem-proving and model-checking. With the technique, one can formally verify whether or not specifications satisfy the security criteria of ISO/IEC 15408. Ambiguity and/or oversight about security in specifications written in natural language can also be detected.

1　はじめに

近年のインターネットの普及を始めとするコンピュータ・ネットワークの発展に伴って，不正アクセスや改竄，情報漏洩といった脅威による被害もまた年々増加している[33]．今やセキュリティ機能は情報システムにおいて不可欠な機能である．このため，情報システムが十分なセキュリティ機能を備えているかどうかを検証することが重要な課題となっている．

検証を行うためには基準が必要であるが，その一方でどこまでセキュリティを保証すれば十分か，という問題がある．セキュリティ対策をどこまで実施しても100%安全ということは言い切れないため，何をもってセキュリティが十分であると判断するのか，その基準を定めることは難しい．また，利用者が，利用対象の情報システムでどの程度のセキュリティ対策が実施されており，その対策でどの程度まで保証してくれているのかを判断することは困難である．そこで我々は，情報システムが備えるべきセキュリティ機能の基準として，セキュリティ製品 (ハード/ソフトウェア) およびシステムの開発や製造・運用などに関するセキュリティ評価のための国際標準ISO/IEC15408[13]を採用した．ISO/IEC15408を基準として検証を行うことにより，検証対象のシステムが少なくとも国際的な基準を満たすということが言える．また，利用者にどのようなセキュリティ機能を備えているのかを明示することもできる．

しかしながら，ISO/IEC15408の評価基準自体は自然言語により，抽象的な要件として記述されている．このため，一見してその意味を理解することは難しい．自然言語で記述された評価対象の仕様がこのような難解な評価基準を満たしているかどうか，人手により検証すると工数やコストがかかる上に，厳密性に欠ける．これらの問題を解決するために，我々

A Security Specification Verification Technique Using Theorem Proving and Model Checking Based on the International Standard ISO/IEC 15408.
Shoichi Morimoto, Shinjiro Shigematsu, Yuichi Goto, Jingde Cheng, 埼玉大学, Saitama University.
コンピュータソフトウェア, Vol.23, No.3 (2006), pp.117–133.
[論文] 2005年9月1日受付.

は形式手法を用いることとした．国家機密事項や人命に関わるような高信頼性が求められるシステムでは，形式手法による分析・検証が不可欠とされる[4]．同様に，セキュリティの検証においても，形式手法を用いて厳密に検証すべきである．また形式手法を用いることにより，計算機による自動化が見込めるため，工数・コストの問題の解決が期待できる．

一般的に形式手法では，形式化した検証対象の仕様が満たすべき性質を検証者自らが作成しなければならないが，これには専門的な知識を必要とする．このため，専門的な知識を持たない検証者は検証すべき性質が何なのか，どう記述してよいのか分からないこともあり得る．また検証したい性質の記述を間違える可能性もある．そこで我々は，あらかじめ ISO/IEC15408 の全てのセキュリティ評価基準を形式化した[30]．前述のように ISO/IEC15408 のセキュリティ評価基準は抽象的な要件であるため，これらの本質的な要件を取り違えないために，数多くの ISO/IEC15408 認証済み製品の仕様書[8] を参照し，実際の評価基準の使われ方からその本質的な意味を解釈した上で形式化した．我々は既に，この形式化した ISO/IEC15408 の基準を用いた定理証明[3] による検証技法を提案した[21][22][26]．この検証技法においては，システムの動的な振る舞いの検証が問題であった．この問題を解決するために，本論文では，我々が既に提案した検証技法に新たにモデル検査[6] を導入し，定理証明とモデル検査を用いた情報セキュリティ仕様の検証技法を提案する．この検証技法により，システムの仕様が ISO/IEC15408 のセキュリティ基準を満たしているかどうか，定理証明とモデル検査を用いた厳密な検証が可能になる．

2 検証の概略と基準の形式化

本研究において用いる ISO/IEC15408 は，「Part1: 概説と一般モデル」，「Part2: セキュリティ機能要件」，「Part3: セキュリティ保証要件」の三部構成になっている．これらのうちの Part2 において，情報システムにおいてセキュリティ対策として実装すべき機能に関する要件が 11 分野 251 項目に渡って規定されている．我々が提案する検証技法では，この 251 項目のエレメントと呼ばれるセキュリティ機能についての要件を形式化し，検証の基準として採用した．

2.1 定理証明による検証技法

前述のように，我々が既に提案した検証技法においては定理証明を用いて検証を行う．定理証明は，システムが常に満たしていなければならない不変的な性質を検証するために適しているとされ，形式化した検証対象の仕様や公理を前提とし，検証基準を導出できるかどうかを推論する．導出可能であれば，その仕様上で検証基準の式が成り立つ，つまり検証基準を満たすということである．この定理証明が可能な形式手法のうち，ソフトウェアの信頼性検証において実績のある Z 記法[12] を採用した．

2.1.1 検証手順

検証の手順は以下のようになっている．

1) 必要な検証基準の雛形 (形式化したエレメント) を選ぶ．
2) 検証したい対象システムの仕様を Z で形式化する．
3) 選んだ検証基準の雛形を，形式化した対象システムの仕様に適合するよう具体化する．
4) Z で形式化した仕様上で，具体化した基準が成り立つかどうか，ツール等により検証する．

まず，利用者は 251 の検証基準の中から，検証したい性質，つまり対象システムに必要と思われる基準を選択する．各検証基準は，いかなるシステムにおいても利用できるよう雛形として形式化した．次に，検証したい対象システムの仕様を，Z で形式化する．そして，基準の雛形を対象システムで利用できるよう具体化した後，Z で形式化した対象システムの仕様上で，具体化した基準が成り立つかどうか，定理証明により検証する．定理証明をサポートするツールとして，Z のエディタと検証の機能を持つ Z/EVES[28] 等が使用できる．Z/EVES は，GUI による Z のエディタ機能が充実しており，またインタラクティブな定理証明機能による強力な検証ツールである．

2.1.2 Z によるエレメントの形式化

Z では，対象システムの仕様を，システムの状態と操作という観点から捉え，これらの状態と操作をスキーマとして表現する[1]．スキーマとは，右側の線のない箱のように表記され，変数 (エンティティ) の宣言とその変数に関係した述語とをグループ化したものである．スキーマでは，変数宣言部で変数を宣言し，述語部でそれらの変数間の性質を記述する．『仕様を Z で形式化する』とは，このスキーマを書くことである．Z で記述された仕様は，システムの状態を示す『状態スキーマ』，ある状態スキーマを変化させる操作を示す『操作スキーマ』，システムの状態や操作ではなく，単に変数の定義とそれらがどのような条件を満たす変数であるかを定義した『型スキーマ』等から構成される．

ISO/IEC15408 のエレメントは，抽象的に記述されており，実際に利用する際には，適用するシステムにおいて，エレメントの記述における各要素が何にあたるのかを Part1 や Part2 の付属書に従って定義する必要がある．よって，このエレメントを形式化する際には，エレメントを利用するために具体化しなければならない項目をそれぞれ Z のスキーマやエンティティに見立て，変数のように扱うことで雛形として形式化した．各雛形におけるイタリック強調体は状態スキーマを，単なる強調体は操作スキーマを，単なるイタリック体は集合を，サンセリフ体は型スキーマ又は論理式を，ローマン体はこれら以外の全て (エンティティ・集合の要素等) を示すこととした．以下に形式化したエレメントの例を示す．

以下は，エレメント FDP_RIP.1.1 の原文である[13]．

> **FDP_RIP.1 サブセット残存情報保護**
> **FDP_RIP.1.1** TSF は，以下のオブジェクト [選択: への資源の割当て，からの資源の割当て解除] において，資源の以前のどの情報の内容も利用できなくすることを保証しなければならない: [割付: オブジェクトのリスト].

