修 士 論 文

FW サーバシステムの検証に関する研究

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

横川 智良

2008 年 3 月

修 士 論 文

FW サーバシステムの検証に関する研究

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　DEFAGO Xavier 教授
審査委員　青木 利晃 助教授

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

0510109 横川 智良

提出年月: 2008 年 2 月

概 要

本稿では，検証対象である FW サーバシステムのオーディット・マネージャに対して，オブジェクト指向を用いたモデリング，定理証明器を用いた検証を行いその動作の正当性を証明した.

目 次

第1章　はじめに

1.1　背景

今日，インターネットなど急速な情報インフラ網の発展により高度な情報化社会が構成されていく一方で，情報が財産となる現代，その流出が深刻な問題にとなっている．こうしたなかで従来に比べ，より信頼性の高い情報セキュリティ技術への関心が高まってきている．では，データのセキュリティを保つためには，どういった手段があるか．そのひとつに，データそのものを加工し鍵をかけ直接保護するという暗号化といった技術，またそれ以前にデータの流れをコントロールし，その流出を防ぐといったアプローチがある．そのほかにも様々な方法があるが，今回はデータの流れのコントロールについて着目し検証することとなる．本研究の検証では，オブジェクト指向モデル述語言語 UML を使いクラス図，シーケンス図を用い仕様の抽象化を行う.. 次にこれらの操作を，関数型言語 ML にて記述する．この ML のプログラムを HOL ４に渡し，システムの検証を行う．この際に，論理生成器に通すことにより ML 形式の記述で基本演算子が使えるようになり HOL 上でオブジェクト指向が使えるようになる．この検証に使用する UML：Unified Modeling Language とは統一モデリング言語であり，オブジェクト指向に基づくモデル記述を共通化したもので，オブジェクト指向分析・設計の標準表記法となっている．HOL とは公開述語論理定理証明器（システム）であり分析モデルの検証に使用することが可能である．多くのデータ型ライブラリ，および強力なデータ構築機能を備えており，システムの対象領域に存在する様々なデータ型を扱うのに適している．

1.2　目的

本研究では，組織内でのデータのセキュリティが論理的に守られていることを，定理証明により検証する．昨年までに FW：Fire Wall サーバシステムについて，ログインマネージャの検証が行われた．そこで今回は，さらにオーディットマネージャ（監査のため様々なイベントの記録を残し，TOE が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に必要な情報を収集）の各機能について検証を行う．また，昨年までの HOL での証明作業においては，大きな人的，時間的コストが必要であることも判明した，今回は，これまで通り HOL での証明作業を行うと同時に，その作業の効率化についての研究も行う．

1.3 アプローチ

検証を行うにあたっては，対象物を抽象化しモデルとすることが必要である，本研究ではオブジェクト指向理論によるモデル化を考え，検証へと繋げていく．検証環境の構築には矢竹らが開発したObjectLogicを使用し，FWサーバシステムのオーディットマネージャについて検証を行っていく．

1.4 論文の構成

２章では形式検証について述べ，加算器について検証例を示し定理照明器HOLについて触れる，３章からは研究のメインとなるオーディットマネージャについてとなる．３章で仕様とオブジェクト指向理論に基づくモデル化について述べ，４章では命題を設定し実際に検証を行う．５章今後の課題，まとめについて述べて本論文を総括する．

第2章　形式検証について

2.1　検証手法

高信頼性のシステムを得るためは，そのソフトウェアの正当性の検証を行う．検証には大きく分けて２つの方法がとられる．

- 定理証明による演繹的手法
- 有限個の状態モデルを網羅的に検査するモデル検査手法

2.2　定理証明

本研究では定理証明による手法を用い検証をおこなっていく．演繹的手法では論理体系に基づき定義した命題が正当性を満たすことを，定理や推論規則を用いて証明する．この手法では表現力の高い論理体系を用いることでより複雑なシステムを検証することが可能である．しかしながら，すべての定理を自動証明することは不可能である．このため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適用する必要が生じる．また演繹的手法では命題に誤りがある場合，どこに誤りがあるかを発見することが難しい．そのため HOL を用いた定理証明では，事前に ML を用いたシミュレーション実行を行うことにより誤りを回避する．

2.3　定理証明の流れ

1. 仕様を定義自然言語や図などで表現された仕様を論理式を用いた表現に変換する
2. 命題を設定証明したい表現について，仕様をふまえた論理式を用いて記述
3. 証明を行う命題をゴールとして前提条件から演繹的推論を行いゴールを書き換えていく

2.4 加算器における例

定理証明の簡単な例として，加算器を対象に動作の検証を行う．実際に input2 ビット output2 ビットの以下の加算器をについてモデリングを行っていく．

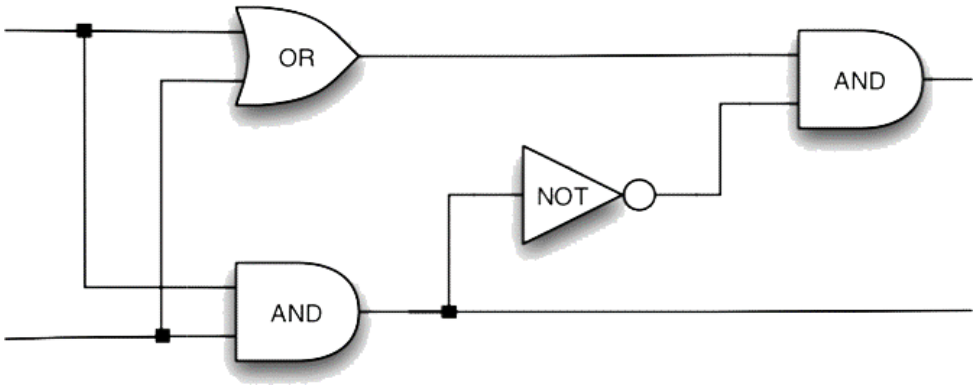

図 2.1: 加算器

1. 仕様を定義

   最初に，加算器における仕様を定義を行う．２ビットの入力をそれぞれ inp1,inp2 とおき．２ビットの出力において下位ビット out1 を上位ビットを out2 と定義したとき．out1 は inp1,inp2 がイコールの場合において T，out2 は inp1 と inp2 の論理積であると定義できる．以下にその関係を示す．

Inp1	Inp2	out1	out2	
F	F	F	F	… 0 + 0 = 00
F	T	T	F	… 0 + 1 = 01
T	F	T	F	… 1 + 0 = 01
T	T	F	T	… 1 + 1 = 10

図 2.2: 加算器の仕様

以上の定義を HOL 上で定義すると以下のコードとして記述することができる．

加算器の仕様

```
val AM1_def = Define
   ‘AM1(inp1:bool, inp2:bool) = (if (inp1=inp2) then F else T)‘;

val AM2_defdef = Define
   ‘AM2(inp1, inp2) = (inp1/\inp2)‘;
```

次に，加算器の実装を行う．実際の加算器は以下の図ような OR,AND,NOT 素子で構成されている．最初に各素子について記述する．例えば OR 素子であれば OR という値を定義する，これは OR(inp1, inp2, out) の型となる値で out は inp1 と inp2 の論理和の形であると定義される．そのほかに AND,NOT,OR についても同様の定義を行う．
そして加算器（回路）自体については AMP_IMP として AM_IMP(inp1, inp2, out1, ou2) の型をとり，inp1,inp2 の入力，out1,ou2 の出力を持つ．その他の入出力を l1,l2 として OR, 2 つの AND,NOT の値へ型の通り引数を割り振る．
さらに AM_IMP(inp1, inp2, out1, ou2) ならば各素子の状態が全て真なければならないので，全ての論理積をとったものを実装として定義する．

