修 士 論 文

組織内データセキュリティの定理証明による検証に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

徳田　拓

2006 年 3 月

修 士 論 文

組織内データセキュリティの定理証明による検証に関する研究

指導教官　片山 卓也 教授

審査委員主査　片山 卓也 教授
審査委員　二木 厚吉 教授
審査委員　小川 瑞史 特任教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

410086 徳田　拓

提出年月: 2006 年 2 月

目 次

第1章　序論

1.1　情報セキュリティ

情報化社会といわれる現在，柔軟かつ透明であり，安全・安心で高い信頼性を有する「高信頼性社会」の実現が，経済や社会の基盤として必要不可欠である．

情報セキュリティの目的は，システム内部の操作ミスや敵対的活動などの脅威から，情報及びシステムを守ることである．情報が水や電気と同じように，経済活動・社会生活の基盤を支えるインフラストラクチャになった今，企業や生活者が安心して情報を使えるようにし，情報の安全を確保することが重要な課題になっている．情報は国家や企業，国民一人ひとりの財産であるという観点から，その財産を守るために情報を安全に取り扱うための考え方やルールを設定する必要がある．このような経緯から，情報社会の中で活動を営むためのルールとして生まれた概念が，情報セキュリティである． それゆえ，情報セキュリティという言葉に含まれる意味は幅広く，個人情報を中心として企業の機密情報，個人の著作権など，さまざまな情報に対して生じる取扱責任を指す．具体的には，秘匿性，完全性および可用性を維持するこという．

秘匿性 秘匿性 (confidentiality) とは，特定の情報を特定の人間以外に知られないようにすることである．システム内に格納されたり，通信メッセージに含まれたりした秘匿すべき情報を，その情報を知ることを許されていない人間による不正な読み出しや，知ることを許された人間による不正な開示からも守らなければならない．

完全性 完全性 (integrity) とは，システムの状態が正しいことを保障することである．情報やサービスが事前の期待通りの状態であること，誤った更新，不正な更新を受けていないこと，正しさの保証がない情報を含んでいないことを含む．適用されるシステムやコンテキストによって，これらはシステム内の情報が互いにつじつまがあっている，システム内の情報が外界の状況を正しく表現している，システム内の情報やプログラムが不正な改竄を受けていない，システムが権限のないユーザの捜査を受けていない，システムが誤った入力を受けていない，といった用件に対応する．

可用性 可用性とは，使用の権利を持つユーザが，CPU，メモリ，ファイル，装置，情報などのシステムのリソースやサービスを使用したいときに，実際に使用可能であることをいう．リソースの喪失，リソースの独占などにより，半永久的，あるいは一時的にそのリソースが仕様不可能になり，必要に応じた仕様が妨害されないようにしなければならない．

1.2　セキュリティポリシー

セキュリティポリシーとは，組織の情報セキュリティ関する基本方針である．セキュリティ対策を効率よく，効果的に行うための指針であり，恒久的にセキュリティを維持するための仕組みを指す.

- 組織のコンピュータやネットワークを不正アクセスなどの脅威から守ること
- 利用者・管理者のセキュリティ意識を向上させること

など，ネットワークシステムを含む社内情報システムの運用・利用の指針を，運用管理者や利用者に伝達，遵守させることを目的としている．また，ネットワークシステムを含む社内情報システムを円滑に運用するうえでの指針として,「何を」「どのように」「どの程度」行うのか，セキュリティ対策基準や個別具体的な実施手順などを含む．どの情報を誰が読み取れるようにするか，どの操作を誰に対して許可するか，どのデータを暗号化するかなど，情報の目的外利用や外部からの侵入，機密漏洩などを防止するための方針を定めたものや，コンピュータウイルス感染によるデータやシステムの破壊や，トラブルによる情報システムの停止，データの喪失などに対してどう対処していくか，といった項目まで含める場合もある.

セキュリティポリシーは細かく分類すると以下の 3 つから構成される.

- エグゼクティブ・ポリシー（セキュリティ方針)
  - セキュリティ原則に基づく上位のポリシーで，社内規定に準ずる“ 憲法 ”のような存在
  - 組織内のネットワーク/コンピュータシステム，運用実態から独立して規定守るべきこと（情報漏洩防止やルール厳守など）を記述
  - 頻繁に更新しない
- ポリシー・スタンダード（セキュリティ運用管理基準または安全対策基準）
  - 企業リソースからプライオリティの高いリソースを決定
  - 対策を実施カテゴリごとに分類し，リソースや実態を明記する“ 法律 ”のような存在組織内のネットワーク/コンピュータシステム，運用実態に則して規定
  - 上位のポリシーに定められた守るべき情報を実現するための“ 手段 ”を記述
- プロシージャ（セキュリティ手順書）
  - “ 日々の手順 ”を記述
  - 一般社員，管理職，システム管理者，役員など，おのおのに適用される“ 手順 ”を規定

– ポリシースタンダードを実践するための具体的な手段

セキュリティーポリシーを示すことにより，対外的なイメージや信頼性が向上するといったメリットもある．

1.3 セキュリティポリシーモデル

セキュリティポリシーモデルとは，システムが備えるべきセキュリティに関する要件を明解かつ簡潔に記述したものである [4]．これは，システムのセキュリティを設計する際に，セキュアなシステムの抽象概念を与えるガイドラインであると同時に，システムのセキュリティを形式的数理論理的に分析する基点となる．

計算機システムのセキュリティは個々の脅威に対する個別の対策の寄せ集めでは達成せず，システム全体のセキュリティを統一的に設計した上で構築し，そのセキュリティを評価すべきである．その際の統一的アプローチの理論的よりどころがセキュリティポリシーモデルである．したがってセキュリティポリシーモデルには，適用システム全体を抽象化し，セキュリティポリシーを正確に記述でき，システム全体のセキュリティ上の特性の分析を可能とすることが求められ，さらに構築したシステムがセキュアであることの検証にも利用できることが望ましい．そのためセキュリティポリシーモデルは数理論理学的な記法で記述することが多く，それによって自然言語による表現に伴う解釈の多様性や暗黙の了解のような曖昧さを排し，同時にモデルのセキュリティ上の性質の論理的な推論による証明を可能とする．

セキュリティポリシーモデルの研究の歴史は長い．古くは 1960 年代のタイムシェアリングシステムの発展の過程で，米軍の情報管理ポリシーであるマルチレベルセキュリティのモデル化と，オペレーティングシステム（OS）への実装が進められた．1980 年代には軍用以外のシステムにおいてもセキュリティポリシーのモデル化が盛んに行われた．会計システムなどのアプリケーションによるセキュリティポリシーにも目が向けられるようになった．

近年は，World Wide Web の普及に象徴されるように，情報処理環境が大きく変化し，情報セキュリティに対する社会的要求も高度化かつ多様化している．それに伴って，セキュリティポリシーモデルが表現すべき内容やその記述形態についても多様な要求がある．そのため，特定のポリシーに特化したモデルだけではなく，多様なポリシーに対応可能な，表現力の高いモデルの研究が盛んである．大規模分散ミドルウエアなど，設計，構築にセキュリティポリシーモデルを適用すべきシステムは多い．

セキュリティポリシーモデルはそれぞれ表現能力に特色があり，秘匿性を重視したモデル，完全性を重視したモデル，秘匿性と完全性を兼ね備えた複合型モデル，中立型モデルに分類される．代表的なセキュリティポリシーモデルとしては

- Bell − LaPadula(Devid Bell&Len Lapadula)
- Biba Protection(Biba)

- CMW(John Woodward)
- Type Enforcement(Earl Boebert&Richard Kain)
- Role-based Access Control(David Ferraiolo&Richard Kuhn)
- Chinese Wall(Brewer&Nash)
- Clark-Wilson(Claerk&Willson)
- Infomation-Flow
- BMA

などがある．BMA は英国医学協会が考案した医学倫理の情報流れ制御を定義したモデルだが，これらのセキュリティポリシーモデルのほとんどは，米国国防総省がスポンサーとなって考案を委託したものである．

ただし，近年セキュリティポリシーという言葉の拡大解釈が進んだ結果，セキュリティポリシーモデルの定義はあやふやなものとなった．今日では，ISO15408 の普及に伴い，プロテクションプロファイルやセキュリティターゲットなど，新しい用語の追加と再定義が実施され，システムの保護要件を表す言葉として分類して使われるようになった．

1.4 情報セキュリティの基準

セキュリティが要求される情報処理システムの提供者は，消費者からの「本当にセキュリティは守られているのか？」という問いには答えなければならない．また，システムを提供者も，システムの開発において同じ質問を自身に問いかけることになる．この問いに答えるための方法を提示したものが情報セキュリティの評価基準である．情報セキュリティの評価基準は，各国の国内基準から世界標準まで，多数存在する．

第三者機関による，セキュリティ基準に準じた評価と公的な認証により，IT 製品・システムに関する知識が乏しい消費者でも，安心して製品・システムを選択することができる．また，提供者はセキュリティ設計への不安を取り除き，消費者に対しては，IT 製品のセキュリティ向上に積極的に取り組む企業姿勢を強くアピールでき，製品セキュリティへの信頼と安心を確保できる．

1.4.1 ISO15408

かつては，情報セキュリティの基準に関し各国が独自で基準をもっていたが (1985 年にアメリカ，1991 年にフランス，イギリス，ドイツ，1993 年にカナダに制定)，1990 年から ISO でそのジャンルの国際標準の作成が進められ，IT セキュリティ評価のコモンクライテリア (CC:Common Criteria) が作られた．CC を元に，1996 年 6 月に国際標準の情報

セキュリティの基準としてISO15408(ISO/IEC 15408)が制定された[1]．またこれと同時に，この標準のインポートによる国内標準化が開始され，2000年7月にはJIS標準（JIS X 5070）として制定さた．両者の規定内容は，同一のものとなっている．

評価・認証の手順

CCでは，多くのコミュニティのニーズに対処することができるITセキュリティ基準のセットを定義している．これらはセキュリティ要件と呼ばれる．また，CCでは，評価対象のことをTOE(Target of Evaluation)と呼ぶ．TOEの開発は，まずCCの用意するセキュリティ要件を元に，TOEに対するプロテクションプロファイル，セキュリティターゲットの開発を行う．

プロテクションプロファイル(PP：セキュリティ要件仕様書)とは，TOEに対するセキュリティ要件を記述したものである．具体的なシステムの実装とは依存しない形式で書かれており，他の実装で再利用が可能となる．PPは，CCのPPに対する評価基準に照らし合わされ，完全で一貫しており，かつ技術的にしっかりしており，TOEに対する要件の記述として適しているか評価される．そして，PPは合否判定がなされ，合格したPPは登録される．

セキュリティターゲット(ST：セキュリティ設計仕様書)とは，評価済みのPPを基にして作成され，TOEの評価に対するセキュリティ要件に加え，設計仕様を記述したものである．システムが具体的に備える保護メカニズムとTOEがどのように結び付けられるかを，ある程度詳細に記述してある．STは，CCのSTに対する評価基準に照らし合わされ，完全で一貫しており，かつ技術的にしっかりしており，TOEに対する要件の記述として適しているか評価される．そして，STは合否判定がなされ，合格したSTは登録される．

以上のステップを踏まえた上で，TOEは，STで記述される内容を具体化し，実装される．

1.5 本研究の背景と目的

1.5.1 背景

現在，情報化社会となり，政府から民間，個人に至るまで情報を扱う様々な活動が盛んに行われている．そこでは，必ずといっていいほど情報技術が関わるシステムが存在し，活動を管理・支援している．情報技術が社会生活に及ぼす影響が大きくなり，ますます高信頼である情報化社会が求められ，それを実現するする様々な技術の研究が行われている．

ソフトウェアの開発では，その問題の多くが分析・設計などの上流工程で作りこまれ，それが全体の生産性や品質に支配的な影響を及ぼしていることがよく知られている．このようなエラーを取り除くため，上流工程段階からモデルを形式化し，正しさを確認・検証しながらシステムの構築を行い，高信頼なソフトウェアの開発を実現しようとする研究が行われている．

一方，情報セキュリティが重要視される中，IT 製品のセキュリティの標準化が行われてきた．それまでは，ソフトウェアごとにセキュリティ仕様は定めらていた．しかし，この状況は開発者は何を実現すればよいか不明瞭であったし，消費者にとっても何が守られているか不明瞭であった．この状況に対し，公的機関がセキュリティの標準化を行い，これにより，開発者にとっては実装すべきセキュリティ機能が明瞭かさた．また，消費者は製品の詳細を理解できなくとも，公的機関の標準規格を満たしているということで安心を得ることができるようになった．現在では，世界標準として ISO15408 が用いられるようになった．

これから，セキュリティ標準が社会に浸透としていくことは間違いない．この状況の中，形式検証の対象としセキュリティという概念を扱うことが必要とされる．

1.5.2 目的

システムの開発には，そのシステムが満たすべき機能要求・セキュリティ要求がある．また，システムの開発過程では，システムの形式モデルを抽象度の高い状態から具体化することで実装を行う．この形式モデルは，そのシステムに与えられた機能要求・セキュリティ要求を実際に満たしている保証はない．本研究では，高階述語論理 HOL[2] による定理証明を用いて，形式モデルが機能要求・セキュリティ要求を満たしているか検証する方法を提案する．

検証により，上流工程の仕様書や形式モデルの機能要求に対する不具合・セキュリティ要求に対する不具合を発見できる．また，これを取り除くことにより，機能的・セキュリティ的に正しいことが論理的に証明された形式化モデルが得られる．

1.6 構成

本研究では，HOL を用いた検証により，形式化されたモデルが，それに対する機能要求やセキュリティ要求を満たされていることを示す方法と，一連の流れを事例ベースで述べる．事例として，ST に準ずる一般的なファイアーウォール (FW) サーバに関する仕様書を前提とする．まず仕様書に関し，オブジェクト指向での形式化を行い，形式化モデルに対し検証を行う．検証までの流れは以下のようになる．

1. FW サーバシステムの ST に準ずる仕様書を想定する
2. 仕様書を元に UML でモデル化をし，クラス図，コラボレーション図を作る

3. モデル化したものをオブジェクト指向理論を用いて ML でコード化する
4. ML コンパイラを用いて簡単な実行テストし，コード化で起こるようなバグを取り除く
5. コードを HOL 形式に書き直す
6. 仕様書の内容を証明するための，システム上の述語で命題を定義
7. 命題の証明

第 2 章では，形式化と形式検証について説明する．第 3 章では，仕様書のオブジェクト指向による形式化方法と，UML，ML 形式・HOL 形式での記述について述べる．第 4 章では，FW サーバの ST を基にした事例について，UML によるモデル化から HOL 形式による記述までについて述べる．第 5 章では，4 章で得られた形式化モデルに対しおこなった検証について述べる．第 6 章では考察を行い，第 7 章でまとめと今後の課題を述べる．

第2章　検証手法

2.1　形式検証

高信頼のソフトウェアを開発するために，開発の上流工程において早期に効率よくシステムの不具合を発見する手段として，形式検証の技術を開発に適用する試みが広がっている．形式検証はシステムの信頼性向上を目的とする技術であり，計算機支援を前提として数学的な基礎基づくことを特徴とする．形式検証は，大きく分けて定理証明による演繹手法とモデル検査手法がある．

2.1.1　演繹手法

演繹手法では与えられた性質を対象システムが満たすことを論理体系にしたがって証明する．表現力の高い論理体系を用いると広範なシステムを検証対象にすることができる．しかし，すべての定理を自動証明することは不可能であることがわかっている．このため，定理証明支援系の場合，人が証明の途中結果を理解した上で，帰納法などのルールを選択する必要がでてくる．演繹手法による検証方法では，対象が誤りを含む場合，対象記述のどこに誤りがあるかを見つけることが難しい．

2.1.2　モデル検査手法

モデル検査技法では対象を有限の状態空間で表現できるシステムに限定する．有限であることから状態の網羅的な探索が可能であり，自動検証が可能となっている．モデル検査技法では，対象が誤りを含む場合，網羅的な探索を行うことから反例を見つけることができる．

2.2　オブジェクト指向モデルにおける形式検証

現在，オブジェクト指向によるソフトウェアの開発が主流であり，本研究の検証対象であるFWサーバシステムについても，オブジェクト指向によるモデル化と，オブジェクト指向に即した検証を行う．ここでは，まずオブジェクト指向は何かということを述べる．次に，本研究で用いる検証法について述べる．

2.2.1 オブジェクト指向

オブジェクト指向はソフトウェアの開発技術の 1 つで，その名前が示すとおり「オブジェクト」という単位ですべてをとらえる考え方である．

- システム化対象となるビジネスをオブジェクト間のコミュニケーションとして考える
- システムをオブジェクトの集合ととらえる
- オブジェクトの振る舞いをプログラムで記述する

というようにあらゆる事象をオブジェクトとして考えていくのがオブジェクト指向である．オブジェクト指向は，大規模・複雑化するソフトウェア開発において開発効率を向上させる技術として注目されている．

オブジェクト指向によるシステム開発では，

- オブジェクト群がどのような構造を持っているのか
- オブジェクトの間でどのようなメッセージが交わされるのか

など，さまざまな視点からオブジェクトをとらえる．これらは，図（ダイアグラム）を使って表される．こうした図のことをモデルという．モデルを用いることで，技術者間でより正確にオブジェクトに関する情報や考えを交換することが可能となる．これは，街を地図に表すことによって，特定の建物や場所の位置，あるいは標高といった情報が皆で共有できる情報として扱えるようになる様と似ている．しかし，モデルを正確に読み取るためには，モデルの描き方のルールを定める必要がある．そうしたモデルの描き方のルールを定めたものを表記法という．

以前は，オブジェクト指向の世界には，OMT 法や Booch 法といったオブジェクト指向を使ったシステム開発のための方法論が乱立し，それぞれで独自の表記法を作っていたため，多くのの表記法が存在した．そのような状況では，同じ「クラスの構造を表すモデル」を描くための表記法が何通りも存在し，方法論が変わるたびに，別の表記法を用いなければならなかった．また，異なるエンジニアの間で情報交換をする際も，表記法が違なるためにモデルが理解できず，スムーズな情報交換ができないという問題もあった．

そこで乱立する表記法を統一するために作られたのが UML(Unified Modeling Language) である．Unified（統一）という言葉の由来はそこから来ている．UML はオブジェクト指向によるシステム開発で用いられるさまざまなモデルの表記法を標準化していき，1997 年に OMG（Object Management Group:オブジェクト指向技術の標準化団体）の標準となった．以後，オブジェクト指向業界での表記法のデファクトスタンダードとして用いられるようになった．

2.2.2 オブジェクト指向の性質と検証法の選択

オブジェクト指向モデルでは，システムは複数のオブジェクトから構成され，それらの協調動作により，システム全体としての機能が実現される．これを状態遷移に基づく形式化により各オブジェクトの状態を状態遷移で表す場合，協調動作を検証するためには複数のオブジェクトの状態遷移図を合成する必要があり，状態爆発が起こる可能性がある．本研究の検証対象は，ST などの仕様書で記述される様なシステムであり，このようなシステムでは，多数のオブジェクトが存在することが予想され，状態遷移に基づいたモデル化は困難であると場合が多いと考えられる．