原文中の **TSF**(TOE Security Functions) とは，評価対象の IT 製品やシステム **TOE**(Target of Evaluation) のセキュリティ機能であり，**選択**では，大括弧内でリストアップされた項目の中から，対象システムにおいて最適と思われるものを 1 つ以上選択しなければならない．また，**割付**とは，TOE ごとに具体的に詳細化しなければならない項目である．FDP_RIP.1.1 は，『**オブジェクトへの資源の割当て**又は**オブジェクトからの資源の割当て解除**が起こった場合，TOE は**資源の以前のどの情報の内容**も利用できないようにしなければならない』と述べている．この要件を Z で形式的に記述すると，以下のようになる．

$$\forall\, \boldsymbol{System}' \mid \mathbf{Allocation\_or\_deallocation\_operation} \bullet (\forall\, \text{previous\_information\_content} \in \mathit{Resource}' \bullet \text{unavailable\_conditions})$$

プライムの付いた $\boldsymbol{System}'$ は，$\boldsymbol{System}$ を変化させる任意の操作によって変化した後の状態を示す．この式は，$\boldsymbol{System}$ において，割当て又は割当て解除の操作 **Allocation_or_deallocation_operation** が実行された後，**資源** *Resource* に割当てられた全ての**以前の情報の内容** previous_information_content が，利用できなくなる条件 unavailable_conditions を満たすことを示す．

以下は，エレメント FPT_RVM.1.1 の原文である[13]．

> **FPT_RVM.1 TSP の非バイパス性**
> **FPT_RVM.1.1** TSF は，TSC 内の各機能の動作進行が許可される前に，TSP 実施機能が呼び出され成功することを保証しなければならない．

原文中の **TSP**(TOE Security Policy) とは，TOE のセキュリティポリシーであり，**TSC**(TSF Scope of Control) とは TSP を施行しなければならない TOE 内の範囲である．FPT_RVM.1.1 は，『**TSP 実施機能**が実行されるまで，**TSC 内の各機能**は許可されない』ということである．ここで，この条件を Z で形式的に記述するために，状態遷移を用いた．**TSP 実施機能**の実行前の状態を S_a，**TSP 実施機能**の実行後の状態を S_b，**TSC 内の各機能**の実行後の状態を S_c，とすると，FPT_RVM.1.1 は以下のように形式的に記述できる．

$$\begin{aligned}&\forall\, \boldsymbol{System}'' \mid \\ &\quad \mathbf{TSP\_enforcement\_functions} \fatsemi \mathbf{TSC\_functions} \bullet \\ &\qquad \text{present\_state} \in S_a \land \text{present\_state}' \notin S_a' \land \\ &\qquad\quad \text{present\_state}'' \notin S_a'' \land \text{present\_state} \notin S_b \land \\ &\qquad\qquad \text{present\_state}' \in S_b' \land \text{present\_state}'' \notin S_b'' \land \\ &\qquad\qquad\quad \text{present\_state} \notin S_c \land \text{present\_state}' \notin S_c' \land \\ &\qquad\qquad\qquad \text{present\_state}'' \in S_c''\end{aligned}$$

A ⨾ B は，スキーマの合成であり，操作 A の後に操作 B が実行されることを示す[29]. この式は，***System*** において，**TSP 実施機能 TSP_enforcement_functions** の実行後，システムの状態 present_state が S_a から S_b に遷移し，その後 **TSC 内の各機能 TSC_functions** が実行され present_state が S_b から S_c に遷移することを示す.

以上のように，エレメントにおいて具体化が必要な項目を，変数とすることで雛形として形式化した. これらの基準は，利用する際に対象システムに適合するよう具体化しなければならない.

2.1.3 検証例と基準の使用法

検証の流れを示すために，以下の例を 2.1.1 項で示した手順に従って検証する. 以下は，ISO/IEC15408 認証済み製品 (デジタルコピー機) の公開セキュリティターゲット[32] からの抜粋である.

仕様 1n(自然言語)

TOE は，消去実行要求の操作で，以下のデータ値と繰り返し回数により，HDD の実データ領域に書込み (上書き) を行うことによりデータを全消去し，残存情報を保護する.

【データ値】

FF 値を暗号化したコード

【繰り返し回数】

1 回

また，TOE は本機能が迂回されないように，操作パネル上に HDD の実データ領域の消去開始，及び消去処理中においては，その進捗状況を常に表示・更新することと，消去完了時にその旨を示すメッセージを表示する. 完了するまでその他の操作は受け付けない.

上記の仕様 1n は，コピー機上の印刷データが HDD 上に残存し，漏洩することを防ぐ機能の仕様である. 文献[32] では，仕様 1n の部分が前述の FDP_RIP.1.1 と FPT_RVM.1.1 を満たす，と述べられているので，実際にこれらを満たしているかどうか検証する.

以下は，仕様 1n を Z で形式化したものである.

仕様 1z (Z 記法)

$[State]$

$Data ::= FF \mid otherdata$

$Message ::= Beginning \mid UnderExecution \mid Completion$

$P : \mathbb{P}(\mathbb{P}\, State)$

$\forall S_1, S_2, S_3 : \mathbb{P}\, State \mid S_1 \in P \land S_2 \in P \land S_3 \in P$
$\land\ S_1 \neq S_2 \bullet S_1 \cap S_2 = \emptyset \land S_2 \neq S_3 \bullet S_2 \cap S_3 = \emptyset$
$\land\ S_1 \neq S_3 \bullet S_1 \cap S_3 = \emptyset$

HardDisk

$data\_area : \mathrm{seq}\ Data$

Copier

$HardDisk$
$S_1, S_2, S_3 : \mathbb{P}\, State;\ present\_state : State$
$message! : Message;\ userdata : \mathrm{seq}\ Data$

InitCopier

$\Delta Copier$

$data\_area' = data\_area \oplus userdata$

Start

$\Delta Copier$

$present\_state \notin S_1;\ present\_state \notin S_2$
$present\_state \notin S_3;\ S_1' = S_1$
$S_2' = S_2;\ S_3' = S_3;\ data\_area' = data\_area$
$message!' = Beginning;\ present\_state' = present\_state$

Execute

$\Delta Copier$

$present\_state \notin S_1;\ present\_state \notin S_2$
$present\_state \notin S_3;\ S_1' = S_1 \cup \{present\_state\}$
$S_2' = S_2;\ S_3' = S_3$
$message! = Beginning;\ message!' = UnderExecution$
$data\_area' = \{n : 1..\#data\_area \bullet n \mapsto FF\}$
$present\_state' = present\_state$

Complete

$\Delta Copier$

$present\_state \in S_1;\ present\_state \notin S_2$
$present\_state \notin S_3;\ S_1' = S_1 \setminus \{present\_state\}$
$S_2' = S_2 \cup \{present\_state\};\ S_3' = S_3$
$data\_area' = data\_area;\ message! = UnderExecution$
$message!' = Completion;\ present\_state' = present\_state$

$Format \mathrel{\widehat{=}} Start \fatsemi Execute \fatsemi Complete$

OtherOperation

$\Delta Copier; \ldots$

$present\_state \notin S_1;\ present\_state \in S_2$
$present\_state \notin S_3;\ S_1' = S_1$
$S_2' = S_2 \setminus \{present\_state\};\ S_3' = S_3 \cup \{present\_state\}$
$\ldots$

上記の仕様 1z について，簡単に説明する．一番上の宣言は，この状態空間には *State* という型が存在することを定義している．次の行は，データ値を示す型 *Data* は *FF* または *otherdata* のどちらかを取り得ることを示す．同様に，出力メッセージを示す型 *Message* は消去開始 *Beginning* または消去実行中 *UnderExecution* または消去完了 *Completion* のいずれかを取り得ることを示す．上下の線のないスキーマは，公理的記述である．集合の集合として，P を定義する．そして，P に含まれる集合として，集合 S_1，集合 S_2，集合 S_3 を定義する．文献[16] に倣って，これらの事実を公理として与え，これらの集合を用いてシステムの状態遷移を表現した．スキーマ *HardDisk* は，データ領域 *data_area* を持つ．*data_area* は，*Data* の列[29] から成る．スキーマ *Copier* は，このシステム自身を表す．スキーマ *Copier* は，ハードディスク *HardDisk*，自身の現在状態を示す *present_state*，出力メッセージ *message*!，ハードディスクに書き込まれるユーザデータ *userdata* を持つ．*message*! のように，! が付いているエンティティは，そのエンティティが出力であることを示す．*userdata* は，初期化スキーマ *InitCopier* で定義されるように，任意の *data_area* 上に上書きされる *Data* の列である．スキーマ *Start*，*Execute*，*Complete* は，それぞれ消去処理の開始・実行・終了の操作を定義している．スキーマ *Start* では，メッセージ *Beginning* を出力し，その他のエンティティについては，変化させない．スキーマ *Execute* では，メッセージ *UnderExecution* を出力し，*present_state* を始状態から S_1 へ遷移させる．そして，データ領域 *data_area* 全てに *FF* を上書きする．スキーマ *Complete* では，メッセージ *Completion* を出力し，*present_state* を S_1 から S_2 へ変化させる．その他のエンティティについては，変化させない．*Format* は，スキーマ *Start*，*Execute*，*Complete* がこの順序で実行されることを示し，一連の消去 (割当て解除) の操作を定義している．*Other-Operation* は，*Format* が終わってからのその他の操作を示す．*present_state* を S_2 から S_3 へ遷移させ，何かしら *Copier* を変化させる操作であることのみを定義し，その詳細は省略した．