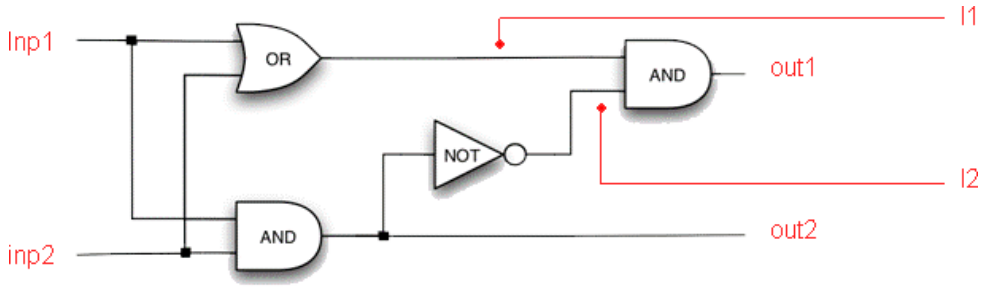

図 2.3: 加算器における input, output

回路構成の仕様

```
val OR_def = Define ‘OR(inp1, inp2, out) = (out = (inp1 \/ inp2))‘;
val AND_def = Define ‘AND(inp1, inp2, out) = (out = (inp1 /\inp2))‘;
val NOT_def = Define ‘NOT(inp, out) = (out = (~inp))‘;
val AM_IMP_def = Define
      ‘AM_IMP(inp1, inp2, out1, out2) =
     ?l1 l2.
      OR(inp1, inp2, l1) /\AND(inp1, inp2, out2) /\NOT(out2, l2)/\AND(l1, l2, out1)‘;
```

2. 命題 (goal) を設定する

   実装した加算器は仕様を充たしているか検証を行っていく．

   HOL における定理証明では，命題を設定し証明したいゴールを記述する．今回の加算器の場合においては全ての inp1，inp2，out1，out2 において AMP_IMP(inp1, inp2, out1, out2) ならば out1 = AM1(inp1, inp2)，out2 =AM2(inp1, inp2) の関係が成り立つことを証明する．そのうえで演繹的推論を進め証明を行っていく．

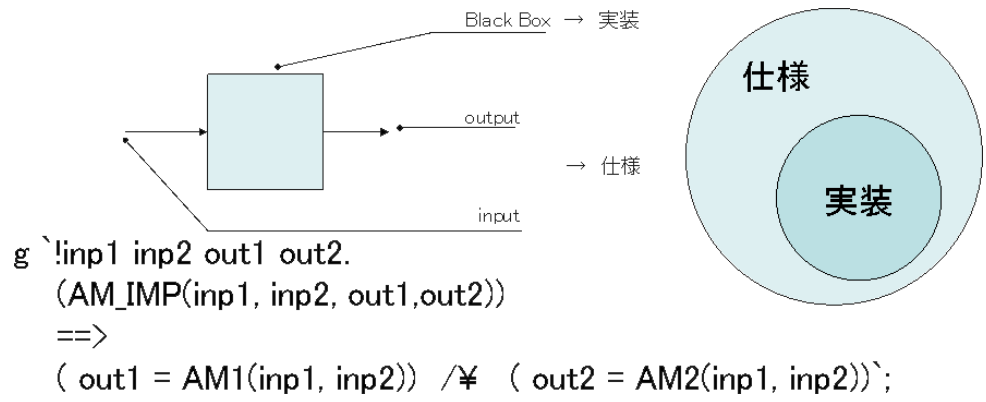

図 2.4: 加算器の命題

命題

```
g ‘!inp1 inp2 out1 out2.
   (AM_IMP(inp1, inp2, out1,out2))
  ==>
   ( out1 = AM1(inp1, inp2))    /\( out2 = AM2(inp1, inp2))‘;
```

3. 演繹的推論 (tactic) による証明を行う

- 命題を設定する

  g ‘!inp1 inp2 out1 out2.
  (AM_IMP(inp1, inp2, out1,out2))
  ==>
  (out1 = AM1(inp1, inp2))　/\(out2 =AM2(inp1, inp2))‘;

```
g `!inp1 inp2 out1 out2.
   (AM_IMP(inp1, inp2, out1,out2))
   ==>
   ( out1 = AM1(inp1, inp2))  /¥  ( out2 =AM2(inp1, inp2))`;
------------------------------------------------------------------------------------------------------------
- - > val it =
   Proof manager status: 1 proof.
   1. Incomplete:
       Initial goal:
       !inp1 inp2 out1 out2.
         AM_IMP (inp1,inp2,out1,out2) ==>
         (out1 = AM1 (inp1,inp2)) /¥ (out2 = AM2 (inp1,inp2))


    : proofs
```

図 2.5: 命題の設定

- 全称記号を取り除く

  e (REPEAT GEN_TAC);

```
e (REPEAT GEN_TAC);
------------------------------------------------------------------------------------------------------------
1 subgoal:
> val it =
   AM_IMP (inp1,inp2,out1,out2) ==>
   (out1 = AM1 (inp1,inp2)) /¥ (out2 = AM2 (inp1,inp2))

    : goalstack
```

←

```
- - > val it =
   Proof manager status: 1 proof.
   1. Incomplete:
       Initial goal:
       !inp1 inp2 out1 out2.
         AM_IMP (inp1,inp2,out1,out2) ==>
         (out1 = AM1 (inp1,inp2)) /¥ (out2 = AM2 (inp1,inp2))


    : proofs
```

図 2.6: タクティック 1

- 定理で書き換える

  e (REWRITE_TAC[AM_IMP_def, AND_def, NOT_def, OR_def, AM1_def, AM2_def]);

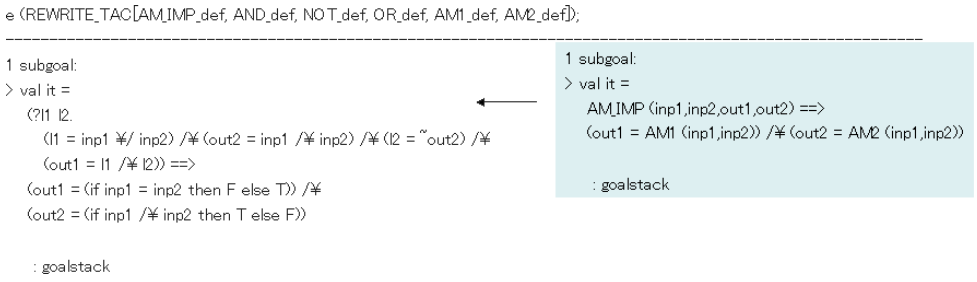

図 2.7: タクティック 2

- 前提条件を仮定として書き換える

  e (REPEAT STRIP_TAC);

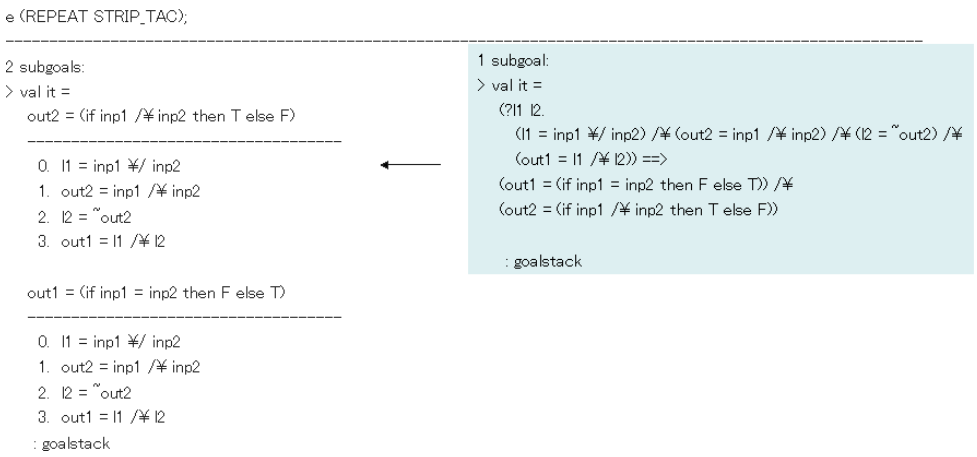

図 2.8: タクティック 3

- bool 値を用いて評価する

  e (RW_TAC bool_ss []);