これに対し，矢竹ら [3] はコラボレーションに基づくオブジェクト指向モデルの検証法を提案し，オブジェクト指向理論を定理証明システム HOL 上の論理モジュールとして定義した．オブジェクト指向理論では，システムの状態はひとつのデータ型として表現され，コラボレーションは，その上の関数適用列として表現される．これにより，システムの無限の状態を 1 つと扱うことができ，任意数のオブジェクトに対応することが可能となる．また，コラボレーションベースの検証法では，メソッド呼び出しという，オブジェクト間の協調動作を直接捉える視点でシステムの振舞いを定義できるため，段階的振舞いによる近似法のように，合成された状態遷移図から協調動作に代わる状態遷移図を抜き出すオーバーヘッドがなくなる．

以上の理由より，複数オブジェクトを扱うことを想定している本研究では，検証手法として矢竹らが提案したオブジェクト指向理論を用いて，HOL 上において定理証明による検証を用いることにした．

2.2.3 コラボレーションベースの検証法

ここでは，本研究が扱うコラボレーションの定義を確認し，コラボレーションベースの検証法について説明する．

コラボレーション

コラボレーションとは，システムのまとまった機能を実現する複数オブジェクト間の協調動作である．システムのまとまった機能とは，オブジェクト間のメソッド呼び出しを複数個組み合わせて実現される．

本研究が対象とするコラボレーションは，演繹的証明において，システムの評価対象となる状態から，次の評価対象となる状態までの遷移を実現する関数列である．例えば，対話型のシステムの場合，ユーザがコマンドを打ち込んでから，次にシステムがコマンドを受け付けることができる状態になるまでの，システムの一連の動作がコラボレーションにあたる．

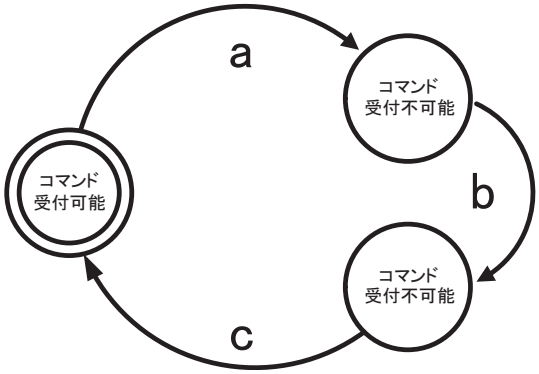

図 2.1: 対話型システムの関数列とコラボレーションの関係

図 2.1 はこれをモデル化したものである．**a**，**b**，**c** は状態の遷移を引き起こす個別の関数とする．ここで，コマンド入力可能状態から，次のコマンド入力可能状態への遷移は，関数列 **abc** によってのみ実現される．関数列 **abcabc** もコマンド入力可能状態から，コマンド入力可能状態への遷移ではあるが，途中で評価対象となる状態を通過しているため，演繹証明の帰納段階とするのは不適切である．

システムには複数のコラボレーションが存在し，それらが繰り返し適用されることによりシステムは状態変化する．

演繹的手法による証明

オブジェクト指向におけるコラボレーションベースの定理証明について説明する．コラボレーションとは，システムのまとまった機能を実現するオブジェクト間の協調動作である．一般に，システムには複数のコラボレーションが存在し，それらが繰り返し適用されることによりシステムは状態変化する．すなわち，システムは次の 2 つの組で定義される．

$$System = (S, F)$$

S はシステムのとりうる状態の集合であり，初期状態 s_0 を含む．F はコラボレーションの有限集合であり，各コラボレーションは状態を変化させる関数 $f : S \to S$ である．[1]

コラボレーションの平行性は考慮しない．つまり，それぞれのコラボレーションはアトミックに適用され，インターリーブすることはない．

前提となる状態に対する S 上述語 $P_{assumption}$，結果となる状態に対する S 上述語 $P_{conclusion}$ とすると，

$$P_{assumption}(s) \Rightarrow P_{conclusion}(f(s)) \quad (f \in F)$$

により，コラボレーション前後の状態に対する命題が与えられる．

[1] f は $f : int \to S \to string * S$ のように入出力を伴うことがある．ここでは簡単のため，入出力がない場合の定式化を示す．

特に，$P_{assumption} = P_{conclusion}$ の場合，一般にこれを不変表明といい，システム上で常に成り立つ性質を述べるために使われる．不変表明の証明は，初期状態からの遷移列に関する帰納法を用いる．

- 初期段階

$$P(s_0)$$

- 帰納段階

$$P(s) \Rightarrow P(f(s)) \quad (f \in F)$$

初期段階・帰納段階を示すことで，不変表明が常に成り立つことを演繹的に証明できる．

第3章　形式検証までの流れ

本章では，自然言語の仕様書から形式化を行い，HOL による定理証明を行う段階までについて述べる．全体の流れは

1. UML のクラス図，コラボレーション図による，オブジェクト指向モデルの作成
2. オブジェクト指向理論の論理モジュールを生成
3. オブジェクト指向理論を用いた ML によるモデルのコード化
4. ML 形式での実行テスト
5. ML 形式による記述を HOL 形式に変換
6. HOL による証明

となる．

3.1　形式化

3.1.1　形式化とは

一般にソフトウェア開発の過程では，自然言語で記述されるような仕様書から，形式化を行う．抽象度の高い形式化された設計記述から設計記述の詳細化・細分化を行い，最終的にコード化し実装に至る．抽象度の高いモデル化では以下の 2 点に注意しなければならない．

- 各詳細化のレベルは，上位レベルの**完全**な具体化であること（すなわち，より高い抽象度で定義されたすべてのセキュリティ機能，特性，及びふるまいが，下位レベルにおいて実証的に提示されていなければならない）
- 各詳細化のレベルは，上位レベルの**正確**な具体化であること（すなわち，上位レベルにおいて必要のないセキュリティ機能，特性，及びふるまいが，より低い抽象度で定義されるべきでない）

また，そこから実行テスト，統合テストと繰り返し完成となる．この抽象度の高い過程は上流過程，コードに近い過程を下流過程と呼ばれる．形式検証とは，実装段階のコードによるシュミレーションではなく，抽象度の高い段階でのモデルに対し，モデル検査や定理証明によって検証を行うことである．

3.1.2 形式化モデルに対する検証の必要性

本研究では，ST 程度の仕様書が前以って与えられていることを想定している．ST は，実装に依存しない形式である PP に実装に依存する情報を追加したものである．ST は，機能要求・セキュリティ要求について，自然言語や図表などにより非形式的な形で記述されたものである．

開発者は，ST や ST を更に詳細化した仕様書に基づき，形式モデルを作ることになる．ここで作り出される形式化モデルは，開発者が非形式的な仕様書の内容を満たすように，創作したものであるが，そのモデルが仕様書の機能要求やセキュリティ要求を満たしている保証はない．そのため，作り出されたモデルが機能要求やセキュリティ要求を満たしているかを検証する必要がでてくる．

3.2 UML によるモデル化

3.2.1 UML によるモデル化の必要性

上位レベルでの設計記述は，開発工程全体で参照されるセキュリティ機能，特性，及びふるまいを明確に理解しやすく示す必要がある．本研究で用いる，オブジェクト指向理論による ML(または HOL) 形式による記述でも，抽象度の高い設計記述は可能である．しかし，ML 形式で書かれた設計記述はコードに近い記述であり，開発に携わる者すべてにとり明確にわかり易く記述されたモデルであるとは言い難い．これには，現在最も一般的に用いられている UML による記述が適している．

検証は，UML で示された設計記述を，オブジェクト指向理論による HOL 形式に忠実に変換し，検証を行う．これにより，機能要求・セキュリティ要求を満たす UML モデルが得られることになる．

3.2.2 使用するダイアグラム

UML は 9 種類のダイアグラムからなっており，用途に合わせ，それらを組み合わせて用いる．本研究では，UML のダイアグラムとしてクラス図とシーケンス図，コラボレーション図を用いる．クラス図はシステムの静的構造を表すものであり，様々な種類のクラス (タイプ) とそれらのクラス間の関係 (リレーションシップ) で構成されている．クラス図はオブジェクト指向分析・設計において最も重要なダイアグラムであり，オブジェクト

指向分析・設計の根幹をなす．シーケンス図・コラボレーション図は，複数のオブジェクトにまたがる協調動作を記述する動的モデルである．シーケンス図は時間に焦点をおき，コラボレーション図は空間に焦点を置いているが，基本的に表現できることは同じである．

HOL による検証では，システムのオブジェクト間の協調動作による状態の変化についての，演繹的な証明によっておこなう．よって，協調動作するオブジェクトを定義するクラス図と，演繹的証明に用いるための，状態の変化をもたらすコラボレーションを記述したシーケンス図（コラボレーション図）を作る必要がある．

3.2.3 オブジェクト指向理論を意識した UML の記述

UML によるモデル化が完成すると，次のステップとして，オブジェクト指向理論を用いた，ML 形式・HOL 形式への変換を行う．これを念頭に置き，クラス図の属性値名や操作名，コラボレーション図のオブジェクト間のメッセージ通信における関数名，引数，返り値，ガード条件などは，ML 形式で共通で使う名称・形式が望ましい．

また，1 つのコラボレーションは複数の操作のまとまった機能であるが，それを構成する操作もまた，複数の操作のまとまった機能である場合がある．これらの操作に対しても，その内容についてのコラボレーション図を描いておくことが望ましい．ただし，すべての操作を詳細化しコラボレーション図を書くのは，非常に労力がかかるため，内容を記述する必要のないような自明な操作についての記述は省いてよい．コラボレーション図を書く段階で，ある程度具体化をしておくと，HOL 上での記述への変換がスムーズに行うことができる．

3.3 オブジェクト指向理論

矢竹ら [3] は，オブジェクト間の協調動作による状態の変化についての，演繹的な証明をおこなうためのシステムを実現するための土台として，オブジェクト指向理論を HOL[2] に定義した．HOL では，システム状態はストアと呼ばれるデータ型で表現する．ストアは，直感的にはシステムに存在するすべてのデータを保持する環境であり，その上にオブジェクトを操作するためのいくつかの基本演算子が定義される．コラボレーション，および，証明する命題は，これらの基本演算子と HOL の理論ライブラリが提供する関数群を用いて定義される関数，述語である．

検証の流れとしては，まず，ユーザはシステムのクラスモデルを定義する．クラスモデルは UML のクラス図のように，システムの静的構造を定義するモデルである．具体的には，クラス名，属性名とその型，継承関係を定義する．クラスモデルは，特定の形式でファイルに宣言され，理論生成機に入力される．理論生成機とは，矢竹らが HOL のメタ言語である Moscow ML [2] で実装したツールであり，対象システムのクラスモデルに基づき，そのシステムの特化した理論を HOL 理論モジュールとして構築する．理論モジュールとは，HOL におけるここの理論の単位であり，型，定数，公理，および定理を格納す

る ML ストラクチャである．構築は Definitional Extention に従う．つまり，自然数やリストといった HOL の既存理論を，定義の導入のみにより拡張し，それらに関する定理を既存の推論規則によって証明する．これは HOL において健全な理論を構築する基本的な手法である．ユーザはこの理論に与えられる基本演算子と HOL が提供する関数郡を用いて，コラボレーション，および不変表明を，システムの初期状態からの遷移列に関する帰納法により証明する．

3.3.1 クラスモデル

クラスモデルは，システムの静的構造を定義する．具体的には，クラス名，属性名とその型，継承関係を定義する．定義に先立ち，型の集合 $Type$ を導入する．$Type$ の要素は，HOL における，各変数を含まない任意の型である．

クラスモデルは，以下の 5 つの組として定義される．

$$CM = (C, A, M_{attr}, M_{inher}, T)$$

- $M_{attr} : C \to Pow(A)$
- $M_{inher} : C \to Pow(C)$
- $T : C \times A \to Type$

C，A はそれぞれ，システムに出現するクラス名の集合，属性名の集合である．M_{attr} は，クラスと，その属性集合を対応付ける写像である．M_{inher} は，クラスと，その子クラス集合を対応付ける写像である．現在のところ，継承関係は，単一継承のみサポートしている．T は，クラスと属性に対し，その属性の型を対応付けている写像である．$C \subset T$ とし，クラス名をそのクラスに属するオブジェクトの型とする．

理論生成機に入力される，ファイルには，クラス名と属性名，属性名の型，属性の初期値を記述する．属性名の型，属性の初期値は HOL 型と ML 型の両方記述しておく．ファイルを理論生成機に入力すると，HOL の理論モジュールが出力され，理論モジュールを読み込むことで，クラスモデルに特化したオブジェクト指向理論を HOL 上で使用できる．クラスモデルを定義するファイルのフォーマットは以下のようになる．

```
class   (1)
attr    (2)    holtype : (3) | (4)    mltype : (5) | (6)
```

- (1):クラス名
- (2):属性名
- (3):属性の HOL 形式の型

- (4):属性の HOL 形式の初期値
- (5):属性の ML 形式の型
- (6):属性の ML 形式の初期値

具体例として，2 次元座標上の位置を表すクラス person があるとする．クラス position は，属性として int 型の x 座標を表す x，int 型の y 座標を表す y を持つ．x，y はそれぞれ初期値 0，1 を持つとする．このときのクラスモデルを定義するファイルの記述は以下のようになる．

```
class person
attr x   holtype : num  | 0    mltype : int  | 0
attr y   holtype : num  | 1    mltype : int  | 1
```

ML 形式と HOL では，型の表記が多少異なる．HOL では int 型は無いため，num 型とする．また，ペアの表記も異なり，int 型のペアは ML 形式では int * int ，HOL 形式では num # num となる．また，bool 型の定数は ML 形式では true false となるが，HOL 形式では T F となる．

3.3.2 ストアとオブジェクトの型

変数，定数の型として，bool，num，string，list など，HOL の既存理論で提供されるものに加え，オブジェクト指向理論では，ストアやオブジェクトについての型が与えられる．

ストアは，型 *store* として定義される．*store* は定数として *store_emp* をもつ．これは，どのクラスのオブジェクトも生成されていない，空のストアを表す．

クラス $c \in C$ (C はシステムに出現するクラス名の集合) に属するオブジェクトは，型 c をもつ．各 c は，定数として c_null をもつ．これは，そのクラスの NULL オブジェクトを表す．NULL オブジェクトはストアには存在しない．

3.3.3 基本演算子

オブジェクト指向理論では，6 種類の基本演算子がストア上に定義される．

new 演算子 ストアに新たなオブジェクトを生成する関数

is 演算子 オブジェクトが特定のクラスのインスタンスであるかを検査する述語

ex 演算子 オブジェクトがストアに存在するかを検査する述語

cast 演算子 オブジェクトの方変換を行う述語

get 演算子 オブジェクトの属性値を取得する関数

set 演算子 オブジェクトの属性値を更新する述語

これらの演算子は以下のように定義される．

- $c\_new : store \to c * store \quad (c \in C)$
- $c\_is\_d : c \to store \to bool \quad (c, d \in C, c \lhd^* d)$
- $c\_ex : c \to store \to bool \quad (c \in C)$
- $c\_cast\_a : c \to store \to d \quad (c, d \in C, c \lhd^+ d \ or \ d \lhd^* c)$
- $c\_get\_a : c \to store \to T(c, a) \quad (c \in C, a \in Mattr(c))$
- $c\_set\_a : c \to T(c, a) \to store \to store \quad (c \in C, a \in Mattr(c))$

ここで，$\lhd$，$\lhd^+$，$\lhd^*$ は，クラスの継承関係を表す記号である．$c \lhd d$ は，c が d の親クラスであることを意味する．つまり，$d \in Minher(c)$ と等価である．さらに，$c \lhd^+ d$ は，c が d の祖先クラスであることを，$c \lhd^* d$ は，$c = d$ または c が d の祖先クラスであることを意味する．

3.3.4 関数，コラボレーションの記述

関数

関数は，基本演算子を組み合わせて記述する．例として，クラス person に関する関数の定義を示す．クラス person は，int 型の属性として x 座標 x，y 座標 y 持つとする．また，クラス person は関数 person_move を持つとする．move は，x 座標，y 座標それぞれ，dx，dy だけ移動させる．この関数は，属性 x，y に対する get 演算子，set 演算子と，ML の整数ライブラリに提供される'+' を用いて，ML 形式で以下のように定義される．

```
person_move : person -> int -> int -> store -> store
fun person_move p dx dy store =
      let
          val x = person_get_x p store
          val y = person_get_y p store
          val s0 = person_set_x p (x+dx) store
          val s1 = person_set_y p (y+dy) s0
      in
          s1
      end;;
```

また，すでに定義された関数は，他の関数で使用できる．person_move を用いて，x 座標を 1 加える関数 person_incX は以下のように記述される．

```
person_incX : person -> store -> store
fun person_incX p store = person_move p 1 0 store;;
```

コラボレーションの記述

コラボレーションは，まとまった機能を実現する複数オブジェクトの協調動作である．これは，複数の関数や基本演算子からなる，機能単位の関数である．よって，記述に関しては，基本的に関数の記述と同じである．

コラボレーションの例を挙げるため，もうひとつのクラス history を用意する．クラス history は，位置の履歴を残すためのクラスで，(int*int) list 型の属性 record をもつ．また，クラス history は関数 addRecord を持つとする．これは，record に履歴を追加する関数であり，以下のように定義できる

```
history_addRecord : history -> int -> int -> store -> store
fun history_addRecord h x y store =
      let
          val record = history_get_record h store
          val s0 = history_set_record h ((x,y)::record) store
      in
          s0
      end;;
```

ここで，person のを移動させ，移動先の位置履歴を記録するコラボレーション moveAndAddHistory を定義すると以下のようになる．

```
moveAndAddHistory : person -> history -> int -> int -> store -> store
fun moveAndAddHistory p h dx dy store =
      let
          val s0 = person_move p dx dy store
          val x = person_get_x p s0
          val y = person_get_y p s0
          val s1 = history_addRecord h (x,y) s0
      in
          s1
      end;;
```

3.3.5 公理

公理は，オブジェクト指向理論で定義した定数や演算子に関する基本的性質を記述したものである．公理は 36 種類存在する．証明の作業では，必要に応じ公理を用いることで，証明内容の単純化や証明を得ることができる．以下に代表的な公理を 2 つ挙げる．

- $\forall o\ s.\ c\_ex\ o\ s\ \Rightarrow\ (c\_get\_a\ o\ (c\_set\_a\ o\ x\ s) = x)$
  オブジェクト o がストアに存在するならば，その属性 a を x に変換し，その直後に同じ属性を取得したとき，その値は x である．

- $\forall o_1\ o_2\ s.\ \neg(o_1 = o_2)\ \Rightarrow\ (c\_get\_a\ o_1\ (c\_set\_a\ o_2\ x\ s) = c\_get\_a\ o_1\ s)$
  オブジェクト o_2 の属性 a を更新直後，それとは異なるオブジェクト o_1 の同じ属性を取得したとき，その値は，更新前に得られる値と等しい，つまり，異なる 2 つのオブジェクトの属性は独立している．