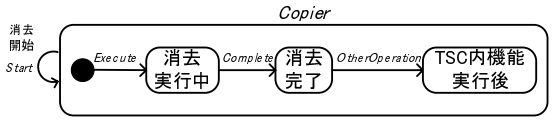


図 1 仕様 1n における状態遷移

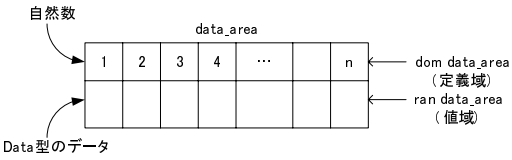


図 2 列 data_area の構成

この仕様における状態遷移を，UML(Unified Modeling Language) のステートマシン図で表したものを図 1 に示す．

仕様 1n(仕様 1z) の場合，FDP_RIP.1.1 を満たすには，割当て解除の操作 *Format* 後に**資源***data_area* の**以前のどの情報の内容**も参照できない，つまり *data_area* 上の全ての *Data* が *FF* 値で上書きされていなければならない．よって，基準の雛形 FDP_RIP.1.1 において，***System*** は仕様 1z でシステム自身を示す状態スキーマ *Copier* で，**Allocation_or_deallocation_operation** は割当て解除の操作スキーマ *Format* で，**資源** *Resource* はハードディスクのデータ領域 *data_area* で，previous_information_content は**資源**に含まれるデータ *Data* 型の要素で，参照できなくなる条件 unavailable_conditions は，データが全て *FF* 値で上書きされることを示す論理式で，それぞれ置き換える．但し，本例の場合ハードディスク上のデータ領域のような順序付けられた連続なオブジェクトを表現するために，*data_area* を集合ではなく列として定義した (図 2)．列とは，プログラミング言語における配列のようなものであり，以下のように自然数と列の要素の関数として定義されている．

$$\mathrm{seq}\ X == \{\, f : \mathbb{N} \pfun X \mid \mathrm{dom}\ f = 1..\#f \,\}$$

2.1.2 項で述べたように，基準の雛形 FDP_RIP.1.1 上の *Resource* は集合で置き換えなければならないので，ran *data_area* として置き換える (図 3)．上記

図 3　検証基準の具体化

の定義より，ran *data_area* は，列 *data_area* が持つ *Data* の集合 (値域) を示す[29].

よって，基準の雛形 FDP_RIP.1.1 を具体化した式は以下のようになる．

$$\forall Copier' \mid Format \bullet (\forall d : Data \mid d \in \operatorname{ran} data\_area' \bullet d = FF)$$

この式は，*Copier* において *Format* 実行後，*data_area* 上の全ての要素が *FF* であることを示す．この式については，Z/EVES により導出可能と判定される．これにより，*Format* 実行後は，確かに資源 *data_area* の全てのデータが参照できないことが言え，仕様 1n は FDP_RIP.1.1 を満たすことが検証できた．

同様に，FPT_RVM.1.1 を具体化し検証する．仕様 1z の場合，**TSP 実施機能**は残存情報を利用できないようにする操作スキーマ *Format* に，**TSC 内の各機能**は操作スキーマ *OtherOperation* に該当する．よって，形式化した FPT_RVM.1.1 において，***System*** は *Copier* で，**TSP_enforcement_functions** は *Format* で，**TSC_functions** は *OtherOperation* で，**TSP 実施機能**直前の状態 S_a は S_1 で，**TSP 実施機能**直後の状態 S_b は S_2 で，**TSC 内の各機能**直後の状態 S_c は S_3 でそれぞれ置き換えることができる．よって，具体化した式は以下のようになる．

$$\begin{aligned}\forall Copier''' \mid Format \fatsemi OtherOperation \bullet\ & present\_state' \in S_1' \\ \wedge\ & present\_state' \notin S_2' \wedge present\_state' \notin S_3' \\ \wedge\ & present\_state'' \notin S_1'' \wedge present\_state'' \in S_2'' \\ \wedge\ & present\_state'' \notin S_3'' \wedge present\_state''' \notin S_1''' \\ \wedge\ & present\_state''' \notin S_2''' \wedge present\_state''' \in S_3'''\end{aligned}$$

この式は，*Format* を実行し，*OtherOperation* を実行すると *Copier* の状態 *present_state* は S_1，S_2，S_3 と順に遷移する，つまり **TSP 実施機能**が実行された後に **TSC 内の各機能**が実行されたことを示す．この式については，Z/EVES により導出可能と判定される．つまり，仕様 1z は式 FDT_RVM.1.1 を満たす．

以上のように，形式化したエレメントを検証対象の仕様に対して具体化し，定理証明により検証する．

2.2　モデル検査の導入

2.1.3 項では，FPT_RVM.1.1 を状態遷移として形式化し，検証した．しかしながら，FPT_RVM.1.1 の式は，単に『ある操作が実行された後，システムの状態が S_1 から S_3 に遷移していく』という意味になっており，正確に要件を表現できているとはいえない．また，基準の式が検証対象のシステムごとの状態遷移に依存してしまうため，この式であらゆるシステムを検証できるとは限らない．更に，検証対象の仕様の形式化が複雑になってしまっている．

このように，ISO/IEC15408 で定義された 251 のエレメントの中には，システムが常に満たしていなければならない静的な要件もあり，またシステムが正しく振る舞わなければならないという動的な要件もある．一般的に，システムの動的な振る舞いを検証するためにはモデル検査が用いられる．モデル検査は，状態遷移系として形式化された検証対象の仕様上で，時相論理で記述された動的な性質を全探査して検証する．よって，FPT_RVM.1.1 のようなシステムの動的な振る舞いについてのエレメントを検証するために，モデル検査を導入する．モデル検査ツールとして，モデルを記述するための言語が分かりやすく，記述し易い，状態数の多いモデルを扱うことができる，CTL と LTL 両方を扱うことが出来るため検査したい項目を柔軟に記述できる，という理由から NuSMV [27] を用いた[2][5].

2.2.1　時相論理によるエレメントの形式化

動的な振る舞い・時間的な制約についてのエレメントは，モデル検査で検証するために時相論理[6] で形式化する．前述の FPT_RVM.1.1 を時相論理によって記述し直すと，以下のようになる．

$$\Box\ (\ \neg\ \mathrm{TSC\_functions}\ \mathrm{W}\ \mathrm{TSP\_enforcement\_functions}\ )$$

□ は Always 演算子で，時相論理式 □ p は常に p が成り立つことを示す．また，W は Weak Until 演算子で，時相論理式 p W q は q が成立するまではずっと p が成り立つということを示す．つまり上記

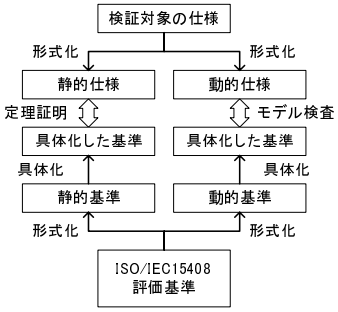


図 4 検証の手順

の式は，TSP_enforcement_functions を満たすまで TSC_functions を満たさないことが常に保証されることを示す．Z で記述した基準の雛形同様，時相論理で記述した基準の雛形においても，サンセリフ体は論理式を，ローマン体は変数を示すこととする．このように，時相論理を用いると，動的な振る舞いを簡潔かつ的確に記述できる．

2.2.2 定理証明とモデル検査による検証

ISO/IEC15408 エレメントの静的・不変的な基準については定理証明で検証するために Z 記法で形式化した．また，動的な振る舞い・時間的な基準についてはモデル検査で検証するために，時相論理式として形式化し直した．その結果，174 の静的な基準と 77 の動的な基準となった．これらをそれぞれ定理証明とモデル検査で検証するために，2.1.1 項で示した手順 2 において，システムの動的な側面については NuSMV で，静的・不変的な側面については Z で形式化することとする (図 4)．

以下は，図 1 を参考に 2.1.3 項における仕様 1n の動的側面を NuSMV で形式化したものである．

仕様 1s(NuSMV)

```
MODULE main
VAR
  data_area:{FF,otherdata};
  Message:{no_message, Beginning, UnderExecution, Completion};
  operation:{no_operation,Start,Execute,Complete,TSC_functions};
ASSIGN
--Message の遷移系
  init(Message):= no_message;
  next(Message):= case
    operation = Start    :Beginning;
    operation = Execute  :UnderExecution;
    operation = Complete :Completion;
    1                    :Message;
  esac;
--data_area の遷移系
  init(data_area):= otherdata;
  next(data_area):= case
    operation = Execute  :FF;
    1                    :data_area;
  esac;
--operation の遷移系
  init(operation):= Start;
  next(operation):= case
    operation = Start    :Execute;
    operation = Execute  :Complete;
    operation = Complete :{no_operation, TSC_functions};
    1                    :operation;
    esac;
```