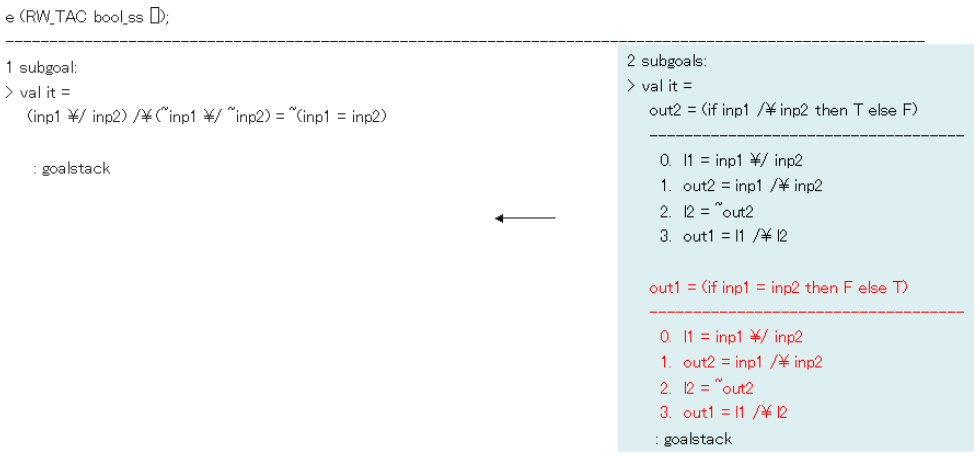

図 2.9: タクティック４

- イコールを双方向のならばに書き換える

  e EQ_TAC;

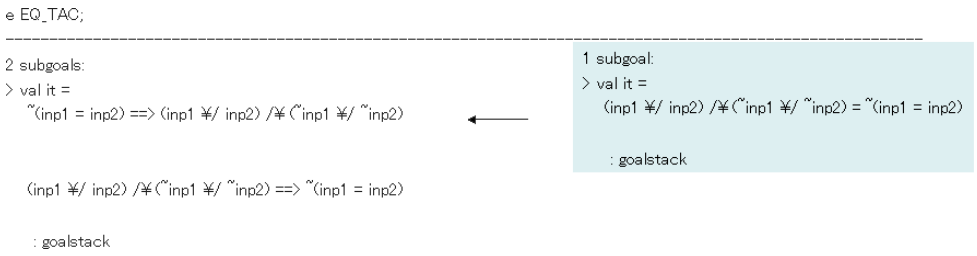

図 2.10: タクティック５

- bool 値を用いて評価する
  e (RW_TAC bool_ss []);

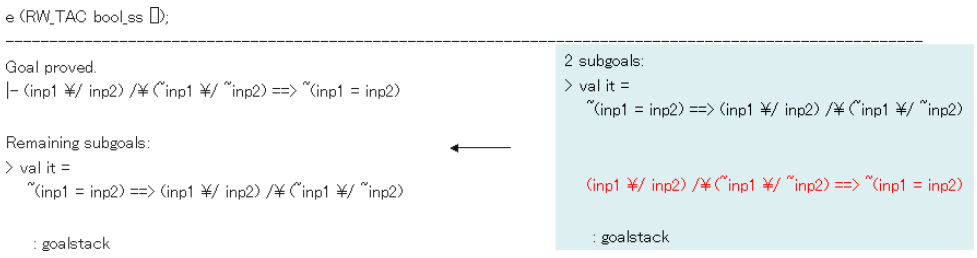

図 2.11: タクティック 6

- 公理，前提条件（一階述語論理）を用いゴールを証明
  e (PROVE_TAC []);

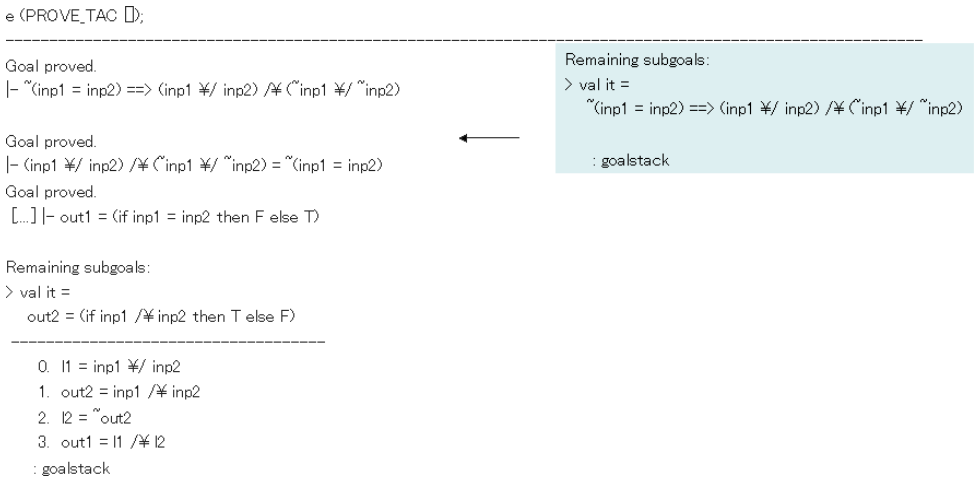

図 2.12: タクティック 7

- e (PROVE_TAC []);

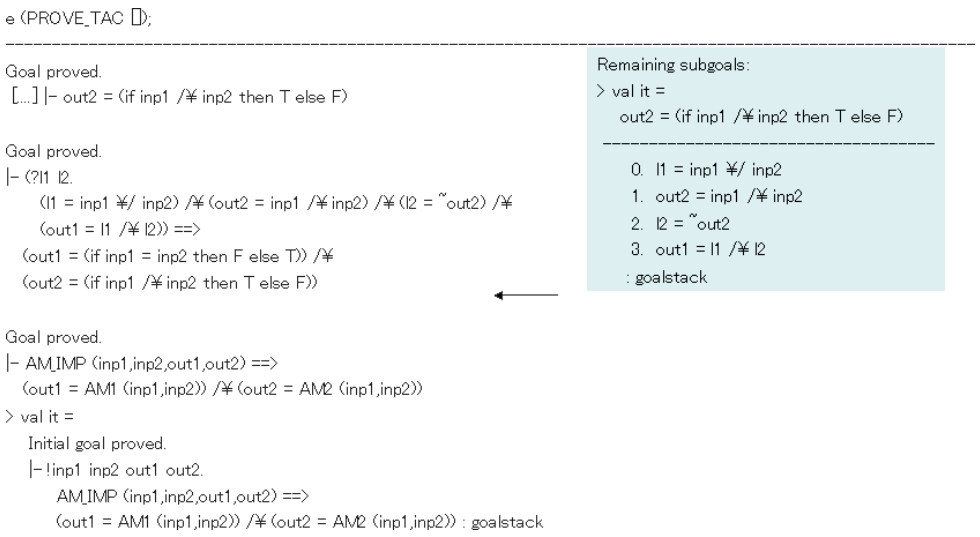

図 2.13: タクティック 8

以上の証明により，加算器の命題を証明することができた.