これらの公理を用いて証明は行われる．証明の基本的な流れは以下のようになる．

1. 命題を基本演算子に分解しする．
2. それぞれの形に適した公理を用いて，冗長な部分を削除・単純化する．
3. 公理を用いて，真理値を得る

矢竹らは，証明作業において，これらの公理を必要に応じて適応するタクティック (証明戦略) もいくつか提供している．

3.4 ML 形式，HOL 形式での記述

HOL は ML の上に作られた定理証明器である．本研究での演繹的な証明は定理証明システム HOL を用いて行われる．そのため，HOL 形式によるシステムの記述は必須事項である．しかしながら，HOL では，関数を実行するという概念がない．したがって，UML 図から直接 HOL 形式の関数に変換した場合，実行テストができないので，関数の記述段階で起こりうるバグが取り除きづらいといえる．

本研究では，モデルの仕様に対する本質的な機能やセキュリティの不具合に関する証明を目的としており，HOL の証明段階でこのようなバグが含まれていた場合，不必要な混乱を引き起こす原因となる．そのため，HOL 形式の記述段階では，下流工程で起こりうるようなバグが，取り除かれていることをが望ましい．

以上の理由より本研究では，まず ML 形式で記述し，テスト実行を行った後 HOL 形式へ変換する．

3.4.1 ML 形式化とテスト実行

オブジェクト指向モデルの ML 形式化を行う．具体的にはオブジェクト指向理論により与えられる基本演算子を用いて，システムのコラボレーションを関数として記述する．システムのコラボレーションは，この段階では UML のシーケンス図（コラボレーション図）によって記述されており，それを元にコード化を行う．基本的な記述法については，section 3.3.4 を参照されたい．

記述した関数，コラボレーションに対し，実行テストを行う．実行テストは，関数に具体的な値を入れ，返り値をチェックする．ただし，オブジェクト指向理論による関数の多くは，オブジェクトの中身の更新を行う関数であり，返り値は更新されたストアとなる場合が多い．このため，チェック対象となる属性を表示する関数を別途用意しする必要がある．例として，クラス person の属性 x，y を知るためには，以下のような関数が必要となる．

```
info_person : person -> store -> (int*int)
fun info_person p store =
      let
          val  x = person_get_x p store
          val  y = person_get_y p store
      in
          (x,y)
      end;;
```

これにより，info_person を用いて，person_move について以下のような実行テストが行える．前提として，状態 store で person 型の p が，座標 (1,1) にいるとする．

```
- info_person p store;;
- - > val it = (1, 1) : int * int
- val newStore = person_move p 1 2 store;;
- - > val newStore = store : store
- info_person p newStore;;
- - > val it = (2, 3) : int * int
- info_person p store;;
- - > val it = (1, 1) : int * int
```

ここでは，まず，初期状態でのストア（store）における，prerson 型のオブジェクト p の位置 (1，1) を表示した．次に，p を x 軸方向に 1，y 軸方向に 2 動かした（person_move p 1 2 store）．person_move は，返り値として更新されてストアを返すだけで，思ったとおりに機能したかは判定できない．次に，その更新されたストア（newStore）における p の位置 (2，3) を表示させた．

また，冗長であるが，再び初期状態でのストア（store）における，p の位置 (1，1) を表示させた．これにより，状態を表す変数ストアが，各状態を正しく保存していることがわかる．

3.5 証明

システム対し，演繹的方法により証明を行う．システムは，そのとりうる状態の集合 S ($s_0 \in S$)，状態を変化させるコラボレーション $f : S \to S$ の集合 F の組によって定義される．

$$System = (S, F)$$

証明は HOL で行うため，まず ML 形式で記述したものを HOL 形式に変換する．

3.5.1 HOL 形式化

ML で記述した関数を HOL 形式に変換する．ML 形式と HOL 形式では，記述の文法が多少異なる．また，ML が提供する関数群と HOL が提供する関数群は異なる．共通である関数（+，-，etc.）は書き直す必要は無いが，HOL で定義されていない関数は，その関数に相当する HOL 形式の関数に書き直さなければならない．

例えば，

```
filter fst snd hd ("stringA"^"stringB")
```

などは，それぞれ

```
FILTER FST SND HD (STRCAT "stringA" "stringB")
```

に書き直す．また，中には ML の関数に対応する HOL の関数が存在しない場合がある．この場合，ML の関数に対応する関数を定義する必要がある．

ML 形式での定義と HOL 形式の定義を比較する．ML 形式の person_move の定義は以下のように記述された．

```
fun person_move p dx dy store =
      let
          val x = person_get_x p store
          val y = person_get_y p store
          val s0 = person_set_x p (x+dx) store
          val s1 = person_set_y p (y+dy) s0
      in
          s1
      end;;
```

HOL 形式では以下のように定義する.

```
val person_move =
   Define ‘person_move p dx dy store =
                let  x = person_get_x p store  in
                let  y = person_get_y p store  in
                let  s0 = person_set_x p (x+dx) store  in
                let  s1 = person_set_y p (y+dy) s0  in
                   s1 ‘;;
```

この定義の結果は以下のように表示される.

```
Definition has been stored under "person_move_def".
> val person_move =
    |- !x y store.person_move p dx dy store =
               (let  x = person_get_x p store  in
                  let  y = person_get_y p store  in
                    let  s0 = person_set_x p (x+dx) store  in
                      let  s1 = person_set_y p (y+dy) s0  in
                        s1):thm
```

このようにして，ML 形式の関数をすべて HOL 形式に書き直す.

3.5.2 命題の設定

具体的な命題の作り方の例を挙げる．HOL 上のオブジェクト指向理論では，システムの状態はストアで表現される．ストアに対する述語は，真理値を返す関数として記述できる．例えば,「クラス person のオブジェクトが，座標 (1,1) にいる」という述語 P_personOn_1_1 は以下のように定義される.

```
val  P_personOn_1_1=
   Define ‘ P_personOn_1_1 p store =
                let  x = person_get_x p store  in
                let  y = person_get_y p store  in
                  ((x=1)/\(y=1))‘;;
```

この述語は，以下のような返り値を返す.

1. ストア s1 では，p の座標を (1,1) とすると
   P_personOn_1_1 p s1 は真となる.

2. ストア s2 では，p の座標を (1,1) 以外とすると（例えば (0,1)，(1,2)etc.）
P_personOn_1_1 p s2　は偽となる．

また，証明の前提条件として,「store 上に p が存在する」という述語 P_exPerson を以下のように定義する．

```
val P_exPerson =
    Define‘ P_exPerson p store = person_ex p store‘;;
```

これらの述語を用いて命題を立てる．命題の例としては，以下のようものがかける．

```
! p store . (P_personOn_1_1 p store)/\
                (P_exPerson p store)  ==>
                let newStore =  person_move p 0 0 store  in
    (P_personOn_1_1 p newStore)
```

これは,
「任意の p store に対し，p が store 上に存在し，p が (1,1) にいるならば，x 軸方向に 0，y 軸方向に 0 移動した後の p の位置は (1,1) である」
という命題である．

3.5.3　証明作業

ここでは，例としてあげた命題を実際に証明してみる．
まず，Goal の設定をする

```
g ‘! p store . (P_personOn_1_1 p store)/\
                (P_exPerson p store)  ==>
                let newStore =  person_move p 0 0 store  in
    (P_personOn_1_1 p newStore) ‘;;


- - > val it =
    Proof manager status: 1 proof.
    1. Incomplete:
         Initial goal:
         !p store.
           P_personOn_1_1 p store /\ P_exPerson p store ==>
           (let newStore = person_move p 0 0 store in
              P_personOn_1_1 p newStore)


     : proofs
```

これに対し，以下の Tactics を順に適用する．Tactics の適用は，
　この部分の証明手順と各 Tactics の意味は，付録 D.1 に記述してある．

```
e (REPEAT GEN_TAC);;
e (LET_TAC);;
e (REWRITE_TAC [person_move]);;
e (LET_TAC);;
e (REWRITE_TAC [P_personOn_1_1]);;
e (LET_TAC);;
e (REWRITE_TAC [P_exPerson]);;
e (ROT_SLICE_TAC);;
e (OBJ_SIMP_TAC);;
e (RW_TAC arith_ss []);;
```

これにより，以下の結果が得られる．

```
> val it =
    Initial goal proved.
    |- !p store.
         P_personOn_1_1 p store /\ P_exPerson p store ==>
         (let newStore = person_move p 0 0 store in
            P_personOn_1_1 p newStore) : goalstack
- -
```

以上により，Goal は証明された．証明された内容は，適当な名前をつけて，定理として利用できる．ここでは lemma_sample と名づけるとする．

```
val lemma_sample = top_thm();;
drop();;
```

また，ここまでの処理は，proveというコマンドを用いて，まとめて以下のように書くこともできる．

```
val lemma_sample = prove
    (‘‘! p store . (P_personOn_1_1 p store)/\
                (P_exPerson p store)  ==>
                let newStore =  person_move p 0 0 store  in
    (P_personOn_1_1 p newStore) ‘‘,
    EVERY
     [REPEAT GEN_TAC,
      LET_TAC,
      REWRITE_TAC [person_move],
      LET_TAC,
      REWRITE_TAC [P_personOn_1_1],
      LET_TAC,
      REWRITE_TAC [P_exPerson],
      ROT_SLICE_TAC,
      OBJ_SIMP_TAC,
      RW_TAC arith_ss []]);;
```

このように整理して保存しておき，必要なときに実行し，利用すると便利である．

第4章　仕様書とモデル化

この章では，自然言語の仕様書から形式化を行いHOL形式の記述に移すまでを実例で示す．事例として，一般的なネットワークアドレス変換機能のついたファイアーウォール(FW)サーバの仕様書を想定する．仕様書の内容としては，STから機能に関するの記述や実装に関するの記述を抽出した程度の内容とする．

仕様書全文は，付録に掲載する．ここでは，仕様書の内容について簡単な説明を行う．詳細な内容については，付録Aを参照されたい．

4.1　FWサーバ仕様書

FWサーバの仕様書は自然言語で書かれており，主に4つの機能に関する要求，セキュリティィに関する要求が記述されている．

- パケットフィルタリングに関するポリシー
- 識別と認証に関するポリシー
- アクセスコントロールに関するポリシー
- 監査に関するポリシー

尚，ここでは対象となるFWサーバ，およびそのシステム全体を指して，TOE（target of evaluation）と呼ぶことにする．また，TOEを利用するユーザをアドミニストレータとする．アドミニストレータはFW管理者と監査役の2種類がいる．

4.1.1　パケットフィルタリングに関するポリシー概要

パケットフィルタリングに関する機能についての記述.

TOEは，packet processing ruleに従い，通過するパケットをパスするかドロップするか決定する．また，ネットワークアドレス変換（NAT:network adress translation）機能が備わっている．

4.1.2 識別と認証に関するポリシー概要

ユーザ情報の管理，ログインに関する機能についての記述.

- TOE は，TOE にアクセスするユーザを，権限を与えられた FW 管理者，または監査役として識別・認証する.
- TOE は，識別認証が成功していない場合，TOE に関する操作を使用することを許可しない.
- FW 管理者・監査役に対する識別・認証は ID とパスワードの照合によって行う.
- TOE は，規定回数パスワード照合に失敗した場合，そのアカウントをロックする.
- TOE は，ユーザが何もせず 15 分経過した場合，アクティブセッションを終了する.

4.1.3 アクセスコントロールに関するポリシー概要

認証を受けたアドミニストレータが使用できる TOE の機能，その制限の仕方についての記述.

- TOE は，ロールに応じて FW 管理者や監査役に対し，TOE を操作するために必須の構成データを読んだり変更する能力を制限する．対象となる項目は以下のようなものである.
  - packet processing rules
  - アドミニストレータの情報
  - アカウントロック設定
  - 監査するイベントの選択
- TOE は，パケットフィルタリング機能を開始・停止する能力を FW 管理者に制限する
- TOE は，全体の TOE の操作を終了させる能力を FW 管理者のみに制限する

4.1.4 監査に関するポリシー概要

監査のため様々なイベントの記録を残す機能についての記述.

TOE は，監査するイベントが発生した場合，監査記録を生成し，TOE のセキュアな操作を監査するための必要な情報を収集する.

4.2 仕様書の形式化

FW サーバの仕様書から，UML でのクラス図，コラボレーション図への形式化を行う．FW サーバの仕様書の内容に関しては，付録 A を参照されたい．

仕様書では，パケットフィルタリング，識別・認証，アクセスコントロール，監査についての 4 つの機能が記述されている．これらの機能のうち，パケットフィルタリング，識別・認証，監査は具体的な操作が提供される．また，アクセスコントロールでは，アドミニストレータがそれらを利用するルールを与えられる．アクセスコントロールの形式化には，強制アクセス制御のセキュリティポリシーモデルである RBAC(Role Based Access Contorl)[5, 6] の概念を取り入れた．

形式化では，まず具体的な機能を提供する，パケットフィルタリング，識別・認証，アクセスコントロールにつて機能別にモデル化をした．次に，それらを応用する機能についてモデル化，最後にそれらの機能を統合する部分をモデル化した．

UML 図に関しては，付録 B を参照されたい．付録 B には，機能ごとのクラス図と代表的ないくつかのコラボレーション図を載せてある．ここでは，機能ごとのクラス図と，それぞれの簡単な役割のみ記述する．

4.2.1 パケットフィルタリング

パケットフィルタリングに関する機能のモデル化には，以下のクラスを用いた行った．

- **pcket**
  パケットフィルタリングの対象となるパケットのクラス
- **pfManager**
  パケットフィルタリング機能を管理するクラス
- **filterRule**
  パケットフィルタリングのフィルタリングに関するルールのクラス
- **dstnatRule**
  ネットワークアドレス変換の宛先の情報の書き換えに関するルールのクラス
- **srcnatRule**
  ネットワークアドレス変換の送信元の情報の書き換えに関するルールのクラス

4.2.2 識別・認証

識別・認証に関する機能のモデル化には，以下のクラスを用いた行った．

- **loginManager**
  登録されたアドミニストレータの情報の管理と，ログイン操作に関する機能を管理するクラス

- **admin**
  アドミニストレータを表すクラス

仕様書では，アドミニストレータのロールは FW 管理者と監査役のみであったが，ロールとして ROOT を加えた．また，loginManager のオブジェクト生成時に，ID が”root”でロールが ROOT であるアドミニストレータが一人存在するようにした．また，ROOT はパスワードを認証を何度失敗してもロックされることはないものとした．

4.2.3 監査

監査に関する機能のモデル化には，以下のクラスを用いて行った．

- **auditManager**
  監査記録の管理と監査記録の書き込み機能を管理するクラス

- **auditRecord**
  監査記録のクラス

- **auditFile**
  監査記録を書き込むファイルのクラス

- **auditStrage**
  監査記録を書き込むファイルの保管空間のクラス

監査記録を書き込む操作は，他の操作の内部で呼び出される．auditManager では，あらかじめ記録できるエベントのタイプを登録しておき，他の操作で，監査記録を書き込む操作が呼び出された際，該当のイベントタイプであった場合のみ，実際に書き込みが行われる．auditRecord は auditFile に規定された量まで書き込まれる．auditStrage は auditFile を複数保管する．

4.2.4 アクセスコントロール

アクセスコントロールに関する機能のモデル化には，以下のクラスを用いて行った．

- **acManager**
  アクセスコントロール機能を管理するクラス

仕様書では，アドミニストレータのロールとしてFW管理者と監査役が存在し，それぞれの役割によって，利用可能な操作，利用不可能な操作，共通に利用できる操作があった．これらのアクセスコントロール要求を実装するために，RBACの概念を元にモデル化した．

Role Based Access Control

RBACは，強制的アクセス制御を有するセキュリティポリシーモデルである[5, 6]．強制的アクセス制御とは，従来のUNIXやWindowsのようなリソースのオーナによる任意のアクセス権設定によるアクセス制御に頼らず，セキュリティ管理者が設定するセキュリティポリシに基づいて全てのアクセス制御を行うことである．セキュリティポリシの強制により，任意のアクセス権設定による設定漏れや他のアクセス権設定と矛盾した設定による危険を防ぐことができる．

具体的には，RBACはロールに割り当てられたユーザと，ロールに割り当てられたパーミッションによって定義される(図4.1)．パーミッションとは，アクセス制限対象となる具体的な操作やリソースのことである．ロールは個々のユーザとパーミッションの間の多対多関係につけた名前とみなすことができる．これにより，ユーザへの許可の割り当てに柔軟性をもたらす．ロールを仲介することで，ユーザは，割り当てられたロールに応じたパーミッションだけの利用に，強制的に制限される．

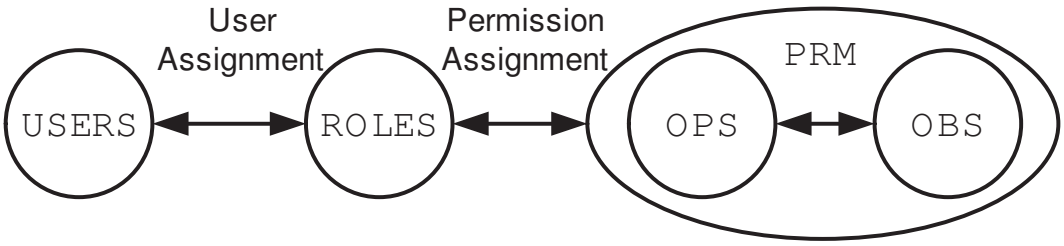

図 4.1: RBACモデル

RBACを用いたモデル化

RBACの概念を取り入れ，本研究ではアクセスコントロールのモデル化を以下のように行った．FWサーバシステムにおけるロールは，ROOT，FW管理者，監査役とし，それぞれにそれぞれに具体的操作を割り当てる．形式化では，ロールと操作にそれぞれint型でユニークな番号を割り当て，ロールと操作の関係を，$int * (int\ list)$型の

$$(ROLE, [operation_1, operation_2, operation_3, ..., operation_n,])$$

という，ペアで表した．アクセスコントロールマネージャは，このロールと操作の対応のリストを保持する．

アクセスコントロールマネージャでは，アドミニストレータがログインした後使用できる操作すべて定義する．アドミニストレータは，ログイン後は画面（コンソール）からは，ここで定義される操作しか利用できないものとした．

具体的には，以下のような記述になる．

```
01: ACMANAGER_OperationA ... =
02:   let  permission = isAccessibleToTheFunction ... operationA'ID ... in
03:     if permission then
04:         PFMANAGER_operationA ...(具体的操作の実行)
05:     else
06:         (何も起こらない)...
```

ここで，1 行目の ACMANAGER_OperationA は，アドミニストレータがコンソールから利用できる操作である．4 行目の PFMANAGER_operationA はパケットフィルタリングマネージャが提供する具体的な操作とする．