上記の仕様 1s について簡単に説明する．NuSMV では，init にそれぞれの変数の初期状態を記述し，next にそれぞれ変数がどのように変化していくかの遷移を記述する．表示メッセージを示す変数 *Message* は，初期値は *no_message* で，*Start* が実行されると *Beginning* となり，*Execute* が実行されると *UnderExecution*，*Complete* が実行されると *Completion* となる．データ領域を示す変数 *data_area* は，初期値は *otherdata* で，*Execute* が実行された後 *FF* となる．システム上の操作を示す変数 *operation* は，初期値は *Start* で，順に *Execute*，*Complete* と変化していく．*Complete* になった後は，操作なし *no_operation* か TSC 内の各機能 *TSC_functions* を非決定的に取る．「非決定的に取る」とは任意のタイミングでいずれかの値を取る，ということを意味する．この仕様 1s において『**TSP 実施機能**が実行された』ということは，フォーマットの完了を示すメッセージが出力される，つまり *Message* が *Completion* になることである．また，『**TSC 内の各機能**が実行される』ということは，*operation* が *TSC_functions* になることである．よって，基準の雛形 FPT_RVM.1.1 は以下のように具体化できる．

$$\Box\ (operation \neq TSC\_functions\ \mathrm{W}\ Message = Completion\ )$$

但し，NuSMV では単項演算子 □ を G と表し，また二項演算子 W は使用できないため，時相論理式 p W q と同値である Strong Until 演算子 U を用いた時相論理式 □ p | (p U q) に記述し直し □ を G と置き

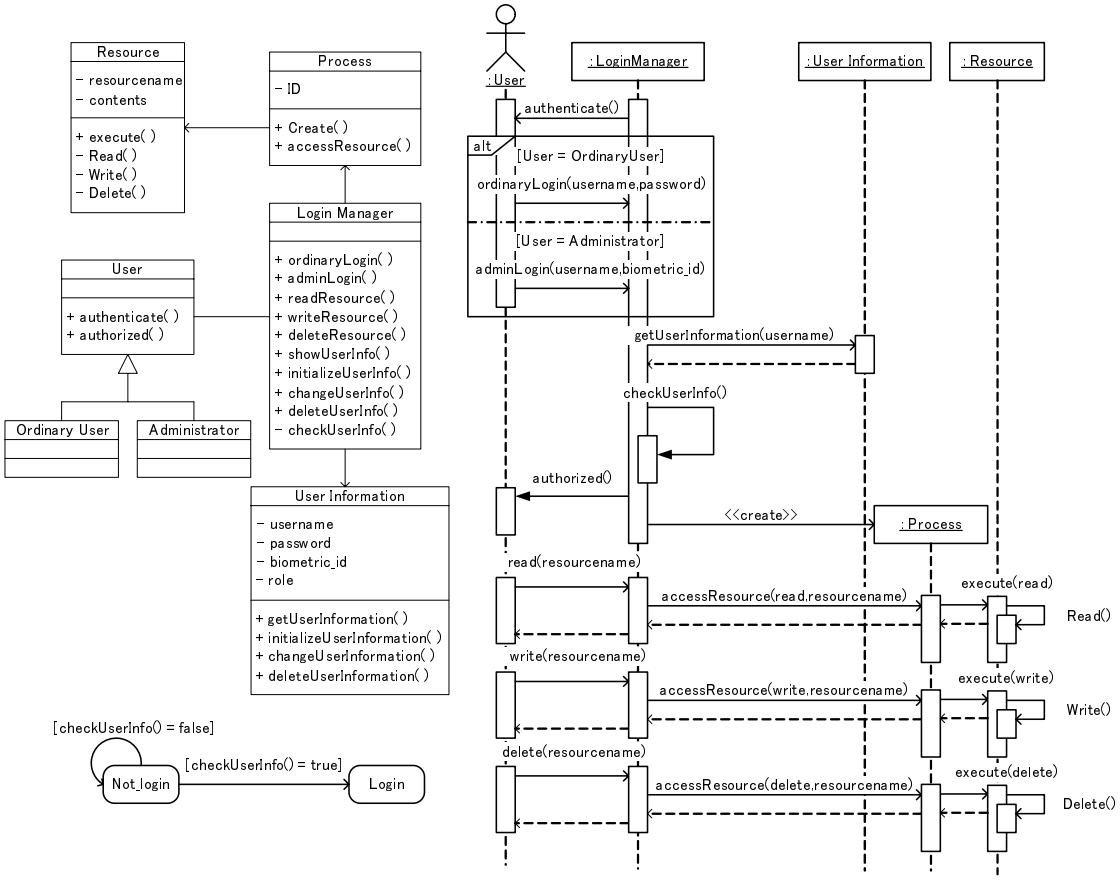


図 5　UML による仕様 (クラス図，ステートマシン図，シーケンス図)

換えて検証する必要がある．この式は，NuSMV により *true* と判定される．仕様 1s では，*Message* = *Completion* となるまで *operation* = *TSC_functions* となることはない，つまり **TSP 実施機能**の実行の後で **TSC 内の各機能**が許可される，ということであり，仕様 1n は FPT_RVM.1.1 を満たすことをモデル検査により検証できた．

このように，静的側面は Z で，動的側面は NuSMV で記述し，それぞれ定理証明とモデル検査で検証する．

3　適用例

2 章では，基準の利用法と検証法の手順を示すため，簡単な検証を行った．本章では，本検証技法の一般性を示すために，より実用的な検証例を示す．

図 5 のような UML による仕様を検証する．以下は，図 5 に対する解説である．

一般ユーザはパスワード，管理者ユーザは生体認証方式によって認証される．それぞれ 6 文字以上 12 文字以下で半角英数字を用いたパスワード，生体認証情報をあらかじめ登録しておく必要がある．一般ユーザはシステムリソースの参照のみ許可され，管理者ユーザは変更，削除も行える．認証が成功するまで，ユーザは如何なる操作も許可されない．システムは認証成功時に，ユーザプロセスを生成し，これを介してシステムリソースへの操作を行う．各ユーザは，ユーザ名，パスワード，生体認証情報を自由に変更できる．

これらの図とその解説が ISO/IEC15408 の基準を満たすかどうか検証する．まず，このシステムに必要な

エレメントを選択する必要があるが，ISO/IEC15408ではどのようなシステムにどの基準が必要かが明示されていないため，利用者が対象システムに必要と思われる基準を自身で判断し選択しなければならない．この選択の指標を明確にするため，我々は全ての認証済み公開セキュリティターゲット[8]において，どのようなシステムの分野・機能のためにどのエレメントが利用されているかを調べ，これらの依存関係を格納したデータベースを構築した[23][25]．このデータベースを用いることにより，必要なエレメントを選択する際の指標とする．本例のようなログイン機能，認証機能，アクセス制御機能，システム資源保護機能の場合は，それぞれFIA_SOS.1.1，FIA_UID.2.1，FIA_UAU.5.2，FIA_USB.1.1が必要とされている．よって，今回はこれらを満たすかどうか検証する．

3.1 定理証明による検証例

図5を検証するために，まず静的構造図(クラス図)をZで形式化する必要がある．この形式化には，RoZ[10]を使用した．RoZは，UMLのクラス図と簡単な注釈を，Zで記述された仕様に変換するツールである．但し，意味的な判断をせず機械的に変換するため，多少の修正は必要である．

3.1.1 Zによる形式記述

以下は，図5のクラス図をRoZによってZに変換し，一部修正した仕様である．

仕様 2z(Z記法)