第3章　FWサーバシステムの仕様とモデル化

FWサーバシステムには４つの機能がある．

1. ログインマネージャ
   ユーザ情報の管理・ログインに関する機能を実行

2. アクセス・コントロールマネージャ
   認証を受けた管理者が使用できる機能，その他の制限の仕方を決定

3. パケット・フィルタマネージャ
   パケット・プロセッシング・ルールに従い，通過するパケットをパスするかドロップするか決定

4. オーディットマネージャ
   監査のための様々なイベント記録を生成
   監査するイベントが発生した場合，監査記録を生成
   セキュアな操作を監査するためのに必要な情報を収集

本研究ではオーディットマネージャについての検証を行った．

3.1　オブジェクト指向によるモデリング

定理証明器HOLはML上に構築されており，検証はMLに酷似したHOLで行う．しかしながら，ML形式での設計記述はコードに近く，誰にでも明確に理解が容易な表現であるとは言い難い．そこで，本研究ではモデル化に際し，機能，特性及びふるまいを明確に解りやすく記述することが可能なオブジェクト指向によるモデリングを行っていく．オブジェクト指向では，オブジェクト群がどのような構造を持つか，オブジェクト間ではどのようなメッセージ通信が行われるかをもとにモデリングを行っていく．オブジェクト指向において最も一般的なUML（Unified Modeling Language）において，前者はクラスモデルとして，後者はシーケンスモデルとして定義することが可能である．また，ObjectLojic（後述）を使用する際にもオブジェクト指向に沿ったシンタックスが必要となる．

3.2　オーディットマネージャの仕様

オーディットマネージャとはFWサーバシステムにおいて監査するイベントが発生した場合，監査記録の生成する機能である．今回，検証を行ったオーディットマネージャの仕様は，メインのクラスとなるamクラスにおいて監査記録を生成するメソッドaddAudrecの定義はイベント発生時，イベントが監査記録を行うリストに含まれているかどうか判断する．リストにはどういったイベントを監査するかが記録されている．ここで発生したイベントが監査リストに含まれない場合は何も行わない．もし監査リストに含まれる場合は，次に記録を行うファイルに空きがあるかどうかを確認する．もし空きがある場合はファイルの先頭に監査記録を生成する．空きがない場合は最も古い監査記録のファイルを削除し，新しいファイルを作成し監査記録を生成する．これがオーディットマネージャの主な仕様となる．

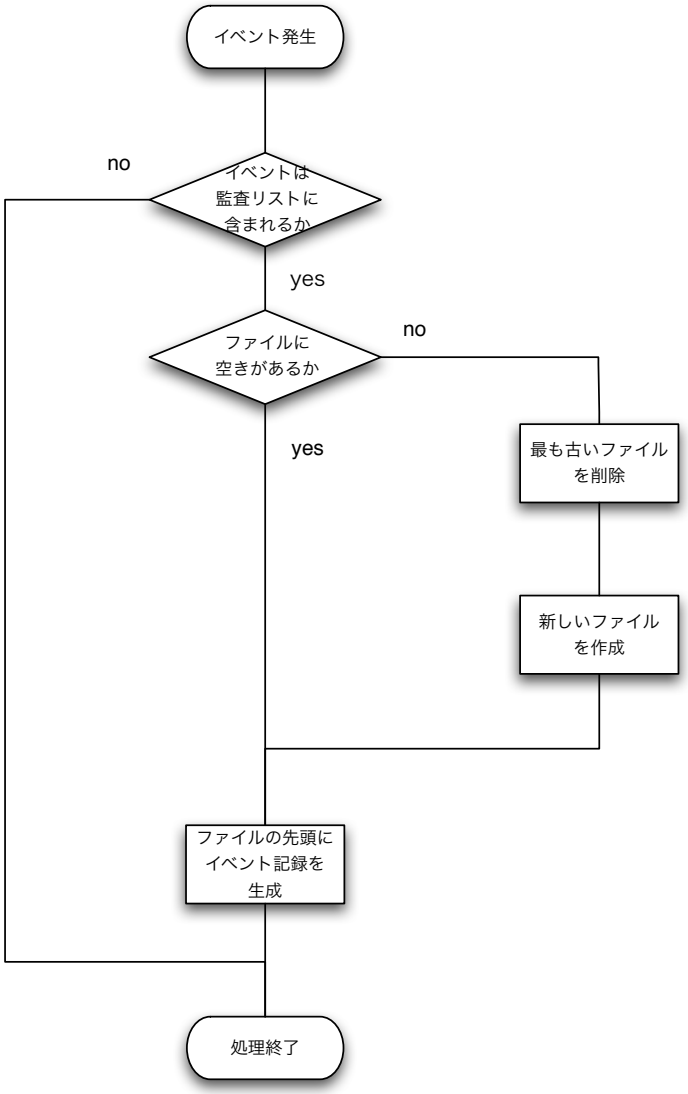

図 3.1: オーディットマネージャの仕様

なお，記録を生成する記録領域については以下のような階層構成になっており，Storage オブジェクトに FileList が格納されておりその下に AudrecList が格納されている．今回は FileList の最大値を３，AudrecList の最大値を５として定義した．

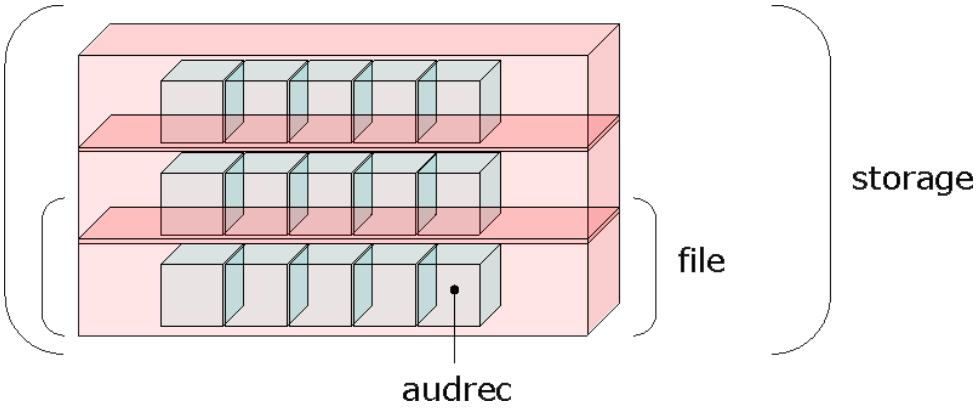

図 3.2: オーディットマネージャのファイルシステム

3.3 オーディットマネージャのモデル化

3.3.1 クラスモデル

静的性質（クラスの属性，メソッド）に基づいて，クラスダイヤグラムを作成する．

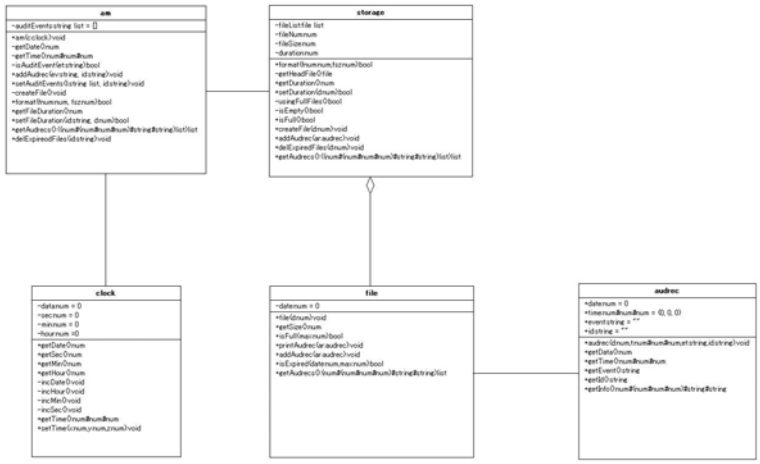

図 3.3: オーディットマネージャのにおけるクラスダイヤグラム

オーディットマネージャのモデル化では，オブジェクト指向理論により各機能ごとにam, storage，file，audrec，clock のクラスを設け仕様に基づいたモデル化を行っていく．各クラスについては以下の通りである．