アクセスコントロールは以下の手順で行われる

1. ロールと操作の対応のリストを元に，ACMANAGER_OperationA を呼び出したアドミニストレータ (ログインしているアドミニストレータ) が，実際にその具体的機能を使用できるかチェックする
   (isAccessibleToTheFunction operationA'ID ...)
2. - 利用できる場合，各クラスが提供する具体的な操作 (PFMANAGER_operationA ...) を呼び出す
   - 利用できない場合，何も起こらず終了する

これにより，操作に対する強制的なアクセス制御が実現される．

FW 管理者，監査役にはそれぞれが行える操作の権限を，ルートにはすべての操作の権限を与えた．また，アドミニストレータがログイン後に使用できる操作は，約 50 種類になった．

4.2.5 時間に関連する機能

時間に関連する機能のモデル化には，以下のクラスを用いて行った．

- **timeManager**
  時間に関連する機能を管理するクラス

仕様書には一定の条件で時間ごと起きるイベントがあった．このような時間によるイベントの発生はシーケンス図では表現しづらいので，仮想的に，時間に関連する機能を管理するクラス timeManager を設け，このクラスによってイベントが引き起こされるようにした．

4.2.6 全体の機能

全体の機能の統合する以下のクラスを定義した．

- **fwSystemManager**
  全体の機能を統合クラス

これまでに個別に書いた機能を，統合するクラス fwSystemManager を用意する．このクラスは，各機能のマネージャのオブジェクトを属性として保持する．この生成されたこのクラスのオブジェクトのは，即ち，FW サーバシステムの初期状態を意味する．

4.3 関数とコラボレーションの ML 形式による記述とテスト実行

各クラス，コラボレーションを ML 形式，HOL 形式で記述する．まず，オブジェクト指向理論の理論生成器で論理モジュールを作るために，クラスモデルのファイルを用意する．これを理論生成器に入力し，生成された論理モジュールにより利用可能となる基本演算子を用いて，各関数，コラボレーションを記述する．

コラボレーションのコード化は，シーケンス図で記述されたコラボレーションを元に，省略されているプライベートな関数などを補いながら記述してゆく．また，実行テスト時にデバッグを行いやすいように，関数実行結果に関するメッセージを返り値に追加した．ML による簡単な実行テストを行った後，HOL 形式に書き換える．コード化とテスト実行方法については，section3.3.4 を参照されたい．FW サーバシステムのクラスファイルについては，付録 C.1 を参照されたい．コラボレーションのコードについては，全体で 1800 行程度になった．コードについては量が多すぎるので，ML で記述した関数名と型のリストのみ，付録 C.2 に掲載する．

第5章　検証

ここでは，HOL で記述した FW サーバシステムのモデルについての検証例をいくつか挙げる．

システムの検証の為には，HOL 形式で記述されたシステムのモデルが必要である．これは，ML 形式でコード化したモデルを HOL 形式に変換し得られる．検証は，コラボレーション実行前後のシステムの状態に注目し，証明すべき命題を設定する．その後，HOL を用いて命題が成り立つことを証明する．

以下「本研究において HOL で実装した FW サーバシステム」を指して,「実装システム」と呼ぶことにする．

5.1　モデルの HOL 形式化

ML で記述した関数を HOL 形式に変換する．ML 形式と HOL 形式では，記述の文法が多少異なる．また，ML が提供する関数群は HOL では使えない．これらは，HOL が提供する形式の関数の対応するものに書き直さなければならない．ML 形式から HOL 形式への書き換えについては，section3.5.1 を参照されたい．ここでは一例として，HOL 形式に変換済みのログインのコラボレーションを載せる．

```
val LOGINMANAGER_login =
 Define ‘LOGINMANAGER_login lm id pw store =
    if ~((loginManager_get_loginAdmin lm store) =  admin_null) then
        ("other user has already been login.",store)
     else
        let
            ad = LOGINMANAGER_getAdmin lm id store
        in
            if (ad = admin_null) then
                ("The ID doesn’t exist",store)
            else
                if (LOGINMANAGER_isAdminLockout ad store) then
                    ("The ID is lockout", store)
                else
                    if (LOGINMANAGER_isCorrectPW ad pw store) then
                        let s0 = admin_set_loginState ad T store  in
                        let s1 = admin_set_failNum ad 0 s0  in
```

```
        let s2 = loginManager_set_loginAdmin lm ad s1  in
            ("login success",s2)
    else
        if ((admin_get_role ad store) = ROOT) then
            ("wrong password. login fail.(root)",store)
        else
            let
                s0 = incFailNum ad store
            in
                if (overLimitUnsuccess lm ad s0) then
                    let
                         s1 = admin_set_lockState ad T s0
                    in
                        ("login fail. And the ID is lockout",s1)
                else
                    ("wrong password. login fail.",s0)‘;;
```

ログインは，以下のような流れで実行される．

1. ログインしているアドミニストレータが存在しないかチェックする．
2. ログインしているアドミニストレータが存在しないならば，入力した id に該当するアドミニストレータが存在するかチェックする．
3. 入力した id に該当するアドミニストレータが存在するならば，該当するアドミニストレータがロック状態ではないかチェックする．
4. 該当するアドミニストレータがロック状態ではないならば，入力されたパスワードが登録されているパスワードと一致するかチェックする．
   - 入力されたパスワードが登録されているパスワードと一致するならば
     - 連続失敗回数のカウンターが 0 に戻され，ログインは成功する．
   - 入力されたパスワードが登録されているパスワードと一致しないならば
     - 該当するアドミニストレータのロールが ROOT で無いならば，連続失敗回数のカウンターを増加させ，ログインは失敗する．
     - 該当するアドミニストレータのロールが ROOT のならば，連続失敗回数のカウンターを増加させず，ログインは失敗する．

5.2 検証

5.2.1 FW サーバシステムにおけるコラボレーション

本研究が対象とするコラボレーションは，演繹的証明において，システムの評価対象となる状態から，次の評価対象となる状態までの遷移を実現する関数列である．例えば，対

話型のシステムの場合，ユーザがコマンドを打ち込んでから，次にシステムがコマンドを受け付けることができる状態になるまでの，システムの一連の動作がコラボレーションにあたる．

実装システムにおいてコラボレーションに該当するものは，

- ログイン
- ログアウト
- アドミニストレータがログイン後使用できる操作，約 50 種
- パケットフィルタリング

である．

5.2.2 命題の設定

検証では，まず証明すべき命題を設定する．命題は，システムの状態から導かれる，証明すべき論理式である．命題は，状態に対する述語をとコラボレーションの作用を組み合わせ定義できる．状態に対する述語とは以下のようなものがある．

1. ID”root”はロック状態ではない
2. ID”root”のロールは ROOT である
3. ロックされていないアドミニストレータが存在する
4. 誰かログインしている
5. パケットフィルタリング機能はオフ状態である

など様々考えられる．「ID”root”はロック状態ではない」をいう述語を HOL で定義すると以下のようになる．

```
val Inv_rootIsUnlock = Define
 `Inv_rootIsUnlock fsm store = fwSystemManager_ex fsm store ==>
             admin_isUnlock
                 (LOGINMANAGER_getAdmin
                      (fwSystemManager_get_loginM fsm stor "root" store)
                 store`;;
```

これは正確には，「fwSystemManager のオブジェクト fsm がストア store 上に存在するならば，そのオブジェクトに対して ID”root”はロック状態でない」と記述される．

5.3 証明例1　不変表明の証明

不変表明の演繹的な証明は一般的な方法である．不変表明 P を S 上の述語としたとき，

- 初期段階

$$P(s_0)$$

- 帰納段階

$$P(s) \Rightarrow P(f(s)) \quad (f \in F)$$

を示すことで，不変表明が常に成り立つことが示される．ここで，f はシステムの状態を変化させるコラボレーションであり，コラボレーションが複数ある場合はすべてについて帰納段階を証明しなければならない．

ここでは，不変表明 Inv_rootIsUnlock について証明を行う．

5.3.1 帰納段階の証明

本来は，すべてのコラボレーションについて証明を行うべきであるが，簡単の為に．あらかじめ，実装システムで，Inv_rootIsUnlock に影響を及ぼすと考えられるコラボレーションは以下の 3 つである，

- LOGINMANAGER_login
  - FW サーバシステムにログインしたいユーザが ID とパスワードで認証を行う操作
  - FW 管理者，監査役の ID で認証を行い，失敗した場合，失敗の回数が規定数に達すると，その ID はロックされるよう設計してある
  - ロールが ROOT の ID で認証を行い失敗した場合は，何度失敗しても ID はロックされないように設計してある
- ACMANAGER_FwAdminLockoutConfig
  - ログイン後に使用できる操作で，FW 管理者のロック状態を設定できる
  - アクセスコントロールにより，ROOT と FW 管理者のみ実際の操作を行えるよう設計してある
- ACMANAGER_AuditorLockoutConfig
  - ログイン後に使用できる操作で，監査役のロック状態を設定できる

– アクセスコントロールにより，ROOT と監査役のみ実際の操作を行えるよう設計してある

これら 3 つの，コラボレーションでは，利用者または対象者のロック状態を変化させる関数を呼び出す設計である．また，この 3 つの以外のコラボレーションでは，ロック状態を変化させる関数は呼び出されない設計である．

ここでは，コラボレーション LOGINMANAGER_login に関する帰納段階の命題を例として挙げる．

```
01: g ‘!fsm store ID pw.
02:     (Inv_rootIsUnlock fsm store)   ==>
03:        let lm =  fwSystemManager_get_loginM fsm store    in
04:        let newS   = SND(LOGINMANAGER_login lm ID pw store)  in
05:        (Inv_rootIsUnlock fsm newS)‘;;
```

このコードでは，任意の fsm(:fwSytemManeger)，sotre(:store)，ID(:string)，pw(:string) に対して，証明すべき命題の設定である．2 行目がコラボレーション適用前の状態に対する述語，5 行目がコラボレーション適用後の状態に対 3,4 行目は，コラボレーションの適用による状態の更新を意味する．と意味的には，自然言語で書くと

「任意の fsm(:fwSytemManeger)，sotre(:store)，ID(:string)，pw(:string) に対し，ID”root”がロック状態でないないならば，
コラボレーション LOGINMANAGER_login が実行された後の状態では，
ID”root”がロック状態でない」

となる．

ただし，**この命題は証明できない**．確かに，LOGINMANAGER_login では，ROOT は何度失敗してもロックされない．しかしながら，システムの実装が以下の様になっているためである．

- 関数 newLoginManager によって，クラス loginManager のオブジェクト生成する際に，ID が”root”でロール ROOT であるアドミニストレータが生成され，登録される．
- 関数 newFwSystemManager によって，クラス fwSystemManager のオブジェクト生成する際に，クラス loginManager のオブジェクトが生成される．

このため，任意の fms が関数 newFwSystemManager によって生成されたものだとは限らないので，ID”root”のロールが ROOT であるという前提条件が不足し，この命題は証明できなかった．

よって，次のような命題を証明する．

```
01: g `!fsm store ID pw.
02:  (Inv_rootIsUnlock fsm store) /\ (Inv_rootIsROOT fsm store)  ==>
03:      let lm =  fwSystemManager_get_loginM fsm store    in
04:      let newS   = SND(LOGINMANAGER\_login lm ID pw store)  in
05:      ((Inv_rootIsUnlock fsm newS) /\ (Inv_rootIsROOT fsm newS))`;;
```

Inv_rootIsROOT は，ID"root"のロールが ROOT であるという述語である．
既に Inv_rootIsROOT がシステム上で常に成り立つことが証明済みならば，
Inv_rootIsROOT 前提として用いるだけで，証明対象とする必要はない．ここでは，
Inv_rootIsROOT については，証明されていないので，Inv_rootIsUnlock，Inv_rootIsROOT
を同時に証明する．

この命題はに対する証明を行った結果，証明は成功した．これにより，

**「任意の fsm(:fwSytemManeger)，sotre(:store)，ID(:string)，pw(:string) に対し，
ID"root"がロック状態でない，かつ，ID"root"のロールが ROOT ならば，
コラボレーション LOGINMANAGER_login 実行された後の状態では，
ID"root"がロック状態でない，かつ，ID"root"のロールが ROOT である」**

ことが証明された．

5.3.2 初期段階の証明

次に初期段階について証明する．システムの初期状態を与える関数は，
newFwSystemManager であり，初期段階の命題の設定は以下のようになる．

```
01: g `!store.
02:     let (fsm,s) =  newFwSystemManager  store  in
03:     ((Inv_rootIsUnlock fsm s) /\ (Inv_rootIsROOT fsm s))`;;
```

この命題を証明することにより，

**「任意の sotre(:store) に対し，システムを生成した状態（初期状態）では，ID"root"が
ロック状態でない，かつ，ID"root"のロールが ROOT である」**

ことが証明された．

5.3.3 演繹的性質によるの結論

以上，初期段階と帰納段階が証明されたことにより，

「任意の fsm(:fwSytemManeger)，sotre(:store)，ID(:string)，pw(:string) に対し，LOGINMANAGER_login を繰り返し実行しても，ID"root"は常にロック状態でない（，かつ，ロールは ROOT である)」

ことが証明された．尚，この証明の Tactics の手順については，付録 D.2 を参照されたい．

5.3.4 証明の一般化と他の不変表明

ここでは，コラボレーション LOGINMANAGER_login に注目し，証明を行った．これと同様に，ACMANAGER_FwAdminLockoutConfig，ACMANAGER_AuditorLockoutConfig，更に他のすべて帰納段階となるのコラボレーションについて同様に証明することで，

「システムのすべての状態で，ID"root"は常にロック状態でない（，かつ，ロールは ROOT である)」

ことが証明される．

以上のような流れで，不変表明は証明される．不変表明として扱われる命題は，システムのあらゆる状態で成り立っていなければならない事象である．

不変表明で表される内容とその命題は，セキュリティの目的である秘匿性，完全性および可用性を維持の中の，主に完全性に当てはまる．つまり，この証明にり，完全性の維持についての部分的証明が得られたといえる．

また，これと同様な方法で他にも以下のような命題が証明可能である，

- アドミニストレータは，
  - アンロック状態，かつ，認証の連続失敗回数が規定値より少ない
  - ロック状態，かつ，認証の連続失敗回数が規定値以上

  のどちらかである
- 監査記録の量が，規定の監査ファイルのサイズを超えていない
- 監査ファイルの量が，規定の監査ストレージのサイズを超えていない
- アクセスコントロールの対応表が，定められたものである（変化しない）

5.4　証明例2　コラボレーション機能の証明

不変表明による証明は，システムの状態に注目したものであったが，ここではシステムの機能に注目した証明について述べる．各機能が正しく機能するかという内容の検証は，実装システムの各操作について，前提（引数）と結果（返り値）が仕様書のとおり (必要十分) の対応になっているかをチェックする．

例として，コラボレーション (PFMANAGER_packetFiltering pfm packet store) について述べる．

5.4.1　パケットフィルタリングについての仕様書の要求

仕様書におけるパケットのパス・ドロップの判断では，以下の場合にパケットをパスすることになる．

1. パケットフィルタリングファンクションがオン状態である
2. FW サーバに送られてきたパケットではない
3. FW サーバから送られるパケットではない
4.
   - 既に許可されたコネクションのパケットである

   または

   - 既に許可されたコネクションのパケットではない
     (a) パケットフィルタリングのルールが定義されている
     (b) パケットフィルタリングのルールに合致している

また，返り値のパケットは，

- パスした場合，ネットワークアドレス変換されたパケット
- ドロップした場合，パケットの null オブジェクト

となる．

5.4.2　命題の設定と証明

これらを，すべて述語にしたものを以下のようにする．各述語の定義内容は省略する．

1．P_pfFunRun

2．¬P_isSentToFw

3. ¬P_isSentFromFw

4.
   - ¬P_isNewConnection
   - P_isNewConnection
     (a) P_pfRuleDefined
     (b) P_matchWithPfRule

ネットワークアドレス変換されたパケットであるという述語を

- P_isNATed

とする.

ここで，パスする場合の述語を P1，P2 にまとめると

- P1 fsm packet store =

```
let pfm =  fwSystemManager_get_pfM fsm store   in
        (P_pfFunRun pfm store)
        /\ (P_isSentToFw pfm packet store)
        /\ (P_isSentFromFw pfm packet store)
        /\ (~P_isNewConnection pfm packet store)
```

- P2 pfm packet store =

```
let pfm =  fwSystemManager_get_pfM fsm store   in
        (P_pfFunRun pfm store)
        /\ (P_isSentToFw pfm packet store)
        /\ (P_isSentFromFw pfm packet store)
        /\ (P_isNewConnection pfm packet store)
        /\ (P_pfRuleDefined pfm store)
        /\ (let s = PFMANGER_dstNAT pfm packet store in
            P_matchWithPfRule packet s)
```

となる．このほかに証明に必要となる前提条件を P3 として，命題は以下のように記述される.

パケットがパスする場合

```
01: g `!fsm store packet.
02:     let pfm =  fwSystemManager_get_pfM fsm store   in
03:     ((P1 pfm packet store) \/ (P2 pfm packet store))
04:     /\ (P3 fsm packet store) ==>
05:       let (returnPacket,newS) =
```

```
06:                SND(PFMANAGER_packetFiltering pfm packet store)  in
07:             P_isNATedPacket pfm packet returnPacket store‘;;
```

パケットがドロップする場合

```
01: g ‘!fsm store packet.
02:     let pfm =  fwSystemManager_get_pfM fsm store    in
03:     ~(((P1 pfm packet store) \/ (P2 pfm packet store))
04:       /\ (P3 fsm packet store)) ==>
05:       let (newPacket,newS) =
06:              SND(PFMANAGER_packetFiltering pfm packet store)  in
07:           returnPacket = packet_null‘;;
```

5.4.3 結論

上の 2 つの命題を証明することで，前提と結果の対応が必要十分であることが証明され，PFMANAGER_packetFiltering が正しく機能することが証明できる．

同様に，各関数・コラボレーションも前提と結果が必要十分であることを示すことで，正しく機能することを証明できる．各関数が正しく機能することの証明は，セキュリティの目的である秘匿性，完全性および可用性を維持の中の，主に完全性に当てはまる．つまり，この証明にり，完全性の維持についての部分的証明が得られたといえる．

5.5 証明例3　アクセスコントロールについての証明

ここでは，アクセスコントロールについての証明を行う．

5.5.1 アクセスコントロールについての仕様書の要求

仕様書においては，アクセスコントロールについては，識別認証およびアクセスコントロールにおいて，以下の要件が求められている．

- **Rule-I&A-1**
  TOE に同時にアクセスできるアドミニストレータ（FW 管理者または監査役）は，一人のみである．アドミニストレータは，TOE の操作を使用する前に，識別・認証される．

- **Rule-AC-1 to Rule-AC-11**
  各ロールは，それぞれに使える操作が限定される．詳しくは付録を参照されたい．

5.5.2 アクセスコントロールの実装方法

実装システムでは，アドミニストレータが使用できる操作の一般形は以下のようになる．

```
01: ACMANAGER_OperationA ... =
02:   let  permission = isAccessibleToTheFunction ... operationA'ID ... in
03:     if permission then
04:         PFMANAGER_operationA ...(具体的操作の実行)
05:     else
06:         (何も起こらない)...
```

ログインマネージャは，属性としてして現在ログインしているアドミニストレータのオブジェクトを保持している．関数 isAccessibleToTheFunction は，ログインしているアドミニストレータのロールと操作の対応について判断を下す．

```
isAccessibleToTheFunction  fsm  operations'ID  store  -> bool
(fsm:fwSytemManager) (operations'ID:int) (store:store)
```

これは，fwSytemManager 型のオブジェクト，使用したい操作の ID，システムの状態を引数にとり，実際の操作を呼ぶ出すか判断する bool 値を返す関数であり，また，述語である．具体的には，以下の 2 点が満たされるとき，返り値は真となる．

1. ログインしているマネージャが存在する
2. ログインしているマネージャのロールに，使用対象の関数 ID が割り当てられている

5.5.3 命題の設定と証明

以上を踏まえ，仕様書の要求を満たすには，

1. ログインしているアドミンが正当なものである
   - ログイン，ログアウトが正しく機能する
   - ログイン，ログアウト以外の操作で，ログインしているアドミニストレータの変更が行われない
2. isAccessibleToTheFunction が，正しく機能する

を示せばよい.