$[Char, PartsInformation, Contents]$

$User ::= Ordinary\_user \mid Administrator$

$Role ::= ordinary\_user \mid administrator$

$Operation ::= read \mid write \mid delete$

$Bool ::= TRUE \mid FALSE;\ String == \mathrm{seq}_1\ Char$

$username == String;\ Password == String$

UserInformation

$password : username \rightarrow Password$

$biometric\_id : username \rightarrow PartsInformation$

$role : username \rightarrow Role$

Password_Policy

$input\_password? : Password;\ enable\_char : \mathbb{F}\ Char$

$a, .., z, A, .., Z, 0, .., 9 : Char$

$enable\_char = \{a, .., z, A, .., Z, 0, .., 9\}$

$\#input\_password? \geq 6;\ \#input\_password? \leq 12$

$\forall c : Char \mid c \in \mathrm{ran}\ input\_password? \bullet c \in enable\_char$

initializeUserInformation

$\Delta UserInformation;\ Password\_Policy$

$u? : username;\ input\_role? : Role$

$input\_biometric\_id? : PartsInformation$

$password' = \{(u? \mapsto input\_password?)\}$

$biometric\_id' = \{(u? \mapsto input\_biometric\_id?)\}$

$role' = \{(u? \mapsto input\_role?)\}$

getUserInformation

$\Xi UserInformation;\ u? : username$

$output\_password! : Password;\ output\_role! : Role$

$output\_biometric\_id! : PartsInformation$

$u? \in \mathrm{dom}\ password;\ u? \in \mathrm{dom}\ biometric\_id$

$u? \in \mathrm{dom}\ role$

$output\_password! = passwordu?$

$output\_biometric\_id! = biometric\_idu?$

$output\_role! = roleu?$

changeUserInformation

$\Delta UserInformation;\ u? : username$

$new\_password? : Password;\ new\_role? : Role$

$new\_biometric\_id? : PartsInformation$

$u? \in \mathrm{dom}\ password;\ u? \in \mathrm{dom}\ biometric\_id$

$u? \in \mathrm{dom}\ role$

$password' = password \oplus \{(u? \mapsto new\_password?)\}$

$biometric\_id' = biometric\_id \oplus \{(u? \mapsto new\_biometric\_id?)\}$

$role' = role \oplus \{(u? \mapsto new\_role?)\}$

deleteUserInformation

$\Delta UserInformation;\ u? : username$

$u? \in \mathrm{dom}\ password;\ u? \in \mathrm{dom}\ role$

$u? \in \mathrm{dom}\ biometric\_id$

$password' = \{u?\} \lhd password$

$biometric\_id' = \{u?\} \lhd biometric\_id$

$role' = \{u?\} \lhd role$

LoginManager

$Not\_login, Login : \mathbb{P}\ User$

$Not\_login \cap Login = \emptyset$

ordinaryLogin

$login\_user? : User$

$input\_username?, u! : username$

$input\_password? : Password$

$u! = input\_username?$

adminLogin

$login\_user? : User$

$input\_username?, u! : username$

$input\_biometric\_id? : PartsInformation$

$u! = input\_username?$

$OrdinaryUserFlow \mathrel{\widehat{=}} ordinaryLogin \gg getUserInformation$

$AdminUserFlow \mathrel{\widehat{=}} adminLogin \gg getUserInformation$

checkOrdUserInfo

$OrdinaryUserFlow;\ judge! : Bool$

$login\_user? = Ordinary\_user$

if $input\_password? = output\_password!$
$\quad \wedge\ output\_role! = ordinary\_user$

then $judge! = TRUE$ **else** $judge! = FALSE$

checkAdminUserInfo

$AdminUserFlow;\ judge! : Bool$

$login\_user? = Administrator$

if $input\_biometric\_id? = output\_biometric\_id!$
$\quad \wedge\ output\_role! = administrator$

then $judge! = TRUE$ **else** $judge! = FALSE$

$checkUserInfo \mathrel{\widehat{=}} checkOrdUserInfo \lor checkAdminUserInfo$

authorized

$\Delta LoginManager;\ checkUserInfo$

$login\_user? \in Not\_login$

if $judge! = TRUE$

then $Login' = Login \cup \{login\_user?\}$

else $Login' = Login$

Resource

$resourcename : String$

$contents : Contents$

Read

$\Xi Resource$

$\ldots$

Write

$\Delta Resource$

$\ldots$

Delete

$\Delta Resource$

$\ldots$

execute

$Read;\ Write;\ Delete$

$kind? : Operation$

---

if $kind? = read$ **then** $Read$

else if $kind? = write$ **then** $Write$

else if $kind? = delete$ **then** $Delete$

else $\Xi Resource$

Process

$ID : \mathbb{N}$

accessResource

$execute$

$argument? : Operation$

$execute[argument?/kind?]$

changeUserInfo

$LoginManager;\ changeUserInformation$

$changeUser? : User;\ change\_user? : username$

$new\_password? : Password$

$new\_biometric\_id? : PartsInformation$

$new\_role? : Role$

$change\_user? \in Login$

$changeUserInformation[change\_username?/u?]$

deleteUserInfo

$LoginManager;\ deleteUserInformation$

$delete\_user? : User;\ delete\_username? : username$

$delete\_user? \in Login$

$deleteUserInformation[delete\_username?/u?]$

showUserInfo

$LoginManager;\ getUserInformation$

$display\_user? : User;\ display\_username! : username$

$display\_password! : Password;\ display\_role! : Role$

$display\_user? \in Login$

$getUserInformation[display\_user?/u?]$

$display\_username! = output\_username!$

$display\_password! = output\_password!$

$display\_role! = output\_role!$

read

$input\_operation?, argument! : Operation$

$input\_operation? = read$

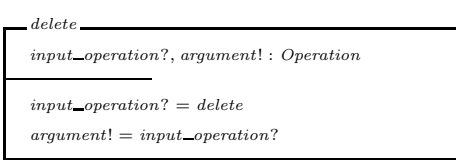
$argument! = input\_operation?$

write

$input\_operation?, argument! : Operation$

$input\_operation? = write$

$argument! = input\_operation?$



$readResource \mathrel{\widehat{=}} read \gg accessResource$
$writeResource \mathrel{\widehat{=}} write \gg accessResource$
$deleteResource \mathrel{\widehat{=}} delete \gg accessTable$

RoZ では，クラス図上のクラスとメソッドをそれぞれ Z の表記法であるスキーマとして変換する．この仕様 2z について簡単に説明する．

冒頭の宣言部は，この状態空間には *Char*，*PartsInformation*，*Contents*，*User*，*Role*，*Operation*，*Bool*，*String*，*username*，*Password* という型 (集合) が存在することを示している．*User* は *Ordinary_user* か *Administrator* を取り得る．同様に，*Role*，*Operation*，*Bool* も列挙された値のいずれかを取る．*String*，*username*，*Password* は *Char* 型のシーケンスである．クラス *UserInformation* に対応するスキーマ *UserInformation* は，要素として *password*，*biometric_id*，*role* を持つ．これらはそれぞれ *username* と，*Password*・*PartsInformation*・*Role* との全域関数である．スキーマ *Password_Policy* は，このシステムにおけるパスワードポリシーを型スキーマとして定義したものである．0,..,9 という表記は文法的に正しくないが，スペースの都合上，この表記を 0 から 9 までを示すこととした．スキーマ *getUserInformation*，*changeUserInformation*，*deleteUserInformation* は，クラス *UserInformation* のメソッドに対応し，それぞれ入力として渡されたユーザ名 *u*? に対応するパスワード・生体認証情報・ロールの情報の取得，変更，削除を定義している．*initializeUserInformation* はユーザ情報の初期化を定義している．クラス *LoginManager* に対応するスキーマ *LoginManager* では，ユーザのログイン状態を表す *Login* と，非ログイン状態を表す *Not_login*(*User* の集合) を定義している．スキーマ *ordinaryLogin* は，一般ユーザのログイン操作を表し，ログインを試みたユーザ *login_user*?にユーザ名 *input_username*? とパスワード *input_password*? の入力を促す操作を定義している．同様に，スキーマ *adminLogin* は，管理者のログイン操作を表し，ログインを試みた管理者にユーザ名と生体認証情報の入力を促す操作を定義している．これら 2 つのスキーマにおける *u*! は，*login_user*? をパイピング [20] により，他のスキーマに渡すために便宜上定義した変数である．*OrdinaryUserFlow* は，*ordinaryLogin* でログインしてきたユーザの情報を *getUserInformation* から取得することをパイピングにより定義している．同様に，*AdminUserFlow* は，*adminLogin* でログインしてきたユーザの情報を *getUserInformation* から取得することをパイピングにより定義している．*checkOrdUserInfo* は，一般ユーザがログインした時の入力されたユーザ名とパスワードが正しいかどうかチェックし，正しければ *TRUE* を，正しくなければ *FALSE* を出力する．同様に，*checkAdminUserInfo* は，管理者がログインした時の入力情報のチェックを行う．RoZ はクラス *LoginManager* のメソッド *checkUserInfo* を 1 つのスキーマとして変換していたが，記述を分かりやすくするため，これら 2 つのスキーマの離接として定義した．スキーマ *authorized* は，*checkUserInfo* の結果が *TRUE* であればユーザを *Login* 状態にし，*FALSE* であればログインを許可しない．スキーマ *Resource* の操作 *Read*，*Write*，*Delete* については，スキーマ *Resource* を操作するスキーマだということだけを定義し，詳細な形式記述については省略した．スキーマ *execute* は，入力 *kind*? によって *Read*，*Write*，*Delete* を使い分ける操作を定義している．クラス Process のメソッドに対応するスキーマ *accessResource* は，間接的にスキーマ *execute* を呼び出すことを定義している．*changeUserInfo*，*deleteUserInfo*，*showUserInfo*，*readResource*，*writeResource*，*deleteResouce* は，クラス *LoginManager* のメソッドであり，ユーザ情報の変更・削除・表示，リソースの読み書き，削除処理を示す．