- am　監査記録の管理
  - 監査リストを設定
  - イベントが監査リストに含まれるか判断
  - 監査を行う
- storage 監査記録を作成するストレージの操作
  - storage, file のサイズを決定
  - file を作る
  - file が空か判断
  - File が一杯か判断

- fie 監査記録を作成するファイルの操作
- audrec 監査記録を作成
  - file に監査記録を作成
- clock 時間を扱う
  - 監査に関連する時間と，その操作を扱う

3.3.2 シーケンスモデル

動的性質（オブジェクト間の動作）に基づき，シーケンスダイヤグラムを作成する．シーケンス図を用いることにより，システムの振る舞い証明したい命題について把握することが容易となる．今回はイベントを記録するメソッドからのシーケンス動作をモデル化を行った．

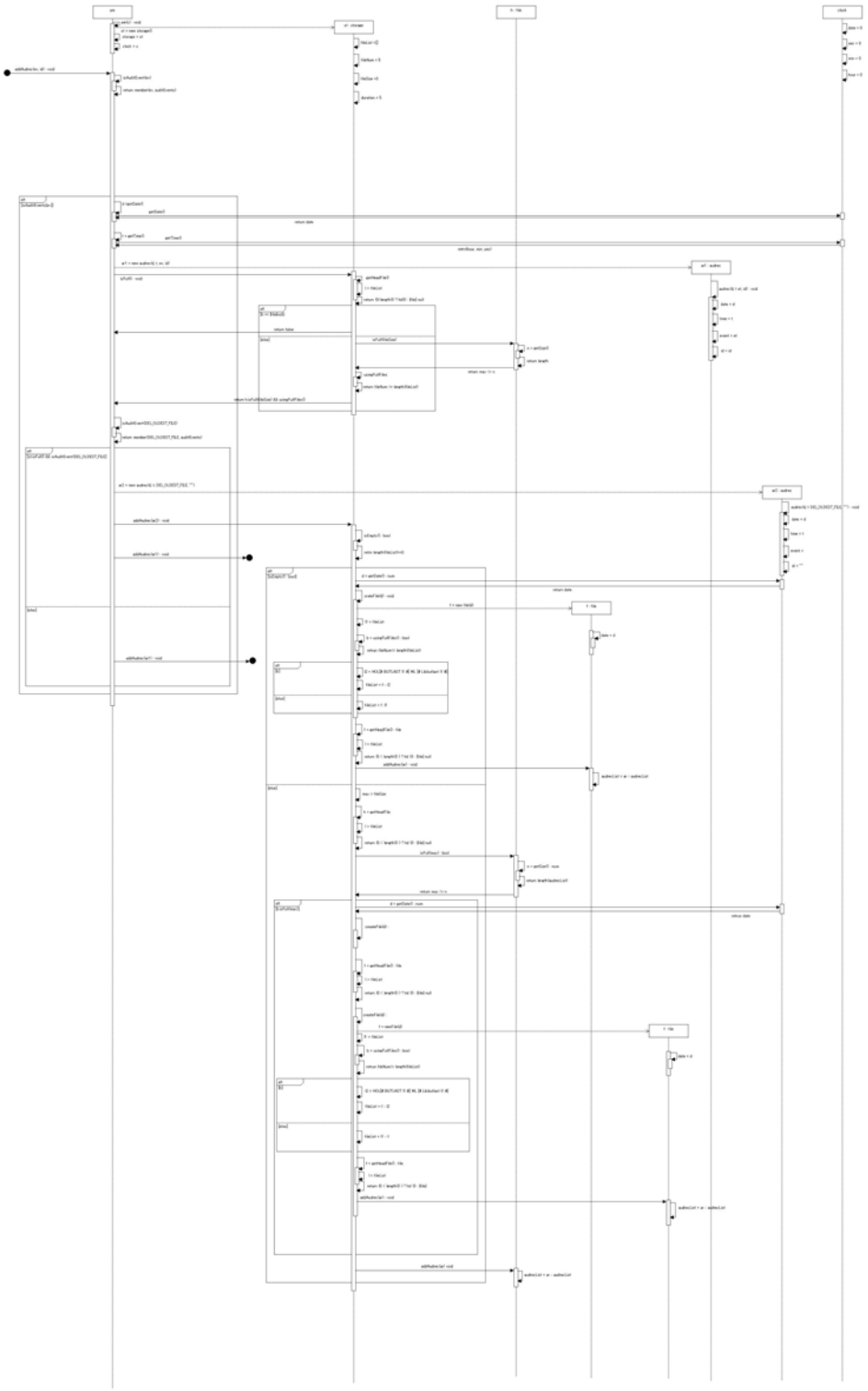

図 3.4: オーディットマネージャのにおけるシーケンスダイヤグラム

第4章　FWサーバシステムの検証

4.1　ObjectLogic

矢竹らはシーケンス動作における状態の変化について，演繹的な証明をおこなうシステムの土台としてオブジェクト指向理論を HOL 上に定義した．これにより HOL 上においてシステムの状態はストアと呼ばれるデータ型で表現されることとなる．その上にオブジェクトを操作するためにいくつかの基本演算子が定義される．

- ストアとオブジェクトの型は
  - 型 store の定義され定数として store_emp をもつ 0
    （直感的にはシステムに存在する全オブジェクトのデータを保持する環境）
  - クラス c ∈ C 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
    (C:システムに出現するクラス名の集合)
- ストア上に定義される６つの基本演算子は
  - 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:オブジェクトの型変換を行う述語
  - 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

ObjectLogic とは論理生成器であり，矢竹らが HOL のメタ言語である ML 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，そのシステムに特化した理論を HOL 論理モジュールとして構築する．論理モジュールとは，HOL における個々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．シーケンス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．さらに，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．

この理論によって与えられる基本演算子と HOL が提供する関数群を用いて，シーケンス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．

4.2 OML による環境構築

検証ではオブジェクト指向に基づくクラスモデルと，各クラスのもつメソッドを OML を用いて記述したものを ObjectLogic にへの入力し，ML 形式のシミュレーション実行環境，HOL 形式の検証環境を構築する．

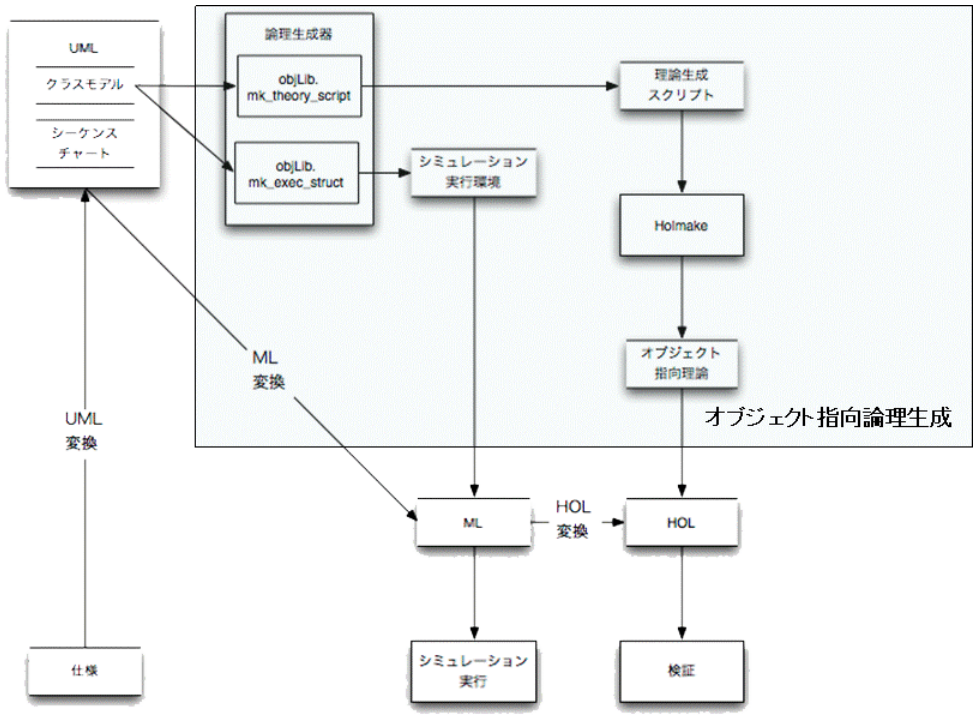

図 4.1: ML,HOL 環境の構築

4.3 シミュレーション実行

4.3.1 ML 形式での記述

ObjectLogic により ML 形式でのコードが提供される．HOL では具体的に値をいれ評価を行い動作を確かめることができない．そこで HOL での検証を行う前に ML でのシミュレーション実行を行い，期待通りの動作が得られるか，コードに誤りが無いかを検討する．