これらを示すために，それぞれ以下の方針で命題の設定と証明行う.

1. 「ログインしているアドミンが正当なものである」について

   - 「ログイン，ログアウトが正しく機能する」については，検証例 2 で示したように，それぞれの関数の機能が正しいことを証明する.
   - 「ログイン，ログアウト以外の操作で，ログインアドミニストレータの変更が行われない」については，ログイン・ログアウト以外の操作について,「操作前と後で，該当のオグジェクトとは変化しない (操作前後でのオブジェクトは等しい)」ということを証明する．それぞれの証明は，検証例 2 で示したように，関数の機能が正しいことを証明することになるが，これは不変表明的性質も持っており，演繹的証明の帰納段階ともなる.

   これらにより演繹的に,「ログインしているアドミンが正当なものである」といえる.

2. 「`isAccessibleToTheFunction` が，正しく機能する」については，検証例 2 で示したように証明する.

5.5.4 結論

以上より，演繹的に，本システムのアクセスコントロール機能は，仕様書のアクセスコントロールに関する要求を満たしていることが証明できる.

アクセスコントロール機能の証明は，セキュリティの目的である秘匿性，完全性および可用性を維持の中の，主に秘匿性・可用性に当てはまる．つまり，この証明にり，秘匿性・可用性の維持についての部分的証明が得られたといえる.

第6章　考察

6.1　命題の設定について

検証内容の設定，命題の設定は，定理証明による形式検証の最も重要で本質的な問題である．セキュリティの世界標準 ISO15408（JIS 標準（JIS X 5070））の情報技術セキュリティ評価のコモンクライテリア（CC）は，様々な IT 製品に対応できる幅広い基準を持っている．プロテクションプロファイル（CC），セキュリティーターゲット（ST）は，CC の内容を元に製作され，また，CC の評価基準により，公的機関から認可を受ける設計仕様書である．つまり，ST 内容を実現しているモデルは，セキュリティが守られているモデルだといえる．本研究では，形式化されたモデルが，形式化以前の仕様書の内容を満たすかどうかを検証することを目的としている．そのためには，ST の内容を正しく表現することのできる形式化の言語が必要である．

状態遷移ベースの検証法では，検証対象をすべて状態遷移図に落とし込み網羅的に検証をする．これは自動検証が可能で，不具合がある箇所を特定することもできる．しかしながら，複数のオブジェクトの協調動作を状態遷移図に落とし込むのは難しく，状態爆発が起こる場合は検証が不可能である．

本研究で用いたオブジェクト指向理論では，合成システムの無限の状態をひとつの状態変数として扱っており，任意のオブジェクト数に対応することができる．また，表現力が豊かな論理式による記述により，命題の設定も非常に多岐にわたることができる．これにより，検証対象となる仕様書の内容を，状態遷移図ではなく，論理式という形式に落とし込む．論理式は表現力が豊かで，仕様書の内容を自然言語に近い形で命題として設定ができ，証明できることが期待される．本研究では，定理証明の一般的証明方法である不変表明の演繹的証明の他に，いくつかの命題の立て方を提案した．しかし，本研究で提案した証明方法だけでは，自然言語で書かれる仕様書の内容の検証について，十分であるとはいえない．「論理式で何を表現できるか」という部分については，多くの課題を残したままである．本研究での命題は ST に応じて設定したが，ST の記載事項のほとんどは CC の内容を継承している．CC に対する命題の設定方法を体系化することを研究対象とするのもよいかもしれない．

6.2 証明作業について

定理証明は自動化が不可能であり，その人的，時間的コストも大きい．証明の対象とする範囲が大きくなれば，証明効率の問題が必ず出てくる．証明をより効率的に行うタクティックについての研究，半自動化の研究が必要である．

HOL での証明には，以下の 2 つの証明方法がある．

1. Forward Proof
    - 推論規則を用いて，正しい事実（定理）から，新しい事実（定理）を導いていき，最終的に目的の定理を導く．
2. Goal-Oriented Proof
    - 目的の論理式 (Goal) を，(複数の）副目的（Sub goals）に分解する．
    - 最終的に，すべての Sub goals が正しければ証明は完了．
    - Tactics により，論理式を Sub goals に分解する．

一般的に，証明作業は Goal-Oriented Proof が使われており，本研究でも Goal-Oriented Proof による証明を行った．証明は，まず命題を設定し，それらを Tactics を用いて sub goal に分解して行きそのすべてを証明する．基本的には，命題の内容をすべて基本演算子が見える形まで分化し，基本演算子の公理を用いて解く．それぞれの Tactics の適用は，人が選択しなければならず，これにより人的コストが非常に大きくなる．

コラボレーションは，基本的に複数の関数を合成したものであり，それらの関数も別の関数の合成したものである場合が多い．コラボレーションの規模が大きくなればなるほど，その展開したものは膨れ上がり，証明作業も複雑化する．複雑なコラボレーションでは，Goal の展開式が大きくなりすぎると，HOL の計算時間が非常に大きくなる．複雑なコラボレーションでは，if 文など分岐がある場合は，できるだけ場合わけをし，小さな部分問題単位で解いていくのがよい．

第7章　まとめ

7.1　まとめ

情報セキュリティの確立を通じ，柔軟かつ透明であり，安全・安心で高い信頼性を有する「高信頼性社会」の実現が，経済や社会の基盤として必要不可欠である．情報セキュリティとは，具体的には，秘匿性，完全性および可用性を維持するこという．ISO15408 では，IT セキュリティ基準のセットである，コモンクライテリア (CC) を定義している．この対象はソフトウェアに限らず，IT 製品（ソフトウェア，ハードウェア，ファームウェア）となっている．この CC が示すセキュリティ要件を元に，TOE に対するプロテクションプロファイル (PP)，セキュリティターゲット (ST) が作られ，審査され，認可される．つまり，認可を受けた ST は，セキュリティの基準を満たしている設計書であるといえる．

一方，オブジェクト設計開発法の上流過程では，UML に代表されるモデリング言語によりシステムの分析モデルが構築される．システムが要求しようを満たすことを保証することを示すための，モデル検査や定理証明などの形式検証の研究が行われている．矢竹らは，オブジェクト間のコラボレーションをベースとした証明のためのオブジェクト指向理論を，定理証明器 HOL 上で理論モジュールとして構築した．

本研究の目的は，HOL 上のオブジェクト指向理論を検証の土台とし，ST で示されるセキュリティ要求，機能要求を満たすことを証明する方法についてその可能性を示すことである．このために，以下の手順で研究を進めた．

7.2　研究の流れ

本研究では，FW サーバシステムの ST の存在と，オブジェクト指向理論を定理証明器 HOL 上で理論モジュール化する技術の存在を前提とし，以下の手順で研究を進めた．

1. FW サーバシステムの ST に準ずる仕様書を想定する．
2. 仕様書に基づき UML でモデル化をし，クラス図，コラボレーション図を作る．
3. モデル化したものをオブジェクト指向理論を用いて ML 形式でコード化する．
4. ML コンパイラを用いて簡単な実行テストし，コード化で起こるようなバグを取り除く．

5. コードを HOL 形式に書き直す.

6. 仕様書の内容を証明するための，システム上の述語で命題を設定する.

7. HOL 上で演繹的な手法により命題を証明する.

これにより，証明できない場合は，モデルがセキュリティ要求，機能要求を満たさないことを示せ，モデルの不具合を指摘できる．また証明できることにより，モデルが ST で示されるセキュリティ要求，機能要求を満たすことを証明することができる．ST のので示される仕様を満たすということは，即ち，セキュリティの基準を満たしているということである．これにより，セキュリティ要求，機能要求を満たしているモデルを得ることができる.

検証例では，不変表明と演繹手法による一般的な命題の設定に加え，いくつかの命題の設定を提案し，証明した.

これらにより，モデルがいくつかのセキュリティ機能を満たしていることが示せた．しかしながら，検証には大きな人的，時間的コストが必要で，モデルが仕様書すべての要求を正しく満たしていることを示すには至らなかった.

7.3 今後の課題

ST は主に自然言語と図表などからなっている．本研究の本質的な部分は，これらの仕様書から証明すべき内容をいかに選択し，いかに論理式として表現するかである．不変表明による命題の設定と帰納法による証明は，定理証明では一般的な方法である．しかし，これだけでは，セキュリティ要求や機能要求についての命題を作るには不十分である．今後，論理式による表現方法と証明できる内容について考えなければならない.

HOL での証明の作業は，非常に大きな人的，時間的コストを費やす．いくら理論が正しく，ST の内容について正しく証明できるとしても，非現実的なコストが必要であるならば，実用化は無理である．今後，証明をより効率的に行うタクティックについての研究，自動化の研究が必要である.

謝辞

本研究を行うにあたり，北陸先端科学技術大学院大学の片山卓也教授には，本研究の目的設定から最後まで、御懇篤なご指導を頂きました.

北陸先端科学技術大学院大学の青木利晃助手には、本研究の方針に迷うときがあったとき、的確な助言をして頂きました。また、詳細の数学的、論理的な内容に関するご助言をいただきました。

北陸先端科学技術大学院大学の先輩であり、本研究の定理証明に用いた HOL 上でのオブジェクト指向理論の提唱者である矢竹健朗氏には、この技術に関しそのノウハウを一から教えていただきました。また、研究全般に関し事細かに相談に乗っていただきました。

研究室の先輩方，同輩の皆様には気軽に様々な相談にのっていただき、また基礎的な知識など多く教えていただきました.

本研究を行うにあたり，お世話になった皆様へ厚く御礼申し上げます.

参考文献

[1] 独立行政法人 情報処理推進機構　セキュリティーセンター
http://www.ipa.go.jp/security/jisec/evalbs.html

[2] HOL 4
http://hol.sourceforge.net/

[3] Kenro Yatake, Toshiaki Aoki ,Takuya Katayama コンピュータソフトウェア、Vol.22,No1(2005),p58-76. [論文]2004 年 6 月 30 日受付

[4] アクセス制御に関するセキュリティポリシーモデルの調査
独立行政法人 情報処理推進機構 セキュリティセンター
http://www.ipa.go.jp/security/fy16/reports
/access_control/documents/PolicyModelSurvey.pdf

[5] Ravi S.Sandha,Edward J Coyne,Hal L.Feinstein,CharlesE.Youman IEEE Computer, Volume 29, Number 2,February 1996,pages 38-47

[6] Indrakshi Ray,Na Li,Robert France,DaeKyoo Kim SACMAT '04, June 2-4,2004, Yorktown Heights,New York, USA.

付 録A　FWサーバ仕様書

FW サーバの仕様書は自然言語で書かれており、主に 4 つの機能に関する機能要求、セキュリティ要求が記述されている。

- パケットフィルタリングに関するポリシー
- 識別と認証に関するポリシー
- アクセスコントロールに関するポリシー
- 監査関するポリシー

各ポリシーについて説明する。
尚、ここでは対象となる FW サーバ、およびそのシステム全体を指して、TOE（target of evaluation）と呼ぶことにする。

A.1　パケットフィルタリングに関するポリシー

A.1.1　概要

TOE は、packet processing rule に従い、通過するパケットをパスするかドロップするか決定する。

A.1.2　ルール

- **Rule-PF-1**
  TOE は、通過するパケットすべてに対し、packet processing rule を適用する。
- **Rule-PF-2**
  packet processing rule は以下のようなものである。
  - パケットは、送信元 IP アドレスにより、パスするかドロップするか規定される
  - パケットは、宛先 IP アドレスにより、パスするかドロップするか規定される
  - パケットは、送信元ポート番号により、パスするかドロップするか規定される

- パケットは、宛先ポート番号により、パスするかドロップするか規定される
- パケットは、プロトコルタイプ（TCP、UDP、ACMP）により、パスするかドロップするか規定される
- パケットは、以上の規定の組み合わせにより、パスするかドロップするか規定される
- パケットは、送信元アドレスに対し、ネットワークアドレス変換（NAT:network adress translation）が行われる
- パケットは、宛先アドレスに対し、ネットワークアドレス変換（NAT:network adress translation）が行われる

- **Rule-PF-3**
  TOE は、packet processing rule が定義されていない場合、パケットをすべてドロップする

- **Rule-PF-4**
  - TOE は、パケットの宛先 IP アドレスが FW サーバ自身の IP アドレスだった場合ドロップする
  - TOE は、パケットの送信元 IP アドレスが FW サーバ自身の IP アドレスだった場合ドロップする

- **Rule-PF-5**
  - TOE は、すでに FW サーバをパスすることが許されたパケットの返信パケットを自動的にパスさせる
  - TOE は、すでに FW サーバをパスすることが許されたパケットのコネクションに属するパケットを自動的にパスさせる

- **Rule-PF-6**
  TOE は、各コネクションの最初のパケットを filter rule table 上の filter rule に照合し、以下の項目すべてに適合するか判断し、パスするかドロップするか決定される
  - 送信元 IP アドレス
  - 宛先 IP アドレス
  - 送信元ポート番号
  - 宛先ポート番号
  - プロトコルタイプ

- **Rule-PF-7**
  TOE は、内部と外部のネットワークが互いに通信できるように、FW サーバをパスすることが許されたパケットに対し、IP アドレスの変換を行う。変換と fulter rule との照合は以下の順で行われる。

  1. 宛先 IP アドレス変換
     - 各コネクションの最初のパケットを dstnat rule table 上の dstnat rule に照合する
     - 宛先 IP アドレスが適合した場合、宛先アドレスは指定された宛先アドレスに変換される
  2. filter rule と照合される。FW サーバを通過する許可が下りたものは、送信元 IP アドレス変換が行われる。
  3. 送信元 IP アドレス変換
     - FW サーバを通過する許可が下りたパケットを、srcnat rule table 上の srcnat rule に照合する
     - 送信元 IP アドレスが適合した場合、送信元 IP アドレスは指定された宛先アドレスに変換される
  4. 送信元 IP アドレス変換が完了したパケットは FW サーバを通過する。

- **Rule-PF-8**
  TOE は、filter rule に適合しなかったパケットをドロップする

- **Rule-PF-9**

  - TOE は、同じ宛先 IP アドレスに対する、秒間あたりのドロップしたパケット数が、FW 管理者が定めた規定値以上になった場合、アラートを発する
  - TOE は、同じ送信元 IP アドレスからの、秒間あたりのドロップしたパケット数が、FW 管理者が定めた規定値以上になった場合、アラートを発する

A.2 識別と認証に関するポリシー

A.2.1 概要

- TOE は、TOE にアクセスするユーザを、権限を与えられた FW 管理者、または監査役として識別・認証する

- TOE は、識別認証が成功していない場合、TOE に関する操作を使用することを許可しない

- FW 管理者・監査役に対する識別・認証は ID とパスワードの照合によって行う
- TOE は、規定回数パスワード照合に失敗した場合、そのアカウントをロックする
- TOE は、ユーザが何もせず 15 分経過した場合、アクティブセッションを終了する

A.2.2 ルール

- **Rule-I&A-1**
  TOE に同時にアクセスできるアドミニストレータ（FW 管理者または監査役）は、一人のみである。アドミニストレータは、TOE の操作を使用する前に、識別・認証される。
- **Rule-I&A-2**
  - アドミニストレータは TOE に登録され、識別・認証の対象となる
  - アドミニストレータは、以下の属性の集合により定義される
    * ID
    * パスワード
    * ロール
- **Rule-I&A-3**
  - アドミニストレータは string 型の ID により識別される
  - 識別されたアドミニストレータは、パスワードにより認証される
- **Rule-I&A-4**
  TOE は、アドミニストレータのパスワードが、規定の長さを持ち、規定された文字によって成り立っていることを保証する。
- **Rule-I&A-5**
  - TOE にアクセスしようとするユーザの不正なパスワード認証の連続回数が規定回数を超えたとアカウントはロックされる
  - ロックされたアカウントは、他のアドミニストレータが解除するか、規定時間が経過することで解除される
- **Rule-I&A-6**
  TOE は、ユーザが何もせず 15 分経過した場合、アクティブセッションを終了する
- **Rule-I&A-7**
  TOE は、アドミニストレータがパスワードを変更した際、再認証を要求する

- **Rule-I&A-8**
  TOE は、ユーザの ID をユーザが利用できる操作に関連付ける

A.3 アクセスコントロールに関するポリシー

A.3.1 概要

- TOE は、ロールに応じて FW 管理者や監査役に対し、TOE を操作するために必須の構成データを読んだり変更する能力を制限する。対象となる項目は以下のようなものである。
  - packet processing rules
  - アドミニストレータリスト
  - アカウントロック設定
  - 監査するイベントの選択
- TOE は、パケットフィルタリング機能を開始・停止する能力を FW 管理者に制限する
- TOE は、全体の TOE の操作を終了させる能力を FW 管理者のみに制限する

A.3.2 ルール

- **Rule-AC-1**
  FW 管理者、監査役は同じロールのアカウントを追加できる
- **Rule-AC-2**
  FW 管理者、監査役は同じロールで、自分自身以外のアカウントを削除できる
- **Rule-AC-3**
  - FW 管理者は、同時に監査役のロールを保持できない
  - 監査役は、同時に FW 管理者のロールを保持できない
- **Rule-AC-4**
  FW 管理者、監査役は自身のパスワードを変更できる
- **Rule-AC-5**
  FW 管理者、監査役は同じロールのアカウントのロック状態を設定できる

- **Rule-AC-6**
  TOE は、監査設定データの設定と監査イベントの選択をする能力を監査役に制限する
- **Rule-AC-7**
  TOE は、監査記録を削除する能力を監査役に制限する
- **Rule-AC-8**
  TOE は、packet processing rule を変更する能力を監査役に制限する
- **Rule-AC-9**
  TOE は、パケットフィルタリング機能を開始・停止・変更する能力を FW 管理者に制限する
- **Rule-AC-10**
  TOE は、監査機能を変更させ能力を監査役に制限する
- **Rule-AC-11**
  TOE は、全体の TOE の操作を終了させる能力を FW 管理者のみに制限する