3.1.2 Z による定理証明

仕様 2z が，要求されているエレメントの式を満たすかどうか検証する．FIA_UAU.5.2 は，定義したルールに従って複数の認証方式を使い分けなければならない，という要件である [13]．この要件を形式化した式は以下のようになる．

FIA_UAU.5.2

$\forall \boldsymbol{TSF}' \mid \text{users} \bullet \textbf{authentication\_mechanisms}$
$\quad \wedge\ \text{condition\_for\_authentication} \bullet \text{authorized\_state}$
$\forall \boldsymbol{TSF}' \mid \text{users} \bullet \textbf{authentication\_mechanisms}$
$\quad \wedge \neg\ \text{condition\_for\_authentication} \bullet \wedge \neg\ \text{authorized\_state}$

上記の 2 つの式のうち，上の式はシステムのセキュリティ管理機能 ***TSF*** により，利用者 users が認証される条件 condition_for_authentication を満たした場合，認証メカニズム **authentication_mechanisms** によって，ルールに従って正しく認証される (authorized_state になる) ことを意味する．下の式は，逆に認証される条件 condition_for_authentication を満たしていない場合は，認証されないことを意味する．本例では，***TSF*** は認証を司る *LoginManager* に対応するため，上記の式の ***TSF*** の部分を *LoginManager* に置き換える．同様に，認証メカニズム **authentication_mechanisms** は，認証のための機能である *checkUserInfo* となる．認証される条件 condition_for_authentication は，一般ユーザの場合は入力したパスワードと，登録してあるパスワードが一致すること，同様に管理者ユーザの場合は，入力した生体情報と，登録してある生体情報が一致することである．認証された状態 authorized_state は，あるユーザを u とすると，変化後のユーザ u' が集合 *Login* に含まれることが条件となる．それぞれ対応する部分を置き換えると，以下のような式になる．

$\forall LoginManager' \mid \forall u : User \mid u = Ordinary\_user$
$\quad \bullet\ checkUserInfo \wedge input\_password? = output\_password!$
$\quad\quad \bullet\ u' \in Login$
$\forall LoginManager' \mid \forall u : User \mid u = Administrator$
$\quad \bullet\ checkUserInfo \wedge input\_biometric\_id? = output\_biometric\_id!$
$\quad\quad \bullet\ u' \in Login$
$\forall LoginManager' \mid \forall u : User \mid u = Ordinary\_user$
$\quad \bullet\ checkUserInfo \wedge input\_password? \neq output\_password!$
$\quad\quad \bullet\ u' \notin Login$
$\forall LoginManager' \mid \forall u : User \mid u = Administrator$
$\quad \bullet\ checkUserInfo \wedge input\_biometric\_id? \neq output\_biometric\_id!$
$\quad\quad \bullet\ u' \notin Login$

上記の 4 つの式のうち，上 2 つの式は正しい情報を入力すると，一般ユーザ，管理者ユーザそれぞれが定めた認証メカニズムの適用ルールに従って正しく認証される，ということを意味し，下 2 つの式は正しい情報を入力しない場合は認証されないことを意味する．上記の 4 つの式は，Z/EVES により導出可能と判定される．つまり仕様 2z 上で FIA_UAU.5.2 が成り立つということである．

次に，FIA_USB.1.1 について検証する．FIA_USB.1.1 は，認証されたユーザは，サブジェクトを通して間接的にシステム資源を利用しなければならない，という要件である [13]．この要件を形式化した式は以下のようになる．

FIA_USB.1.1

$\forall \Delta \boldsymbol{Resource} \bullet \textbf{a\_resource\_access\_operation}$

Δ はスキーマの変化を示す ($\Delta Schema$ は変化前 $Schema$ と変化後 $Schema'$ を含む)．この式は，システム上のリソース ***Resource*** の全ての変化はサブジェクトの操作 **a_resource_access_operation** によって起こることを意味する．つまり，***Resource*** は **a_resource_access_operation** によってのみアクセスされる，ということである．***Resource*** は本例でシステム資源を示す状態スキーマ *Resource* に，**a_resource_access_operation** は資源にアクセスするための操作スキーマ *accessResource* に対応するため，具体化した式は以下のようになる．

$\forall \Delta\ Resource \bullet accessResource$

この式についても Z/EVES により導出可能と判定される．*Resource* は *Process* の *accessResource* によってのみ変化させられる，つまり仕様 2z は FIA_USB.1.1 を満たす，ということである．

続いて，FIA_SOS.1.1 について検証する．FIA_SOS.1.1 は，システムにおける秘密は定義された品質尺度を常に満たさなければならない，という要件である [13]．この要件を形式化した式は以下のようになる．

FIA_SOS.1.1

$\forall \Delta \boldsymbol{Subject}; \text{secrets} \mid \textbf{An\_operation} \bullet \text{a\_defined\_quality\_metric}$

この式は，任意の操作 **An_operation** が実行された後も，***Subject*** の秘密 secrets は定義された品質尺度 a_defined_quality_metric を常に満たすことを示す．本例の場合，秘密はパスワード，定義された品質尺度はこのシステムのパスワードポリシーに該当する．よって，形式化された FIA_SOS.1.1 におい

て，***Subject*** は本例で秘密を所有する状態スキーマ *UserInformation* で，任意の操作 **An_operation** は仕様 2z における秘密を変化させる操作スキーマの 1 つである *initializeUserInformation* で，秘密 secrets は初期化後のパスワードを示す *Password* 型のエンティティ *input_password?* で，定義された品質尺度 a defined quarity metric はパスワードポリシー「6 文字以上 12 文字以下，半角英数字」を示す論理式で，それぞれ置き換えることができる．

$$\forall \Delta UserInformation;\ input\_password? : Password \mid initializeUserInformation \bullet (\forall c : Char \mid c \in \operatorname{ran} input\_password? \bullet c \in \{a, .., z, A, .., Z, 0, .., 9\} \land \#input\_password? \geq 6 \land \#input\_password? \leq 12)$$

この式は，*UserInformation* において *initializeUserInformation* が実行された後も，*Password* はパスワードポリシーを満たすことを示す．この式については，Z/EVES により導出可能と判定される．よって，パスワード初期化後はパスワードポリシーを満たしているということができる．

本例では，パスワードの初期化以外にも秘密を変化させる操作が存在する．FIA_SOS.1.1 は，前述のように『秘密は常に定義された品質尺度を満たさなければならない』という要件であったので，秘密を変化させる操作全てにおいて，検証を行わなければならない．よって，秘密を変化させるもう 1 つの操作スキーマ *changeUserInformation* についても検証を行う．*changeUserInformation* について，先程と同様の手順で具体化した式を以下に示す．

$$\forall \Delta UserInformation;\ new\_password? : Password \mid changeUserInformation \bullet (\forall c : Char \mid c \in \operatorname{ran} new\_password? \bullet c \in \{a, .., z, A, .., Z, 0, .., 9\} \land \#new\_password? \geq 6 \land \#new\_password? \leq 12)$$

しかしながら，この式については，Z/EVES により導出することができない．つまり，パスワードの変更後はパスワードポリシーを満たしていない．本例において，パスワード変更後のパスワードポリシー適用について触れていないためである．*changeUserInformation* 後もパスワードポリシーを満たすよう以下のように修正する．

changeUserInformation

$\Delta UserInformation;$ **Password_Policy**

$u? : username;\ new\_role? : Role$

$new\_biometric\_id? : PartsInformation$

$u? \in \operatorname{dom} password;\ u? \in \operatorname{dom} biometric\_id$

$u? \in \operatorname{dom} role$

$password' = password \oplus \{(u? \mapsto \mathbf{input\_password?})\}$

$biometric\_id' = biometric\_id \oplus \{(u? \mapsto new\_biometric\_id?)\}$

$role' = role \oplus \{(u? \mapsto new\_role?)\}$

この修正により，以下の式が導出可能となる．

$$\forall \Delta UserInformation;\ input\_password? : Password \mid changeUserInformation \bullet (\forall c : Char \mid c \in \operatorname{ran} input\_password? \bullet c \in \{a, .., z, A, .., Z, 0, .., 9\} \land \#input\_password? \geq 6 \land \#new\_password? \leq 12)$$

changeUserInformation 後もパスワードポリシーを満たすことが検証できた．この修正に対応して図 5 の解説文を修正することで，本例が FIA_SOS.1.1 を満たすようになる．このように，本例におけるセキュリティ上の欠陥を検出し，修正することもできる．