4.3.2 シミュレーション実行

オーディットマネージャの検証を行うにあたり，着目するべきはイベント発生時に監査リストに含まれるイベントの場合のみ監査記録を生成，そのイベントが記録されているか否かである．よってイベントが発生した場合を考えシミュレーション実行を行う．以下にその例を記述する．

コード上ではこの動作はイベントが auditEvents に含まれている場合 → file に記録を行う

- new_clock を空のストア（store_emp）に適用し, 結果の値を cl, s に格納
  クロックオブジェクトを作成

  val (cl,s) = new_clock store_emp;
  - - >val cl = <clock>: clock
  val s = <store>: store

  cl:生成された clock オブジェクト
  s :生成後のストア

- new_am cl をストア (s) に適用し, 結果の値を am, s に格納
  オーディットマネージャを作成

  val (am,s) = new_am cl s;
  - - >val am = <am>: am
  val s = <store>: store

  am:生成された am オブジェクト
  s :生成後のストア

- am_format am (3,5) をストア (s) に適用し, b は 3 および 5>=0 であれば true, 状態を s に格納
  fileList の大きさを３に，audrecList の大きさを５に設定する

  val (b,s) = am_format am (3,5) s;
  - - >val b = true : bool
  val s = <store>: store

  b:format の成功を bool 値
  s:生成後のストア

- am_setAuditEvents am (["E1","E2","E3"],""),"") をストア (s) に適用し, 結果の値を s に格納
  監査を行う記録を E1,E2,E3 のリストとして auditEvens に記録

  ```
  val s = am_setAuditEvents am (["E1","E2","E3"],"") s;
  - - >val s = <store>: store
  ```

  s:生成後のストア

- am_addAudrec am ("E1","A") をストア (s) に適用し, 結果の値を s に格納
  E1 のイベントを監査記録に追加する
  E1 は監査リストに存在するので監査記録が生成される

  ```
  val s = am_addAudrec am ("E1","A") s;
  - - >val s = <store>: store
  ```

  s:生成後のストア

- 以下同じようにイベントを監査記録に追加する
  しかしながら E4,E5 については, 監査リストに入っていないので監査記録は生成されない

  ```
  val s = am_addAudrec am ("E2","A") s;
  val s = am_addAudrec am ("E3","A") s;
  val s = am_addAudrec am ("E4","C") s;
  val s = am_addAudrec am ("E5","C") s;
  val s = am_addAudrec am ("E1","A") s;
  val s = am_addAudrec am ("E1","B") s;
  val s = am_addAudrec am ("E1","C") s;
  ```

- 監査リストを確認する

  ```
  am_get_auditEvents am s;
  - - >val it = ["E1", "E2", "E3"] : string
  ```

- 監査記録を確認する

  ```
  am_getAudrecs am s;
  - - >val it =
        [[(0, ((0, (0, 0)), ("E1", "C")))],
        [(0, ((0, (0, 0)), ("E1", "B"))), (0, ((0, (0, 0)), ("E1", "A"))),
  ```

```
    (0, ((0, (0, 0)), ("E3", "A"))), (0, ((0, (0, 0)), ("E2", "A"))),
    (0, ((0, (0, 0)), ("E1", "A")))]] :
(int * ((int * (int * int)) * (string * string))) list list
```

以上のシミュレーション実行を行った結果，たしかに監査リストにあるイベントのみが監査記録として記録されていることが確認できた．また，追加された監査記録は以下のように audrec に収納されている．

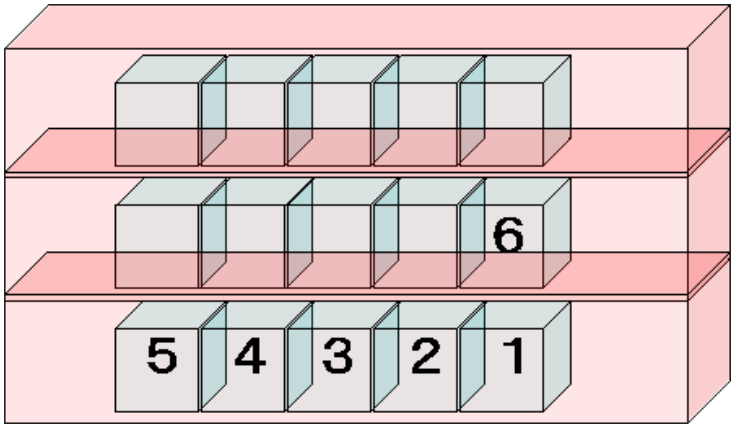

図 4.2: 作成された監査記録

```
val s = am_addAudrec am (“ E1 ”,“ A ”) s;    →  1
val s = am_addAudrec am (“ E2 ”,“ A ”) s;    →  2
val s = am_addAudrec am (“ E3 ”,“ A ”) s;    →  3
val s = am_addAudrec am (“ E1 ”,“ A ”) s;    →  4
val s = am_addAudrec am (“ E1 ”,“ B ”) s;    →  5
val s = am_addAudrec am (“ E1 ”,“ C ”) s;    →  6
```

このほかに，同様にファイルが Full であるときの動作についてもシミュレーション実行を行った結果，最も古いファイルが削除され，新しいファイルが作成され，そのファイルの先頭に監査記録が生成された．これよりプログラムは正常に動作していると確認できた．

4.4 検証環境

4.4.1 HOL 形式での記述

ObjectLogic により HOL での検証環境が構築される．

4.5 オーディッとマネージャの検証

4.5.1 命題の設定

ここでは，オーディットマネージャについて具体的な命題を設定する．具体的には，オーディットマネージャオブジェクトがストアに存在 (am_ex am s) し，かつストレージオブジェクトが存在 (storage_ex(am_get_storage am s) s) し，かつファイルサイズが 0 以上 (storage_getFileSize (am_get_storage am s) s $>$ 0) ならば，イベント ev を書き込み更新したストアを s'let s' = (am_addAudrec am (ev,id) s) in としてイベント ev が監査リストに含まれている (am_isAuditEvent am ev s) ならば，ストア s' のストレージの最初のファイルの先頭に記録されているイベントは書き込んだイベント ev である．すなわち，監査リストに含まれるイベントが発生した場合，ファイルの先頭にそのイベントの監査記録が生成されるというオーディットマネージャの主となる機能を証明する．

命題を設定するにあたり，まずシステムのいかなる状態においても変わらない値storage_getHeadFile_ex_I
として，不変表明を定義する

```
val storage_getHeadFile_ex_Inv = Define
    ‘storage_getHeadFile_ex_Inv am s =
    let st = am_get_storage am s in
~storage_isEmpty st s ==>
    file_ex (storage_getHeadFile st s) s‘;;
```

am オブジェクトより storage オブジェクトを get し st とし，st オブジェクトが空でないならば st オブジェクトの先頭の file は存在する
すなわち st が空でないならば file は存在するということについて不変表明として定義できる