A.4 監査に関するポリシー

A.4.1 概要

TOE は、監査するイベントが発生した場合、監査記録を生成し、TOE のセキュアな操作を監査するための必要な情報を収集する。

A.4.2 ルール

- **Rule-Audit-1**
  - TOE、監査役が指定したイベントが発生した場合、TOE のセキュアな操作を監査するため、監査データを記録する
  - 監査記録は、そのイベントを発生させたユーザの ID に関連付けられる
- **Rule-Audit-2**
  TOE は、監査記録のストレージがいっぱいになったとき、監査役によって定められた容量を超えないように、最も古いファイルに上書きをする
- **Rule-Audit-3**
  TOE は以下に示されるような、潜在的なセキュリティ違反を発見したときに、警告メッセージを表示してアドミニストレータに知らせる。

- アドミニストレータの不正な認証の回数が、規定値を超えた場合
- TOE の稼働率を低下されるようなパケット数が来た場合
- TOE を操作するのに必須の実行ファイルや設定ファイルがダメージを受けたことを発見した場合

- **Rule-Audit-4**
  TOE は、監査記録のファイルサイズが監査役の指定したサイズに到達した場合、ファイルを循環させる

- **Rule-Audit-5**
  TOE は、監査記録のファイルを日ごとに循環させる

- **Rule-Audit-6**
  TOE は、監査記録のファイルを、監査役が定めた保存期間の保管し、期限が切れるとファイルを破棄する

- **Rule-Audit-7**
  TOE は、FE 管理者・監査役に、FW サーバのコンソールから TOE にアクセスした場合のみ、監査記録を読むことを許可する

- **Rule-Audit-8**
  TOE は監査記録の探索を以下の項目の情報によって可能なように管理する
  - ユーザの ID
  - IP アドレス
  - 日時
  - プロトコルタイプ
  - packet processing rule ID
  - 表示結果