3.2 モデル検査による検証例

続いて，振る舞い図についてモデル検査で検証する．ここでは時相論理により形式化した FIA_UID.2.1 を検証する．以下の記述は，図 5 の振る舞い図を状態遷移系として NuSMV 上で記述したものである．但し，紙面の都合上，検証に影響がないことを確認した上で一般ユーザと管理者ユーザを区別せず記述した．

仕様 2s(NuSMV)

```
MODULE main
VAR
  input_username:boolean;
  input_password:boolean;
  User:{Login,Not_login};
  operation:{no_operation,login,read,write,delete};
DEFINE
  CheckUserInfo:= input_username & input_password;
ASSIGN
--input_username の遷移系
  init(input_username):={0,1};
  next(input_username):={0,1};
--input_password の遷移系
  init(input_password):={0,1};
  next(input_password):={0,1};
--User の遷移系
  init(User):=Not_login;
```

```
  next(User):= case
    operation!=login     :User;
    CheckUserInfo :Login;
    1                    :User;
    esac;
--operation の遷移系
  init(operation):=no_operation;
  next(operation):= case
    User=Login     :{no_operation,read,write,delete};
    User=Not_login :{no_operation,login};
    1              :no_operation;
    esac;
```

上記の仕様 2s について，簡単に説明する．ユーザ名とパスワードを表す変数 *input_username* と *input_password* は，正しいものが入力されるとは限らないので，非決定的に 0(正しくない) か 1(正しい) を取る．ユーザの状態を表す変数 *User* は，認証前状態 *Not_login* と認証後状態 *Loign* を取る．初期状態は *Not_login* で，以後は *operation* = *login* の時に *CheckUserInfo* が真である，つまりユーザがログインを試みた時に，入力された情報が正しければ *Login* に遷移し，それ以外の場合は現状維持とする．ユーザが行える操作を表す変数 *operation* は，*User* が状態 *Not_login* では何もしない *no_operation* かログインを試みる *login* を取り，状態 *Login* では何もしない *nothing*，リソースの読み取り，書き込み，削除に該当する *read*，*write*，*delete* を非決定的に取る．

FIA_UID.2.1 は，システムにおけるあらゆるアクションよりも前に，ユーザ認証を行わなければならない，という要件である [13]．つまり，ユーザは正しく認証されるまで，いかなるアクションも実行できない，ということである．この条件を時相論理式で記述すると，以下のようになる．

FIA_UID.2.1

□ (¬ any_action_occur W authorized_state)

上記の式は，常に，ユーザが認証される条件を満たし，認証状態 authorized_state になるまで，あらゆるアクション any_action_occur が成立しないという意味である．仕様 2s 上では，認証された状態 authorized_state は，*User* が *Login* 状態になることであり，あらゆるアクションが起こる any_action_occur とは，ログイン以外の操作を取る，つまり *operation* が *read, write, delete* となる場合である．つまり具体化した式は以下のようになる．

$$\Box(\ \neg\ (operation{=}read \vee operation{=}write \vee operation{=}delete)\ \mathrm{W}\ (User = Login)\ )$$

2.2.2 項の時相論理式と同様，単項演算子 □ や二項演算子 W を NuSMV で検証可能な式に置き換えて検証すると，上記の式は *true* と判定される．このように，FIA_UID.2.1 を満たすことをモデル検査により形式的に検証することができた．

4 考察

2.1 節で示したように，Z はシステムの動的な振る舞いの記述，検証には適していない．逆に，NuSMV ではある変数がどのような条件でどう変化していくかということしか記述できず，Z のようにどの要素がどの要素を持っている，等のシステムを構成する要素間の関連性を厳密に記述できない．本検証技法において，静的側面を Z で，動的側面を NuSMV で形式化し，それぞれ定理証明とモデル検査を用いることで，静的・動的要件を的確に記述・検証することができた．

また本検証技法においては，検証対象の仕様を検証者自らが形式化しなければならないが，検証者が形式記述に不慣れであることもあり得る．また，形式化する際の解釈や記述の相違により検証が困難になってしまう可能性がある．このため本研究では，明確で読みやすく直感的に理解しやすい，人間が使用する際の使いやすさが考慮されている [20]，という点で Z を採用した．NuSMV を用いた理由も同様である．Z では，集合論や一階述語論理に基づき，対象システムの仕様をシステムの状態と操作という観点から捉え，これらの状態と操作をスキーマとして表現する．このスキーマや集合を用いて，エレメントを一般化し雛形として形式化しやすいという点も，Z を採用した理由の 1 つである．また，3 章の例のように，検証対象の仕様が UML で表現されていれば，RoZ を用いて形式化することができる．これにより，形式化の困難さの軽減や，形式化した結果をある程度統一することができ，アド・ホックな形式化を防ぐ効果が期待できる．Z で記述，検証できない部分については，モデル検査ツールで記述・検証した．UML の振る舞い図は，ほぼ機械的に状態遷移系として形式化できる．

また，我々が提案したデータベース[23][25]を用いることで，検証対象のシステムに必要なエレメント選択の負担を軽減することができる．

加えて，エレメントの形式化については，具体化しなければならない項目をフォントを使い分けて表現することにより，置き換えるべき要素を指定し，更に形式化した雛形上の変数名で対応が分かりやすいようにすることで，自然言語におけるエレメントの具体化をそのまま適用できるようにした．更に，多数の公開されている ISO/IEC15408 取得済み仕様書[8]を参照し，検証対象の仕様を形式化した際にエレメントのどの部分が状態スキーマでどの部分が操作スキーマ，スキーマの要素，集合，論理式にすれば適当かということを十分検討した上で形式化した．このように，利用者が自然言語によるエレメント中の各項目が検証対象のシステムにおいて何に該当するのかを考えるだけで，基準の具体化が容易にできるようにした．但し，具体化にはある程度 ISO/IEC15408 や形式記述についての知識が必要となる．

5 関連研究と今後の課題

Z 以外の定理証明系として，VDM[14] や Agda[7] がある．VDM は，最も最近では携帯電話の組み込み用 IC チップ開発[15]に導入される等，様々な実用例が報告されている有効なツールである．また Agda は型理論に基づく強力な証明支援系である．VDM においては Promela 記述・SPIN[11]によるモデル検査の導入が，Agda においてはモデル検査ツールの共通語となる様相 μ 計算が取り入れられ，SMV のプラグインの開発[9]が行われている．このように，様々な検証環境において定理証明とモデル検査の環境が融合されつつある．我々が形式化した ISO/IEC15408 のエレメントは，現状では Z と時相論理式が扱えるツールでのみ利用できる．よって，今後は Z や NuSMV 以外のどのような検証環境においても利用できるよう検討していく．本論文では，Z と時相論理式により ISO/IEC15408 のエレメントを記述したが，Z は一階述語論理で表現されており，特に VDM とは表現の仕方が異なるのみで，記述できる内容はほぼ同等である．また，時相論理式は主要なモデル検査ツールで検証が可能である．このような点を踏まえ，形式化したエレメントの改良を行っている．

本検証技法を実際の大規模システム等に適用する場合，検証対象の形式化と基準の具体化が課題となる．Z で記述された仕様は，前述のようにスキーマで構成され，形式化したエレメントにおいてはそれらの構成要素で置き換えられるようにしてある．時相論理によって記述されたエレメントも同様である．但し，いきなり大規模なシステム全体を形式化し，検証することは難しい．たとえ形式化できたとしても，膨大な形式仕様に対して複数の検証基準を同時に適用しなければならないため，基準の具体化が困難になる．これには，検証対象の仕様を必要に応じて 2.1.3 項で示した例の規模程度に細分化して形式化することで対応できる．我々は実際に，文献[32]を細分化することで仕様全体を検証した．また，3 章で示したように，形式化には RoZ も利用できる．一方，UML によって記述された振る舞い図のモデル検査による検証も確立している[17][18][19][31]．ソフトウェア開発において，UML による仕様記述はデファクト・スタンダードとなっている．このため，検証対象の仕様の静的側面はクラス図で，動的側面は振る舞い図で記述しておけば，それぞれの図を RoZ や文献[17][18][19][31]に基づいて容易に形式化でき，我々が形式化したエレメントを用いることができる．しかしながら，検証対象を細分化して形式化した場合については，それぞれの記述の間の整合性の問題も生じてくる．また，本論文で示した例では，1 つの形式仕様に対して複数の基準を検証することが出来たが，システムや選んだエレメントの組み合わせによっては同時に適用できない可能性もある．今後はこれらの問題の解決法について検討していく．