次にゴールを設定する

```
g‘!am s ev id.
storage_getHeadFile_ex_Inv am s
/\
  (am_ex am s)
/\
  (storage_ex (am_get_storage am s) s)
/\
  (storage_getFileSize (am_get_storage am s) s > 0)
==>
```

```
  let s' = (am_addAudrec am (ev,id) s) in
  (am_isAuditEvent am ev s)
==>
  ((am_getHeadEvent am s') = ev)‘;;
```
コードの意味するところは前提条件として不変表明，am オブジェクトが存在する，ストレージオブジェクトが存在するファイルサイズが 0 以上であるならば，ev という監査記録を記録をおこなたストアを s' として，ev が監査記録リストに含まれるならば，ストア s' において，ファイルの先頭に記録されたイベントは ev でる．すなわち，監査記録リストに含まれるイベントが発生すると，ファイルの先頭に監査記録が生成されるということを証明したい．

4.5.2 証明

証明を行っていくに際し，その方法は大きく分けて 2 つある．

1. Forward Proof
   公理とすでに証明された定理に導出規則を適用して新たな定理を導く証明法

2. Goal Oriented Proof
   証明対象の命題を goal に設定し，tactic と呼ばれる証明戦略により，より簡単なゴールへと分解していく証明方法

一般的に通常は Goal Oriented Proof をとり，goal に tactic を適用しゴールを変化させて証明を行うが，証明途中に ForwardProof を用いることにより証明の効率をあげれる場合がある．証明の効率についても，後述の証明のなかで触れる．

オーディットマネージャの全体の証明は付録に記載する．ここでは演繹的推論による定理証明について，作業をトレースしながら紹介していく

```
e(REPEAT GEN_TAC);;
```
全称記号を取り除く

```
e(LET_TAC);;
```
cyberLET 項を取り除く

```
e(REPEAT STRIP_TAC);;
```
前提を仮定に書き換える STRIP_TAC を複数回 REPEAT しながら行う

```
e(RL_TAC [am_addAudrec_def]);;
```

am_addAudrec の定理について展開し goal を書き換える

```
e(RL_TAC [FST_new_audrec_FST_audrec_new]);;
```
補題 FST_new_audrec_FST_audrec_new の定理について展開し goal を書き換える

```
e(ASM_SIMP_TAC bool_ss []);;
```
仮定の定理を用い goal を書き換える

```
e (Cases_on ‘
     storage_isEmpty (am_get_storage am s) s \/
     file_isFull (storage_getHeadFile (am_get_storage am s) s)
     (storage_get_fileSize (am_get_storage am s) s) s‘);;
storage_isEmpty (am_get_storage am s) s \/
     file_isFull (storage_getHeadFile (am_get_storage am s) s)
     (storage_get_fileSize (am_get_storage am s) s) s
```
を仮定とし true のとき
と false の場合に goal を分岐する

```
e (ASSUME_TAC (REWRITE_RULE [storage_ex_new_audrec, storage_isEmpty_new_audrec,
file_isFull_new_audrec, storage_getHeadFile_newaudrec,
storage_get_fileSize_new_audrec, file_new_new_audrec,
new_audrec_audrec_new]
(SPECL
[‘‘am_get_storage am s‘‘,
 ‘‘SND (new_audrec (am_getDate am s,am_getTime am s,ev,id) s)‘‘,
 ‘‘FST (new_audrec (am_getDate am s,am_getTime am s,ev,id) s)‘‘]
(LET_RULE storage_getHeadFile_storage_addAudrec_1))));;
```
LET_RULE により storage_getHeadFile_storage_addAudrec_1 の LET 項をとりのぞいた
ものに SPECL によって全称記号に先頭から‘‘am_get_storage am s‘‘, ‘‘SND (new_audrec (am_getDate
順番で当て割り当てる. またこの定理を REWRITE_TAC によって storage_ex_new_audrec,storage_isEmpty_
によって展開したものを ASSUME_TAC により仮定へ追加する, この場合はゴールに合わせ
定理を用い定理を記述し直す, FowardProof となる.

```
e (ASM_UNDISCH_TAC 6);;
```
仮定の 6 番を前提条件として書き出す

```
e(SIMP_TAC list_ss []);;
```
リストの規則を使い goal を書き換える

```
e (RW_TAC bool_ss []);;
```
if の項を subgoal に分解する

```
e(SLICE_TAC);;
```
関係のないメソッドに対してスライシングを行う

```
e(OBJ_SIMP_TAC);;
```
オブジェクト理論を用いたシンプリファイを行い goal を書き換える

```
val FST_new_audrec_FST_audrec_new = prove
(``!d t ev id s.
 FST(new_audrec (d, t, ev, id) s) =
 FST(audrec_new s)``,
);;
```
tactic を用い tactics（補題）を定義することができる
=の左辺は右辺により定義することができ，定義を何度も使用する場合や状態数の多い展開となるときに有効である

4.5.3 結論

オーディットマネージャは命題に設定した監査記録リストに含まれるイベントが発生すると，ファイルの先頭に監査記録が生成されるとおい内容について仕様を充たしているということが証明できた.

第5章　考察

今回用いた演繹手法による検証では，その手順として，命題としてgoalを設定し，それらをRW_TACなどのTacticを用いてsubgoalに分解し，さらにメソッドの定義よりAtomicな基本演算子にまで展開し，SLICE_TACやSIMP_TACを用いて仮定や定理などから証明するという過程をとった．この証明における表現として論理式を用いる，定理証明では検証対象の仕様を論理式として定義するが論理式は表現の幅が広く，多くの状態遷移を持つモデルについても記述することができる，状態遷移を再起性をモデルでは無限の状態遷移を有するモデルも表現可能である．このことはオーディットマネージャの証明作業を通しても見ることができた，メソッドを定義により展開することにより多くのsubgoalに分岐する場合があり，こういった場合には，Case_onにより前提条件を強め証明をおこなう範囲を狭めることや，補題を立て，f(g(s)) → f(s)のように直接影響しないメソッドをスライスすることや，FowardProofによりtactic → tacticへの変換を行い，定理の方からgoalの形へ近づけていくことにより証明の効率を上げることができた．また命題を証明することができたが，命題の設定の仕方によっては仕様の再現に漏れがあることも考えられる．今回の場合においては，監査リストに含まれるイベントが発生した場合，ファイルの先頭にある記録は，その発生したイベントであると考え証明を行ったが，イベント発生以前に同じイベントが格納されている場合も考えられる．こういった場合については前提条件の指定を狭め，記録が無いとき，また記録があるときの２通りを考え帰納法を用いた証明を行わなければならないように思える．さらに証明においては記録を行うfileSizeが0以上であるという前提条件を使用しなかったため，この条件を除き，証明を行ったところ証明が完了した．シミュレーション実行においてfileSizeを０に設定し，この動作を確認したところ，数に制限なく監査記録を生成することができたため，この部分は実装コードのバグであると考えられる．証明をおこなった監査リストに含まれる記録は作成されていることは証明でき，この機能については問題がないが，fileSizeは監査記録を行う動作に付随する条件であり，証明を行った監査を行うという動作も阻害される可能性もあり考えなくてはならない結果となった．

第6章　おわりに

本研究の目的では検証の効率化を検討する予定であったが，検証に時間を費やしてしまい．結果として検証し証明するまでの研究結果となった．今回の証明においてもシミュレーションでは単純にみえる動作でも検証を行ってみるとかなりの時間的，人的なコストが必要となることがわかった．また効率化を行う際にも goal を分割すれば良いのか，補題を立てれば良いのかどういった方法が少ないコストで証明を完了するか，今後は検証の効率化が必要になってくるのではないだろうかと考える．また，仕様をモデル化し論理式により記述する場合に性質を保っているかという曖昧性に付いても今後の課題のようである．

第7章　謝辞

本研究を行うにあたり，始終御指導賜った片山卓也教授に深く感謝致します．またゼミなどを通し岸知二教授，青木利晃准教授，DEFAGO Xavier 准教授には貴重なご意見をいただきました．また片山研究室の矢竹健朗助教には検証技術，プログラムに一から教えていただき，本研究において様々なアドバイスして頂きました．片山研究室，岸研究室，青木研究室，DEFAGO 研究室の先輩，同輩の皆様には様々な相談にのっていただきました．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．

第8章　付録A

FWサーバシステムにおけるオーディットマネージャにおけるHOLでの検証コードを以下に示す.