付 録B　FWサーバシステム　UML図

ここではモデル化したFWサーバシステムのUML図を載せる。

B.1　クラス図

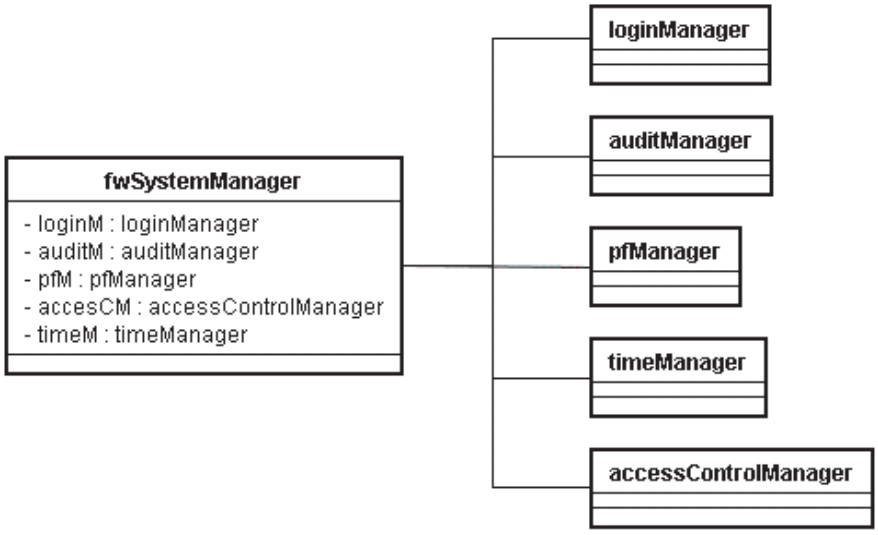

図 B.1: クラス図：FWシステム全体

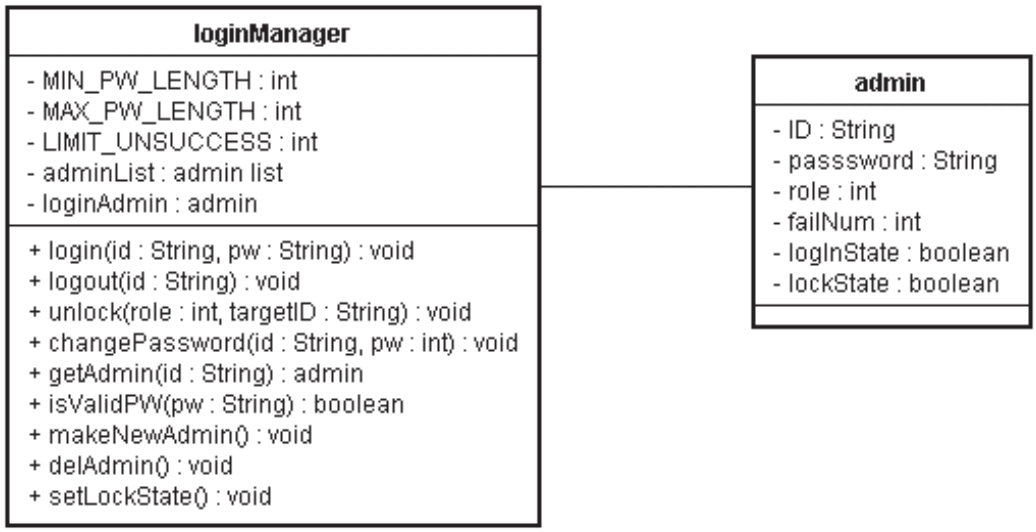

図 B.2: クラス図：識別・認証機能

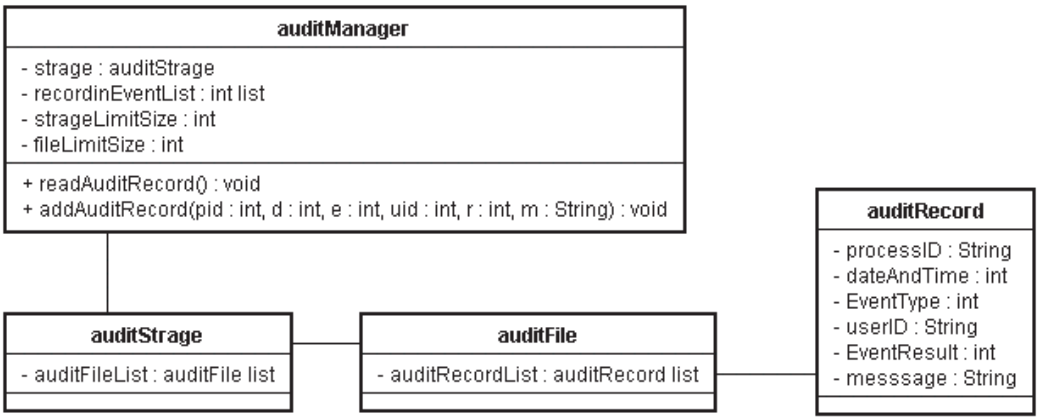

図 B.3: クラス図：監査機能

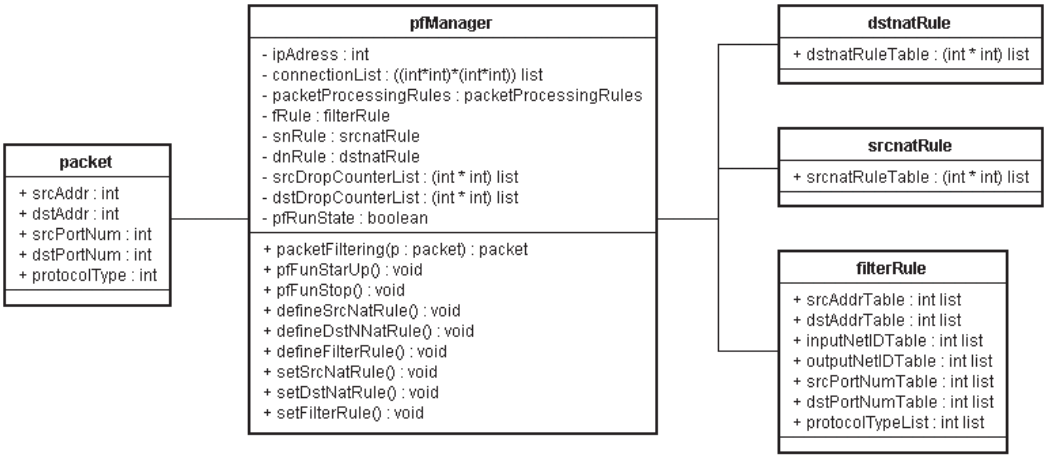

図 B.4: クラス図：パケットフィルタリング機能

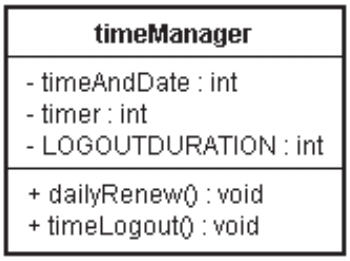

図 B.5: クラス図：時間に関する機能

図 B.6: クラス図：アクセスコントロール機能

B.2　シーケンス図

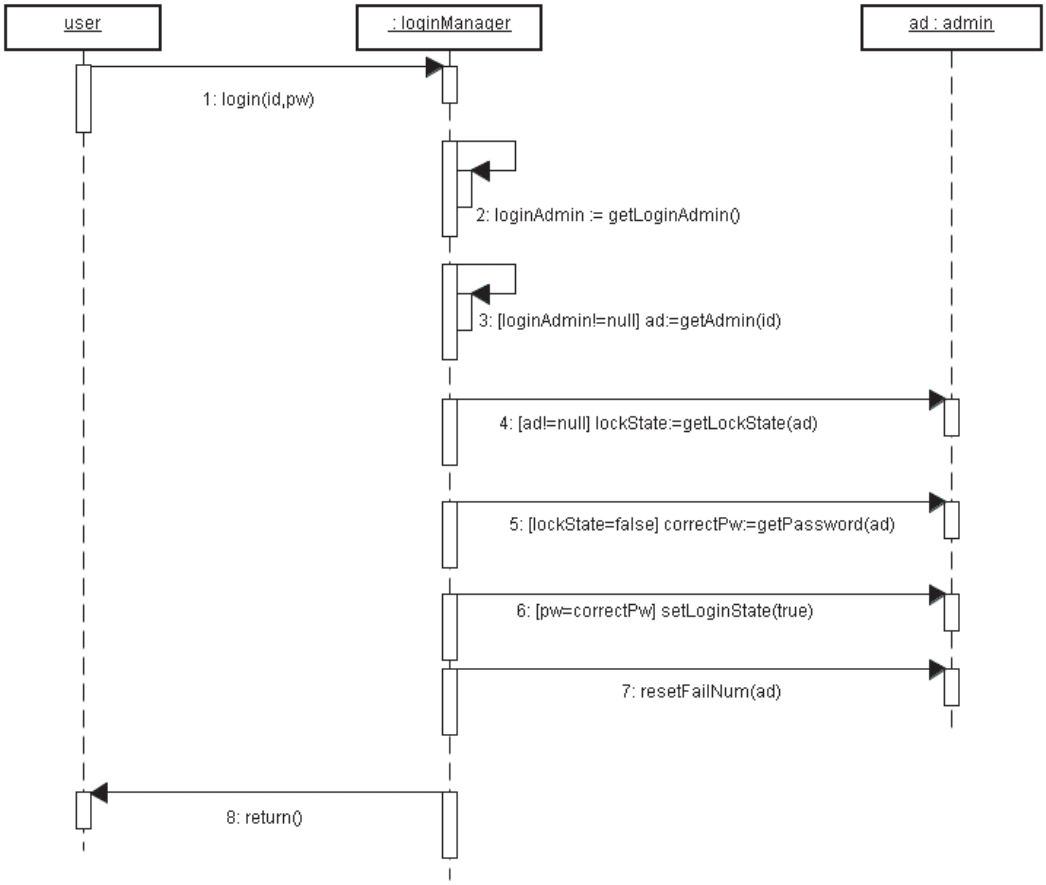

図 B.7: シーケンス図：ログイン成功

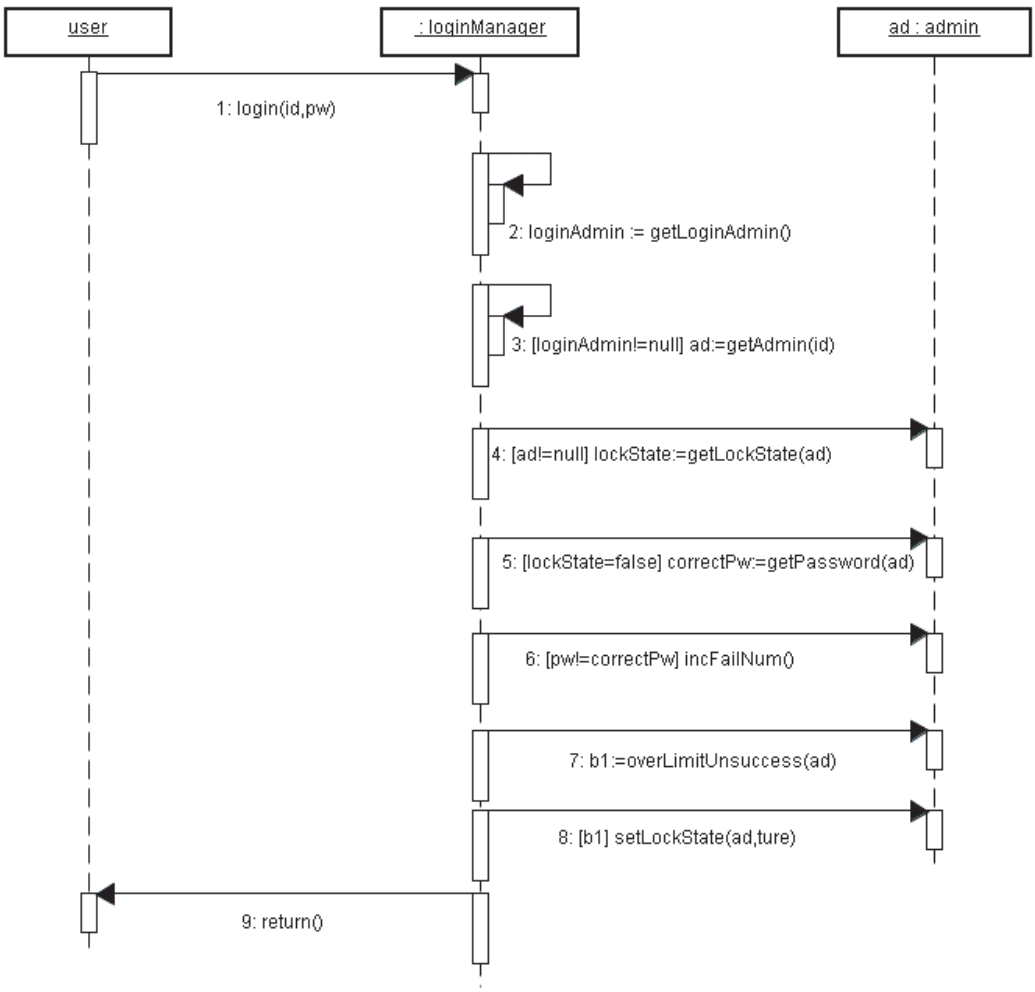

図 B.8: シーケンス図：ログイン失敗後、アカウントロック

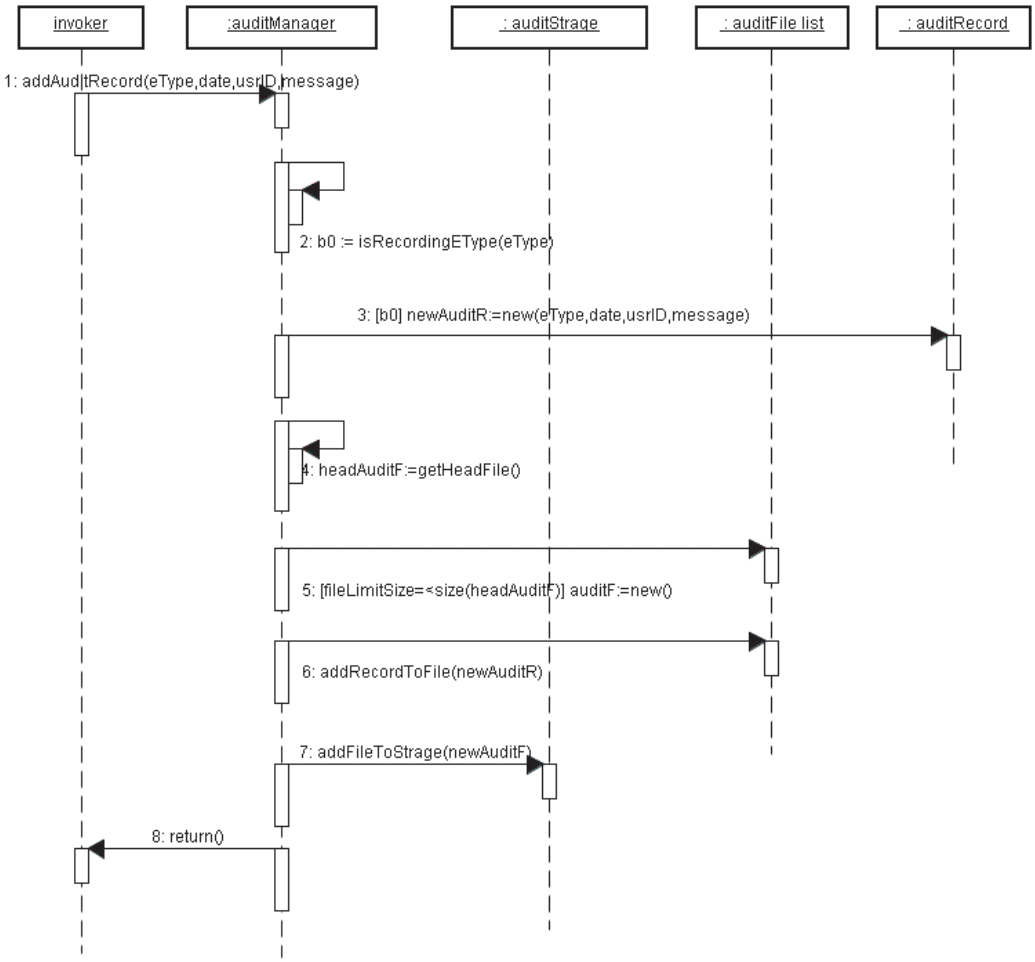

図 B.9: シーケンス図：監査記録の生成

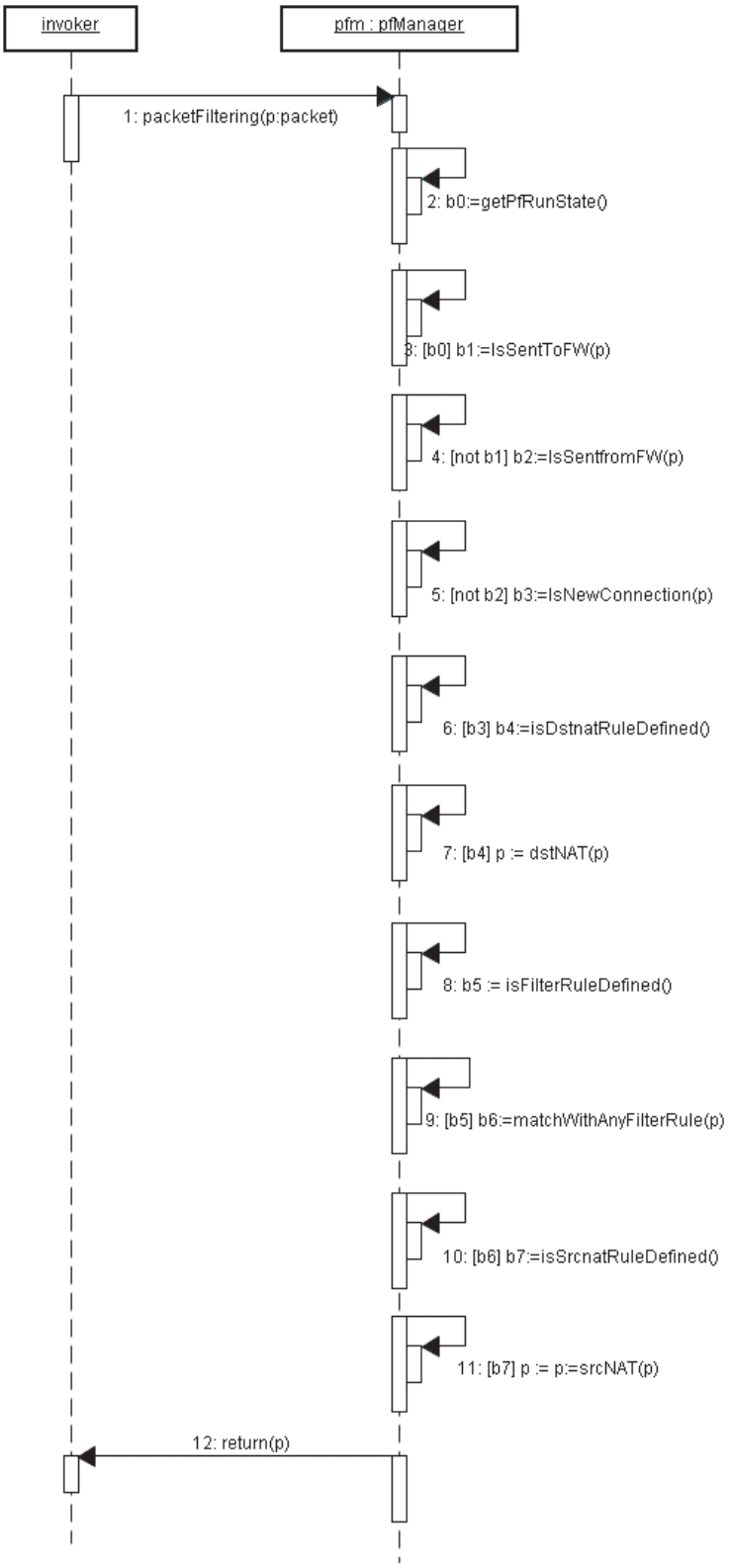

図 B.10: シーケンス図：パケットフィルタリング

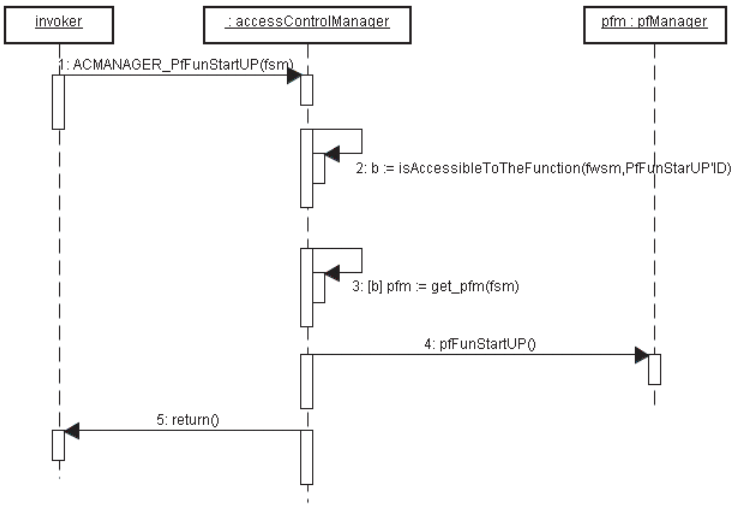

図 B.11: シーケンス図：アクセスコントロールからの、パケットフィルタリング機能の開始

付 録C　FWサーバシステムのML形式、HOL形式による記述

C.1　クラスモデル

```
system dfwSystem

class loginManager
attr MIN_PW_LENGTH    holtype : num        | 5     mltype : int         | 5
attr MAX_PW_LENGTH    holtype : num        | 10    mltype : int         | 10
attr DURATION         holtype : num        | 200   mltype : int         | 200
attr LIMIT_UNSUCCESS  holtype : num        | 3     mltype : int         | 3
attr adminList        holtype : admin list | []    mltype : admin list  | []
attr loginAdmin
     holtype : admin      | admin_null    mltype : admin  | admin_null

class admin
attr ID          holtype : string    | ""    mltype : string    | ""
attr password    holtype : string    | ""    mltype : string    | ""
attr role        holtype : num    | 0    mltype : int    | 0
attr failNum     holtype : num    | 0    mltype : int    | 0
attr loginState  holtype : bool    | F    mltype : bool    | false
attr lockState   holtype : bool    | F    mltype : bool    | false


class auditManager
attr strage
     holtype : auditStrage | auditStrage_null
     mltype  : auditStrage | auditStrage_null
attr recordingEventList  holtype : num list |[]    mltype : int list |[]
attr strageLimitSize     holtype : num      | 0    mltype : int      | 0
attr fileLimitSize       holtype : num      | 0    mltype : int      | 0

class auditStrage
attr auditFileList
     holtype : auditFile lis | []    mltype : auditFile list    | []

class auditFile
```

```
attr auditRecordList
     holtype : auditRecord list    | []    mltype : auditRecord list    | []

class auditRecord
attr eventType      holtype : num       | 0        mltype : int    | 0
attr dateAndTime    holtype : num       | 0        mltype : int    | 0
attr userID         holtype : string    | ""    mltype : string    | ""
attr message        holtype : string    | ""    mltype : string    | ""


class pfManager
attr ipAddres    holtype : num   | 0   mltype : int   | 0
attr connectionList
     holtype : ((num # num)#(num # num)) list  | []
     mltype  : ((int * int)*(int * int))list   | []
attr fRule
     holtype : filterRule | filterRule_null
     mltype  : filterRule | filterRule_null
attr snRule
     holtype : srcnatRule | srcnatRule_null
     mltype  : srcnatRule |srcnatRule_null
attr dnRule
     holtype : dstnatRule | dstnatRule_null
     mltype  : dstnatRule |dstnatRule_null
attr srcDropCounterList
     holtype : (num # num)list | []   mltype : (int * int)list | []
attr dstDropCounterList
     holtype : (num # num)list | []   mltype : (int * int)list | []
attr dosCounterLimitNum
     holtype : num     | 3   mltype : int           | 3
attr pfRunState  holtype : bool     | F   mltype : bool           | false

class packet
attr srcAddr    holtype : num    | 0    mltype : int    | 0
attr dstAddr    holtype : num    | 0    mltype : int    | 0
attr srcPortNum    holtype : num    | 0    mltype : int    | 0
attr dstPortNum    holtype : num    | 0    mltype : int    | 0
attr inputNetID    holtype : num    | 0    mltype : int    | 0
attr outputNetID    holtype : num    | 0    mltype : int    | 0
attr protocolType    holtype : num    | 0    mltype : int    | 0

class filterRule
attr srcAddrTable   holtype : num list    | []    mltype : int list   | []
attr dstAddrTable   holtype : num list    | []    mltype : int list   | []
attr srcPortNumTable   holtype : num list   | []    mltype : int list   | []
attr dstPortNumTable   holtype : num list   | []    mltype : int list   | []
```

```
attr inputNetIDTable    holtype : num list   | []    mltype : int list   | []
attr outputNetIDTable   holtype : num list   | []    mltype : int list   | []
attr protocolTypeTable
     holtype : num list   | []    mltype : int list   | []

class dstnatRule
attr dstnatRuleTable
     holtype : ((num # num)#(num # num)) list  | []
     mltype  : ((int * int)*(int * int))list   | []

class srcnatRule
attr srcnatRuleTable
     holtype : ((num # num)#(num # num)) list  | []
     mltype  : ((int * int)*(int * int))list   | []


class accessControlManager
attr accessControlRuleTable
     holtype : (num # num list) list  | []
     mltype  : (int * int list)list   | []


class timeManager
attr timeAndDate holtype : num   | 0  mltype : int   | 0
attr timer       holtype : num   | 0  mltype : int   | 0
attr LOGOUTDURATION  holtype : num   | 5   mltype : int      | 5


class fwSystemManager
attr loginM
     holtype :  loginManager | loginManager_null
     mltype  :   loginManager | loginManager_null
attr auditM
     holtype :  auditManager  | auditManager_null
     mltype  :  auditManager  | auditManager_null
attr pfM
     holtype :   pfManager | pfManager_null
     mltype  :   pfManager | pfManager_null
attr accessCM
     holtype :   accessControlManager | accessControlManager_null
     mltype  : accessControlManager | accessControlManager_null
attr timeM
     holtype :   timeManager | timeManager_null
     mltype  :   timeManager | timeManager_null
```

C.2 関数とコラボレーション

以下はFWサーバシステムの各クラスの操作を記述した、MLの関数である。コード本体は1800行程度の内容である。ここでは定数、関数名とその型を記す。

```
''a rm = fn : ''a -> ''a list -> ''a list
LOGINMANAGER_isValidPW = fn : loginManager -> string -> store -> bool
LOGINMANAGER_exIDinAdminList = fn :
  loginManager -> string -> store -> bool
LOGINMANAGER_addAdminToList = fn :
  loginManager -> admin -> store -> store
LOGINMANAGER_newAdmin = fn :
  loginManager -> string -> string -> int -> store -> store
LOGINMANAGER_makeNewAdmin = fn :
  loginManager -> string -> string -> string -> int -> store ->
  string * bool * store
LOGINMANAGER_delAdmin = fn :
  loginManager -> string -> store -> string * store
newLoginManager = fn : store -> loginManager * store
LOGINMANAGER_getAdmin = fn : loginManager -> string -> store -> admin
LOGINMANAGER_isAdminLockout = fn : admin -> store -> bool
LOGINMANAGER_isAdminLogin = fn : admin -> store -> bool
LOGINMANAGER_isCorrectPW = fn : admin -> string -> store -> bool
LOGINMANAGER_incFailNum = fn : admin -> store -> store
LOGINMANAGER_overLimitUnsuccess = fn : loginManager -> admin -> store -> bool
LOGINMANAGER_setLoginState = fn :
  loginManager -> string -> bool -> store -> store
LOGINMANAGER_setLockState = fn :
  loginManager -> string -> bool -> store -> store
LOGINMANAGER_setFailNum = fn :
  loginManager -> string -> int -> store -> store
LOGINMANAGER_logout = fn :
  loginManager -> string -> store -> string * store
LOGINMANAGER_unlock = fn :
  loginManager -> string -> store -> string * store
LOGINMANAGER_login = fn :
  loginManager -> string -> string -> store -> string * store
LOGINMANAGER_changePassword = fn :
  loginManager -> string -> string -> string -> string -> store ->
  string * store

AUDITMANAGER_setNewAFList = fn : auditStrage -> store -> store
AUDITMANAGER_setNewStrage = fn :
  auditManager -> int -> int -> store -> store
AUDITMANAGER_setStrageLimitSize = fn :
  auditManager -> int -> store -> store
AUDITMANAGER_setFileLimitSize = fn :
  auditManager -> int -> store -> store
newAuditManager = fn : store -> auditManager * store
AUDITMANAGER_setRecEvent = fn :
  auditManager -> int list -> store -> store
AUDITMANAGER_addRecEvent = fn : auditManager -> int -> store -> store
```

```
AUDITMANAGER_rmRecEvent = fn : auditManager -> int -> store -> store
AUDITMANAGER_makeNewAuditRecord = fn :
  int -> int -> string -> string -> store -> auditRecord * store
AUDITMANAGER_addRecordToFile = fn :
  auditRecord -> auditFile -> store -> store
AUDITMANAGER_addAuditRecord = fn :
  auditManager -> int -> int -> string -> string -> store -> string * store
AUDITMANAGER_dailyRenewal = fn : auditManager -> store -> string * store

newPfManager = fn : int -> store -> pfManager * store
PFMANAGER_pfFunStartUP = fn : pfManager -> store -> string * store
PFMANAGER_pfFunStop = fn : pfManager -> store -> string * store
PFMANAGER_setConnectionList = fn :
  pfManager -> ((int * int) * (int * int)) list -> store -> store
PFMANAGER_addConnectionToList = fn :
  pfManager -> (int * int) * (int * int) -> store -> store
PFMANAGER_rmConnectionFromList = fn :
  pfManager -> (int * int) * (int * int) -> store -> store
PFMANAGER_addPacketConnectionToList = fn :
  pfManager -> packet -> store -> store
PFMANAGER_rmPacketConnectionInList = fn :
  pfManager -> packet -> store -> store
PFMANAGER_defineFilterRule = fn :
  pfManager -> int list -> int list -> int list -> int list -> int list ->
  int list -> int list -> store -> store
PFMANAGER_setSrcAddrTable = fn : pfManager -> int list -> store -> store
PFMANAGER_setDstAddrTable = fn : pfManager -> int list -> store -> store
PFMANAGER_setSrcPortNumTable = fn :
  pfManager -> int list -> store -> store
PFMANAGER_setDstPortNumTable = fn :
  pfManager -> int list -> store -> store
PFMANAGER_setInputNetIDTable = fn :
  pfManager -> int list -> store -> store
PFMANAGER_setOutputNetIDTable = fn :
  pfManager -> int list -> store -> store
PFMANAGER_setProtocolTypeTable = fn :
  pfManager -> int list -> store -> store
PFMANAGER_addSrcAddrToTable = fn : pfManager -> int -> store -> store
PFMANAGER_addDstAddrToTable = fn : pfManager -> int -> store -> store
PFMANAGER_addSrcPortNumToTable = fn : pfManager -> int -> store -> store
PFMANAGER_addDstPortNumToTable = fn : pfManager -> int -> store -> store
PFMANAGER_addInputNetIDToTable = fn : pfManager -> int -> store -> store
PFMANAGER_addOutputNetIDToTable = fn : pfManager -> int -> store -> store
PFMANAGER_addProtocolTypeToTable = fn :
  pfManager -> int -> store -> store
PFMANAGER_rmSrcAddrFromTable = fn : pfManager -> int -> store -> store
PFMANAGER_rmDstAddrFromTable = fn : pfManager -> int -> store -> store
PFMANAGER_rmSrcPortNumFromTable = fn : pfManager -> int -> store -> store
PFMANAGER_rmDstPortNumFromTable = fn : pfManager -> int -> store -> store
PFMANAGER_rmInputNetIDFromTable = fn : pfManager -> int -> store -> store
PFMANAGER_rmOutputNetIDFromTable = fn :
  pfManager -> int -> store -> store
PFMANAGER_rmProtocolTypeFromTable = fn :
```

```
  pfManager -> int -> store -> store
PFMANAGER_isFilterRuleDefined = fn : pfManager -> store -> bool
PFMANAGER_matchWithAnyFilterRule = fn :
  pfManager -> packet -> store -> bool
PFMANAGER_defineSrcnatRule = fn :
  pfManager -> ((int * int) * (int * int)) list -> store -> store
PFMANAGER_isSrcnatRuleDefined = fn : pfManager -> store -> bool
PFMANAGER_setSrcnatRule = fn :
  pfManager -> ((int * int) * (int * int)) list -> store -> string * store
PFMANAGER_addSrcnatRule = fn :
  pfManager -> (int * int) * (int * int) -> store -> string * store
PFMANAGER_rmSrcnatRule = fn :
  pfManager -> (int * int) * (int * int) -> store -> string * store
PFMANAGER_defineDstnatRule = fn :
  pfManager -> ((int * int) * (int * int)) list -> store -> store
PFMANAGER_isDstnatRuleDefined = fn : pfManager -> store -> bool
PFMANAGER_setDstnatRule = fn :
  pfManager -> ((int * int) * (int * int)) list -> store -> string * store
PFMANAGER_addDstnatRule = fn :
  pfManager -> (int * int) * (int * int) -> store -> string * store
PFMANAGER_rmDstnatRule = fn :
  pfManager -> (int * int) * (int * int) -> store -> string * store
newPacket = fn :
  int -> int -> int -> int -> int -> int -> int -> store -> packet * store
PFMANAGER_isSentToFW = fn : pfManager -> packet -> store -> bool
PFMANAGER_isSentFromFW = fn : pfManager -> packet -> store -> bool
PFMANAGER_isNewConnection = fn : pfManager -> packet -> store -> bool
PFMANAGER_isDstnatRuleDefined = fn : pfManager -> store -> bool
PFMANAGER_matchWithAnyDstNatRule = fn :
  pfManager -> packet -> store -> bool
PFMANAGER_dstNAT = fn : pfManager -> packet -> store -> store
PFMANAGER_isSrcnatRuleDefined = fn : pfManager -> store -> bool
PFMANAGER_matchWithAnySrcNatRule = fn :
  pfManager -> packet -> store -> bool
PFMANAGER_srcNAT = fn : pfManager -> packet -> store -> store
('a, 'b) alert = fn : 'a -> 'b -> 'b
PFMANAGER_clearCounterList = fn : pfManager -> store -> store
''a PFMANAGER_incCountL = fn :
  ''a -> (''a * int) list -> (''a * int) list
PFMANAGER_addDropCounter = fn : pfManager -> packet -> store -> store
PFMANAGER_isOverDstDoropCounterList = fn : pfManager -> store -> bool
PFMANAGER_isOverSrcDoropCounterList = fn : pfManager -> store -> bool
PFMANAGER_packetFiltering = fn :
  pfManager -> packet -> store -> string * packet * store

newTimeManager = fn : store -> timeManager * store
TIMEMANAGER_timeLogout = fn :
  timeManager -> loginManager -> store -> string * store
TIMEMANAGER_incTime = fn :
  timeManager -> loginManager -> store -> string * store
TIMEMANAGER_resetTimer = fn : timeManager -> store -> store
TIMEMANAGER_dailyRenew = fn : auditManager -> store -> string * store
```

```
newACManager = fn : store -> accessControlManager * store
ACMANAGER_isAccessibleToTheFunction = fn :
  fwSystemManager -> string -> int -> store -> string * bool
ACMANAGER_auditRecord = fn :
  fwSystemManager -> string -> int -> string -> store -> string * store
ACMANAGER_PfFunStartUP = fn :
  fwSystemManager -> string -> store -> string * store
ACMANAGER_PfFunStop = fn :
  fwSystemManager -> string -> store -> string * store
ACMANAGER_FwAdminAdd = fn :
  fwSystemManager -> string -> string -> string -> string -> store ->
  string * bool * store
ACMANAGER_FwAdminDel = fn :
  fwSystemManager -> string -> string -> store -> string * store
ACMANAGER_AuditorAdd = fn :
  fwSystemManager -> string -> string -> string -> string -> store ->
  string * bool * store
ACMANAGER_AuditorDel = fn :
  fwSystemManager -> string -> string -> store -> string * store
ACMANAGER_UnlockFwAdmin = fn :
  fwSystemManager -> string -> string -> store -> string * store
ACMANAGER_UnlockAuditor = fn :
  fwSystemManager -> string -> string -> store -> string * store
ACMANAGER_PwChange = fn :
  fwSystemManager -> string -> string -> string -> string -> store ->
  string * store
ACMANAGER_ReadAuditRecords = fn :
  fwSystemManager -> string -> store -> string * store
ACMANAGER_AuditFileSizeConfig = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_AuditStrageSizeConfig = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_SelectAuditableEvents = fn :
  fwSystemManager -> string -> int list -> store -> string * store
ACMANAGER_DefineSrcNatRule = fn :
  fwSystemManager -> string -> ((int * int) * (int * int)) list -> store ->
  string * store
ACMANAGER_SetSrcNatRule = fn :
  fwSystemManager -> string -> ((int * int) * (int * int)) list -> store ->
  string * store
ACMANAGER_AddSrcNatRule = fn :
  fwSystemManager -> string -> (int * int) * (int * int) -> store ->
  string * store
ACMANAGER_RmSrcNatRule = fn :
  fwSystemManager -> string -> (int * int) * (int * int) -> store ->
  string * store
ACMANAGER_DefineDstNatRule = fn :
  fwSystemManager -> string -> ((int * int) * (int * int)) list -> store ->
  string * store
ACMANAGER_SetDstNatRule = fn :
  fwSystemManager -> string -> ((int * int) * (int * int)) list -> store ->
  string * store
ACMANAGER_AddDstNatRule = fn :
```

```
  fwSystemManager -> string -> (int * int) * (int * int) -> store ->
  string * store
ACMANAGER_RmDstNatRule = fn :
  fwSystemManager -> string -> (int * int) * (int * int) -> store ->
  string * store
ACMANAGER_DefinePFRule = fn :
  fwSystemManager -> string -> int list -> int list -> int list -> int list ->
  int list -> int list -> int list -> store -> string * store
ACMANAGER_SetSrcAddrTable = fn :
  fwSystemManager -> string -> int list -> store -> string * store
ACMANAGER_SetDstAddrTable = fn :
  fwSystemManager -> string -> int list -> store -> string * store
ACMANAGER_SetSrcPortNumTable = fn :
  fwSystemManager -> string -> int list -> store -> string * store
ACMANAGER_SetDstPortNumTable = fn :
  fwSystemManager -> string -> int list -> store -> string * store
ACMANAGER_SetInputNetIDTable = fn :
  fwSystemManager -> string -> int list -> store -> string * store
ACMANAGER_SetOutputNetIDTable = fn :
  fwSystemManager -> string -> int list -> store -> string * store
ACMANAGER_SetProtocolTypeTable = fn :
  fwSystemManager -> string -> int list -> store -> string * store
ACMANAGER_AddSrcAddrToTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_AddDstAddrToTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_AddSrcPortNumToTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_AddDstPortNumToTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_AddInputNetIDToTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_AddOutputNetIDToTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_AddProtocolTypeToTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_RmSrcAddrFromTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_RmDstAddrFromTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_RmSrcPortNumFromTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_RmDstPortNumFromTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_RmInputNetIDFromTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_RmOutputNetIDFromTable = fn :
  fwSystemManager -> string -> int -> store -> string * store
ACMANAGER_RmProtocolTypeFromTable = fn :
  fwSystemManager -> string -> int -> store -> string * store

newFwSystemManager = fn : store -> string * fwSystemManager * store
```