また，ISO/IEC15408 Part1 において，セキュリティ基本設計書のテンプレートであり，それ自身も ISO/IEC15408 の認証を受けることができるプロテクションプロファイルという文書が提案されている．我々は，このプロテクションプロファイルにおいて定義されているセキュリティ対策機能の単位を UML パターンとしてモデル化し，これらの UML パターンを用いてセキュリティ基準を満たした仕様を記述する手

法を提案した[24]．この仕様記述法により記述した仕様も，本論文で述べた検証技法を用いて検証することができる．

加えて，我々は公開されている認証済み仕様書[8]を多数検証することで形式化したエレメントの正しさを確認しているが，今後も引き続きこの作業を行うことで形式化したエレメントの妥当性を実証，洗練していく．

6 まとめ

本論文では，ISO/IEC15408 のセキュリティ評価基準を形式化し，これに基づく情報セキュリティ仕様の検証技法について述べた．この検証技法により，情報システムの仕様が，ISO/IEC15408 で定義されたセキュリティ評価基準を満たしているかどうかを定理証明とモデル検査により厳密に検証することができる．検証対象の仕様が少なくとも情報セキュリティ評価の国際標準である ISO/IEC15408 に準拠しているということが言える．利用者は，システムが満たすべきセキュリティ基準を一から考える必要はない．

また本論文では，例として実際に ISO/IEC15408 認証済の公開されている仕様の一部を検証した．単純な例ではあるが，原理的には問題なく検証できることを実証できた．更に，UML による仕様の検証も行い，本検証技法の効果を示した．今後は，更に現実的な問題に本検証技法を適用し，有効性を実証する．我々は現在，仕様の形式化や必要なエレメントの判断・選択・具体化も含め，本論文で述べた検証技法の自動化を進めている．

謝辞 本論文の原稿について貴重な御意見を下さった査読委員の方々，編集委員の方々に深く感謝致します．

参 考 文 献

[1] 荒木 啓二郎, 張 漢明: プログラム仕様記述論, オーム社, 2002.
[2] Berard, B. et al.: *Systems and Software Verification -Model-Checking Techniques and Tools*, Springer-Verlag, 1999.
[3] Bertot, Y. and Casteran, P.: *Interactive Theorem Proving and Program Development*, Springer-Verlag, 2004.
[4] Bowen, J. and Hinchey, M.: *High-Integrity System Specification and Design*, Springer-Verlag, 1999.
[5] Cimatti, A., Clarke, E., Giunchiglia, F. and Roveri, M.: NuSMV: a New Symbolic Model Verifier, in *Proceedings of the 11th International Conference Computer Aided Verification* (*CAV'99*), 1999, pp. 495–499.
[6] Clarke, E., Grumberg, O. and Peled, D.: *Model Checking*, Mit Press, 2000.
[7] Coquand, C.: *Agda*, http://www.cs.chalmers.se/~catarina/agda/
[8] Common Criteria Org: *Evaluated Product Files*, http://www.commoncriteriaportal.org/public/files/epfiles/
[9] 独立行政法人 産業技術総合研究所 システム検証研究センター: 統合的検証環境の構築, http://unit.aist.go.jp/cvs/
[10] Dupuy, S., Ledru, Y. and Chabre-Peccoud, M.: An Overview of RoZ: a Tool for Integration UML and Z Specifications, in *Proceedings of the 12th Conference on Advanced information Systems Engineering* (*CAiSE 2000*), 2000, pp. 417-430.
[11] Gerard, H.: *The Spin Model Checker*, Addison Wesley, 2003.
[12] ISO/IEC13568 Standard: *Information Technology - Z Formal Specification Notation - Syntax, Type System and Semantics*, 2002.
[13] ISO/IEC15408 Standard: *Information Technology - Security Techniques - Evaluation Criteria for IT Security*, 1999.
[14] 株式会社 CSK: VDM Tools, http://www.csk.co.jp/services/list/ito/embeded/embed_vdm/
[15] 栗田 太郎, 太田 豊一, 中津川 泰正: 携帯電話組み込み用 “モバイル FeliCa IC チップ” 開発における形式仕様記述手法導入の効果と課題, ソフトウェアシンポジウム 2005, 2005.
[16] 小池 孝政, 織田 健: システム構成要素の状態遷移規則に着目した仕様の妥当性検証法, ソフトウェア科学会第 17 回大会論文集 CD-ROM, D8-4, 2000.
[17] Latella, D., Majzik, I. and Massink, M.: Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model Checker, in *Formal Aspects of Computing*, Volume 11 Issue 6, Springer-Verlag, 1999, pp. 637-664.
[18] Lilius, J and Paltor, I.: *vUML: a Tool for Verifying UML Models*, Technical Report 272, Turku Centre for Computer Science, 1999.
[19] Mikk, E., Lakhnech, Y. and Siegel, M.: Implementing Statecharts in Promela/SPIN, in *Proceedings of Second IEEE Workshop on Industrial Strength Formal Specification Techniques* (*WIFT'98*), IEEE Press, 1998, pp. 90–101.
[20] 水野 忠則: プロトコル言語, カットシステム, 1994.
[21] Morimoto, S., Shigematsu, S. and Cheng, J.: A Formal Method for Verifying Security Specifications Based on International Standard ISO/IEC

15408 (Fast Abstract), in *Supplement of the IEEE-CS 2005 International Conference on Dependable Systems and Networks* (*DSN'05*), IEEE Press, 2005, pp. 62–63.

[22] Morimoto, S., Shigematsu, S., Goto, Y. and Cheng, J.: A Security Specification Verification Technique Based on the International Standard ISO/IEC 15408 (short paper), in *Proceedings of the 21st Annual ACM Symposium on Applied Computing* (*SAC'06*), ACM Press, 2006, pp. 1802–1803.

[23] Morimoto, S., Horie, D. and Cheng, J.: A Security Requirement Management Database Based on ISO/IEC 15408, in *Proceedings of the International Conference Computational in Computational Science and its Applications* (*ICCSA'06*), Lecture Notes in Computer Science, Vol. 3982, Springer-Verlag, 2006, pp. 1–15.

[24] 森本 祥一, 程 京徳: UML によるプロテクションプロファイルのモデル化とその形式的検証, 電子情報通信学会論文誌「情報・システム: D」, Vol. J89-D, No. 4 (2005), pp. 726–742.

[25] 森本 祥一, 堀江 大輔, 程 京徳: ISO/IEC15408 に基づく情報セキュリティ要求管理データベース, 日本データベース学会論文誌 DBSJ Letters, Vol. 4, No. 3 (2005), pp. 13–16.

[26] 森本 祥一, 重松 真二郎, 程 京徳: ISO/IEC15408 の形式化に基づく情報セキュリティ仕様の形式的検証法 (ポスター論文), 先進的計算基盤システムシンポジウム SACSIS2005 論文集, IPSJ Symposium Series Vol. 2005, No. 5 (2005), pp. 201–202.

[27] NuSMV, http://nusmv.irst.itc.it/

[28] ORA Canada: *Z/EVES*, http://www.ora.on.ca/z-eves/welcome.html

[29] Potter, B., Sinclair, J. and Till, D.: *An Introduction to Formal Specification and Z, 2nd Edition*, International Series in Computer Science, Prentice-Hall, 1996.

[30] 埼玉大学大学院 理工学研究科 情報数理科学専攻 先端情報システム工学研究室: ISO/IEC15408 セキュリティ機能要件の形式記述, 2005. http://www.aise.ics.saitama-u.ac.jp/

[31] Schäfer, T., Knapp, A. and Merz, S.: Model Checking UML State Machines and Collaborations, *Electronic Notes in Theoretical Computer Science*, Vol. 55, No. 3 (2001), pp. 1–13.

[32] 東芝テック株式会社: e-STUDIO 550/650/810 用スクランブラボード GP-1010 Security Target TOE バージョン V2.0, 2004.

[33] 財団法人インターネット協会: インターネット白書 2005, インプレス, 2005.