付 録D　HOLによる証明例

D.1　section3.5.3の証明作業

- **Goalの設定をする.**

```
g `! p store . (P_personOn_1_1 p store)/\
                (P_exPerson p store)  ==>
                let newStore =  person_move p 0 0 store  in
                    (P_personOn_1_1 p newStore) `;;

- - > val it =
    Proof manager status: 1 proof.
    1. Incomplete:
         Initial goal:
         !p store.
           P_personOn_1_1 p store /\ P_exPerson p store ==>
           (let newStore = person_move p 0 0 store in
              P_personOn_1_1 p newStore)


     : proofs
```

- **全称限定!を取り除く.**

```
e (REPEAT GEN_TAC);;

- - OK..
1 subgoal:
> val it =
    P_personOn_1_1 p store /\ P_exPerson p store ==>
    (let newStore = person_move p 0 0 store in P_personOn_1_1 p newStore)

     : goalstack
```

- let を取り除く．

```
e LET_TAC;;

- - OK..
1 subgoal:
> val it =
    P_personOn_1_1 p store /\ P_exPerson p store ==>
    P_personOn_1_1 p (person_move p 0 0 store)

    : goalstack
```

- person_move の定義を用いて置き換えを行う．

```
e (REWRITE_TAC [person_move]);;

- - OK..
1 subgoal:
> val it =
    P_personOn_1_1 p store /\ P_exPerson p store ==>
    P_personOn_1_1 p
      (let x = person_get_x p store in
         let y = person_get_y p store in
           let s0 = person_set_x p (x + 0) store in
             let s1 = person_set_y p (y + 0) s0 in s1)

    : goalstack
```

- let を取り除く．

```
e LET_TAC;;

- - OK..
1 subgoal:
> val it =
    P_personOn_1_1 p store /\ P_exPerson p store ==>
    P_personOn_1_1 p
      (person_set_y p (person_get_y p store + 0)
         (person_set_x p (person_get_x p store + 0) store))

    : goalstack
```

- P_personOn_1_1 **の定義を用いて置き換えを行う．**

```
e (REWRITE_TAC [P_personOn_1_1]);;

- - OK..
1 subgoal:
> val it =
    (let x = person_get_x p store in
       let y = person_get_y p store in (x = 1) /\ (y = 1)) /\
    P_exPerson p store ==>
    (let
       x =
       person_get_x p
         (person_set_y p (person_get_y p store + 0)
            (person_set_x p (person_get_x p store + 0) store)) in
       let
         y =
         person_get_y p
           (person_set_y p (person_get_y p store + 0)
              (person_set_x p (person_get_x p store + 0) store)) in
         (x = 1) /\ (y = 1))

     : goalstack
```

- let **を取り除く．**

```
e LET_TAC;;

- - OK..
1 subgoal:
> val it =
    ((person_get_x p store = 1) /\ (person_get_y p store = 1)) /\
    P_exPerson p store ==>
    (person_get_x p
       (person_set_y p (person_get_y p store + 0)
          (person_set_x p (person_get_x p store + 0) store)) =
     1) /\
    (person_get_y p
       (person_set_y p (person_get_y p store + 0)
          (person_set_x p (person_get_x p store + 0) store)) =
```

```
     1)

     : goalstack
```

- P_exPerson の定義を用いて置き換えを行う．

```
e (REWRITE_TAC [P_exPerson]);;

- - OK..
1 subgoal:
> val it =
    ((person_get_x p store = 1) /\ (person_get_y p store = 1)) /\
    person_ex p store ==>
    (person_get_x p
       (person_set_y p (person_get_y p store + 0)
          (person_set_x p (person_get_x p store + 0) store)) =
     1) /\
    (person_get_y p
       (person_set_y p (person_get_y p store + 0)
          (person_set_x p (person_get_x p store + 0) store)) =
     1)

     : goalstack
```

- ここで，すべて基本演算子の形となった．次に，ROT_SLICE_TAC を用いて，余分な部分を省く．

```
e ROT_SLICE_TAC;;

- - OK..
1 subgoal:
> val it =
    ((person_get_x p store = 1) /\ (person_get_y p store = 1)) /\
    person_ex p store ==>
    (person_get_x p (person_set_x p (person_get_x p store + 0) store) =
     1) /\
    (person_get_y p (person_set_y p (person_get_y p store + 0) store) = 1)

     : goalstack
```

- OBJ_SIMP_TAC を用いて，内容を整理する．

```
e OBJ_SIMP_TAC;;

- - OK..
1 subgoal:
> val it =
    ((person_get_x p store = 1) /\ (person_get_y p store = 1)) /\
    person_ex p store ==>
    (1 + 0 = 1)

     : goalstack
```

- 最後に，(1 + 0 = 1) を示す．

```
e (RW_TAC arith_ss []);;

- - OK..

Goal proved.
|- ((person_get_x p store = 1) /\ (person_get_y p store = 1)) /\
   person_ex p store ==>
   (1 + 0 = 1)

Goal proved.
|- ((person_get_x p store = 1) /\ (person_get_y p store = 1)) /\
   person_ex p store ==>
   (person_get_x p (person_set_x p (person_get_x p store + 0) store) =
    1) /\
   (person_get_y p (person_set_y p (person_get_y p store + 0) store) =
    1)

Goal proved.
|- ((person_get_x p store = 1) /\ (person_get_y p store = 1)) /\
   person_ex p store ==>
   (person_get_x p
      (person_set_y p (person_get_y p store + 0)
         (person_set_x p (person_get_x p store + 0) store)) =
    1) /\
   (person_get_y p
```

```
      (person_set_y p (person_get_y p store + 0)
         (person_set_x p (person_get_x p store + 0) store)) =
    1)

Goal proved.
|- ((person_get_x p store = 1) /\ (person_get_y p store = 1)) /\
   P_exPerson p store ==>
   (person_get_x p
      (person_set_y p (person_get_y p store + 0)
         (person_set_x p (person_get_x p store + 0) store)) =
    1) /\
   (person_get_y p
      (person_set_y p (person_get_y p store + 0)
         (person_set_x p (person_get_x p store + 0) store)) =
    1)

Goal proved.
|- (let x = person_get_x p store in
      let y = person_get_y p store in (x = 1) /\ (y = 1)) /\
   P_exPerson p store ==>
   (let
      x =
      person_get_x p
        (person_set_y p (person_get_y p store + 0)
           (person_set_x p (person_get_x p store + 0) store)) in
      let
        y =
        person_get_y p
          (person_set_y p (person_get_y p store + 0)
             (person_set_x p (person_get_x p store + 0) store)) in
        (x = 1) /\ (y = 1))

Goal proved.
|- P_personOn_1_1 p store /\ P_exPerson p store ==>
   P_personOn_1_1 p
     (person_set_y p (person_get_y p store + 0)
        (person_set_x p (person_get_x p store + 0) store))
```

```
Goal proved.
|- P_personOn_1_1 p store /\ P_exPerson p store ==>
   P_personOn_1_1 p
     (let x = person_get_x p store in
        let y = person_get_y p store in
          let s0 = person_set_x p (x + 0) store in
            let s1 = person_set_y p (y + 0) s0 in s1)

Goal proved.
|- P_personOn_1_1 p store /\ P_exPerson p store ==>
   P_personOn_1_1 p (person_move p 0 0 store)

Goal proved.
|- P_personOn_1_1 p store /\ P_exPerson p store ==>
   (let newStore = person_move p 0 0 store in P_personOn_1_1 p newStore)
> val it =
    Initial goal proved.
    |- !p store.
         P_personOn_1_1 p store /\ P_exPerson p store ==>
         (let newStore = person_move p 0 0 store in
            P_personOn_1_1 p newStore) : goalstack
- -
```

以上により，Goal は証明された．証明された内容は，適当な名前をつけて，定理として利用できる．

```
val lemma_sample = top_thm();;
drop();;
```

ここまでの処理は，まとめて以下のように書くこともできる．

```
val lemma_sample = prove
    (``! p store . (P_personOn_1_1 p store)/\
                (P_exPerson p store)  ==>
                let newStore =  person_move p 0 0 store  in
                    (P_personOn_1_1 p newStore) ``,
    EVERY
     [REPEAT GEN_TAC,
      LET_TAC,
      REWRITE_TAC [person_move],
```

```
      LET_TAC,
      REWRITE_TAC [P_personOn_1_1],
      LET_TAC,
      REWRITE_TAC [P_exPerson],
      ROT_SLICE_TAC,
      OBJ_SIMP_TAC,
      RW_TAC arith_ss []]);;
```

このように整理して保存しておき，必要なときに実行し，利用すると便利である．

D.2 section5.3.1 の証明作業

コラボレーション
LOGINMANAGER_login
に対する，不変表明

```
Inv_rootIsUnlock fsm store /\ Inv_rootIsROOT fsm store
```

の証明例を載せる．なお，簡単のため，HOL のメッセージは省略し，入力だけを記載する．

D.2.1 状態に対する述語

```
val admin_isUnlock =
    Define ‘ admin_isUnlock admin store =
                ~ admin_get_lockState admin store‘;;
val Inv_rootIsUnlock = Define
    ‘Inv_rootIsUnlock fsm store =
    fwSystemManager_ex fsm store ==>
      admin_isUnlock
       (LOGINMANAGER_getAdmin (fwSystemManager_get_loginM fsm store) "root" store)
       store‘;;
val Inv_rootIsROOT = Define
    ‘Inv_rootIsROOT fsm store =
    fwSystemManager_ex fsm store ==>
       ((admin_get_role
        (LOGINMANAGER_getAdmin (fwSystemManager_get_loginM fsm store)
        "root" store)store)
        = ROOT)‘;;
```

D.2.2 帰納段階の証明

```
g `!fsm store pw.
Inv_rootIsUnlock fsm store /\ Inv_rootIsROOT fsm store  ==>
    let lm =  fwSystemManager_get_loginM fsm store in
        Inv_rootIsUnlock fsm (SND(LOGINMANAGER_login lm "root" pw store))
       /\Inv_rootIsUnlock fsm (SND(LOGINMANAGER_login lm "root" pw store))`;;
e (REPEAT GEN_TAC);;
e LET_TAC;;
e (REWRITE_TAC [LOGINMANAGER_login]);;
e LET_TAC;;
e (RW_TAC bool_ss[]);;
e (RW_TAC list_ss[]);;
e (RW_TAC list_ss[]);;
e (RW_TAC list_ss[]);;
e (RW_TAC list_ss[]);;
e (REWRITE_TAC [LOGINMANAGER_getAdmin]);;
e (RW_TAC bool_ss []);;
e (REWRITE_TAC [admin_isUnlock]);;
e (REWRITE_TAC [Inv_rootIsROOT]);;
e ROT_SLICE_TAC;;
e (REWRITE_TAC [LOGINMANAGER_getAdmin]);;
e ROT_SLICE_TAC;;
e (REWRITE_TAC [LOGINMANAGER_exIDinAdminList]);;
e ROT_SLICE_TAC;;
e (REWRITE_TAC [EQ_SYM_RULE LOGINMANAGER_exIDinAdminList]);;
e (RW_TAC bool_ss []);;
e (ASSUM_LIST (fn thl => THM_UNDISCH_TAC (el 5 thl)));;
e (R_TAC [Inv_rootIsUnlock]);;
e (REWRITE_TAC [LOGINMANAGER_getAdmin]);;
e (R_TAC [admin_isUnlock]);;
e ROT_SLICE_TAC;;
e (RW_TAC bool_ss []);;
e (ASSUM_LIST (fn thl => THM_UNDISCH_TAC (el 7 thl)));;
e (R_TAC [LOGINMANAGER_getAdmin]);;
e (RW_TAC bool_ss []);;
e (RW_TAC list_ss []);;
e (RW_TAC list_ss []);;
e (R_TAC [Inv_rootIsUnlock]);;
e (R_TAC [admin_isUnlock]);;
e (R_TAC [incFailNum]);;
e ROT_SLICE_TAC;;
e (RW_TAC bool_ss []);;
e (ASSUM_LIST (fn thl => THM_UNDISCH_TAC (el 6 thl)));;
e (ASSUM_LIST (fn thl => THM_UNDISCH_TAC (el 3 thl)));;
e (R_TAC [Inv_rootIsROOT]);;
e (RW_TAC bool_ss []);;
```

```
e (RW_TAC list_ss []);;
e (R_TAC [Inv_rootIsUnlock]);;
e (REWRITE_TAC [Inv_rootIsROOT]);;
e (R_TAC [incFailNum]);;
e ROT_SLICE_TAC;;
e (RW_TAC bool_ss []);;
e (ASSUM_LIST (fn thl => THM_UNDISCH_TAC (el 6 thl)));;
e (ASSUM_LIST (fn thl => THM_UNDISCH_TAC (el 3 thl)));;
e (R_TAC [Inv_rootIsROOT]);;
e (RW_TAC bool_ss []);;
val lemma_rootIsUnlockAndROOT_inductStep1 = top_thm();;
drop();;
```

D.2.3 初期状態の証明

```
val lemma3 = prove
    (‘‘!store.FST(newManagers store) =
     FST (fwSystemManager_new store)‘‘,
     EVERY
     [R_TAC [newManagers],
      LET_TAC,
      LET_PAIR_TAC]);;

val initState = prove
(‘‘!store. (let (fsm,lm,s) = newManagers store in
             (Inv_rootIsROOT fsm s) /\ (Inv_rootIsUnlock fsm s))‘‘,
   EVERY [(REPEAT GEN_TAC),
          LET_TUPLE3_TAC,
          (REWRITE_TAC [lemma3]),
          REWRITE_TAC [Inv_rootIsUnlock],
          REWRITE_TAC [admin_isUnlock],
          (REWRITE_TAC [Inv_rootIsROOT]),
          (REWRITE_TAC [LOGINMANAGER_getAdmin]),
          (REWRITE_TAC [LOGINMANAGER_exIDinAdminList]),
          EVERY [REPEAT GEN_TAC,
                 R_TAC [newManagers],
                 LET_TAC,
                 LET_PAIR_TAC,
                 ROT_SLICE_TAC,
                 R_TAC [newTimeManager],
                 R_TAC [newACManager],
                 LET_TAC,
                 LET_PAIR_TAC,
                 ROT_SLICE_TAC,
                 R_TAC [newPfManager],
                 LET_TAC,
```

```
        LET_PAIR_TAC,
        ROT_SLICE_TAC,
        R_TAC [newAuditManager],
        LET_TAC,
        LET_PAIR_TAC,
        R_TAC[AUDITMANAGER_setNewStrage],
        LET_PAIR_TAC,
        R_TAC[AUDITMANAGER_setNewAFList],
        LET_PAIR_TAC,
        ROT_SLICE_TAC,
        R_TAC [newLoginManager],
        LET_TAC,
        LET_PAIR_TAC,
        R_TAC [LOGINMANAGER_newAdmin],
        LET_TAC,
        LET_PAIR_TAC,
        R_TAC [LOGINMANAGER_addAdminToList],
        LET_TAC,
        LET_PAIR_TAC,
        ROT_SLICE_TAC,
        OBJ_SIMP_TAC],
(R_TAC [rich_listTheory.SOME_EL,FILTER]),
(SIMP_TAC list_ss[]),
(OBJ_SIMP_TAC)]);;
```