8.1　A.1補題（タクティカル）

```
val FST_new_audrec_FST_audrec_new = prove
(‘‘!d t ev id s.
 FST(new_audrec (d, t, ev, id) s) =
 FST(audrec_new s)‘‘,
  (RL_TAC [new_audrec_def])
);;

val am_get_storage_storage_addAudrec = prove
(‘‘!am st ar s.
 am_get_storage am (storage_addAudrec st ar s) =
 am_get_storage am s‘‘,
 EVERY[
  (RL_TAC [storage_addAudrec_def]),
  (RW_TAC bool_ss [])
  THENL[
   EVERY[
    (RL_TAC [file_addAudrec_def]),
    (RL_TAC [storage_getHeadFile_def]),
    (SLICE_TAC),
    (RL_TAC [storage_createFile_def]),
    (RL_TAC [storage_usingFullFiles_def]),
    (RL_TAC [new_file_def]),
    (RL_TAC [file_file_def]),
    (RL_TAC [audrec_getDate_def]),
    (RL_TAC [storage_isEmpty_def]),
```

```
 (RL_TAC [file_addAudrec_def]),
 (RW_TAC bool_ss [])
]
THENL[
 EVERY[
  (SLICE_TAC)
 ],
 EVERY[
  (SLICE_TAC)
 ]
]
,
EVERY[
 (RL_TAC [file_addAudrec_def]),
 (RL_TAC [storage_getHeadFile_def]),
 (RL_TAC [storage_createFile_def]),
 (SLICE_TAC),
 (RL_TAC [storage_usingFullFiles_def]),
 (RW_TAC bool_ss [])
]
THENL[
 EVERY[
  (RL_TAC [new_file_def]),
  (RL_TAC [file_file_def]),
  (SLICE_TAC)
 ],
 EVERY[
  (RL_TAC [new_file_def]),
  (RL_TAC [file_file_def]),
  (SLICE_TAC)
 ]
]
,
EVERY[
 (RL_TAC [file_addAudrec_def]),
 (RL_TAC [storage_getHeadFile_def]),
 (SLICE_TAC)
]
```

```
  ]
 ]
);;

val am_get_storage_new_audrec = prove
(``!am d t ev id s.
 am_get_storage am (SND(new_audrec (d,t,ev,id) s)) =
 am_get_storage am s``,
 EVERY[
  (RL_TAC [new_audrec_def]),
  (RL_TAC [audrec_audrec_def]),
  (SLICE_TAC)
 ]
);;

val storage_ex_new_audrec = prove
(``!st d t ev id s.
storage_ex st (SND (new_audrec (d,t,ev,id) s)) = storage_ex st s``,
EVERY[
(RL_TAC [new_audrec_def]),
(RL_TAC [audrec_audrec_def]),
(SLICE_TAC)
]
);;

val storage_isEmpty_new_audrec = prove
(``!st d t ev id s.
storage_isEmpty st (SND (new_audrec (d,t,ev,id) s)) =
storage_isEmpty st s``,
EVERY[
(RL_TAC [storage_isEmpty_def]),
(RL_TAC [new_audrec_def]),
(RL_TAC [audrec_audrec_def]),
(SLICE_TAC)]
);;

val file_isFull_new_audrec = prove
(``!f x d t ev id s.
```

```
file_isFull f x (SND (new_audrec (d,t,ev,id) s)) =
file_isFull f x s``,
EVERY[
(RL_TAC [file_isFull_def]),
(RL_TAC [file_getSize_def]),
(RL_TAC [new_audrec_def]),
(RL_TAC [audrec_audrec_def]),
(SLICE_TAC)]
);;

val storage_getHeadFile_newaudrec = prove
(``!st d t ev id s.
storage_getHeadFile st (SND (new_audrec (d,t,ev,id) s)) =
storage_getHeadFile st s``,
EVERY[
(RL_TAC [storage_getHeadFile_def]),
(RL_TAC [new_audrec_def]),
(RL_TAC [audrec_audrec_def]),
(SLICE_TAC)]
);;

val storage_get_fileSize_new_audrec = prove
(``!st d t ev id s.
storage_get_fileSize st (SND (new_audrec (d,t,ev,id) s)) =
storage_get_fileSize st s``,
EVERY[
(RL_TAC [new_audrec_def]),
(RL_TAC [audrec_audrec_def]),
(SLICE_TAC)]
);;

val file_new_new_audrec = prove
(``!d t ev id s.
FST (file_new (SND (new_audrec (d,t,ev,id) s))) = FST (file_new s)``,
EVERY[
(RL_TAC [new_audrec_def]),
(RL_TAC [audrec_audrec_def]),
(SLICE_TAC)]
```

```
);;

val new_audrec_audrec_new = prove
(``!d t ev id s.
FST (new_audrec (d, t, ev, id) s) = FST (audrec_new s)``,
(RL_TAC [new_audrec_def, audrec_audrec_def])
);;

val storage_getHeadFile_storage_createFile = prove
(``!st d s.
 storage_ex st s
  ==>
 (storage_getHeadFile st (storage_createFile st d s) =
 FST(file_new s))
``,
 EVERY[
  (RL_TAC [storage_getHeadFile_def]),
  (RL_TAC [storage_createFile_def]),
  (REPEAT STRIP_TAC),
  (Cases_on `storage_usingFullFiles st (SND (new_file d s))`)]
 THENL[
  EVERY[
    (ASM_SIMP_TAC bool_ss []),
    (RL_TAC [new_file_def]),
    (RL_TAC [file_file_def]),
    (SLICE_TAC),
    (OBJ_SIMP_TAC),
    (RW_TAC list_ss [])
  ],
  EVERY[
    (ASM_SIMP_TAC bool_ss []),
    (RL_TAC [new_file_def]),
    (RL_TAC [file_file_def]),
    (SLICE_TAC),
    (OBJ_SIMP_TAC),
    (RW_TAC list_ss [])
  ]
 ]
```

```
);;

val case1_tac =
EVERY
[RL_TAC [storage_addAudrec_def],
 ASM_SIMP_TAC bool_ss [],
 ASM_SIMP_TAC bool_ss [storage_getHeadFile_storage_createFile],
 RL_TAC [storage_getHeadFile_def],
 RL_TAC [file_addAudrec_def],
 SLICE_TAC,
 RL_TAC [storage_createFile_def],
 RW_TAC bool_ss []] THENL
 [EVERY
  [RL_TAC [new_file_def,file_file_def],
   SLICE_TAC, OBJ_SIMP_TAC,
   RW_TAC list_ss []],
  EVERY
  [RL_TAC [new_file_def,file_file_def],
   SLICE_TAC, OBJ_SIMP_TAC,
   RW_TAC list_ss []],
  EVERY
  [ASM_UNDISCH_TAC 2,
   Cases_on `storage_usingFullFiles st
   (SND (new_file (audrec_getDate ar s) s))` THENL
    [EVERY
     [ASM_SIMP_TAC bool_ss [],
      RL_TAC [new_file_def,file_file_def],
      SLICE_TAC, OBJ_SIMP_TAC,
      RW_TAC list_ss []],
     EVERY
     [ASM_SIMP_TAC bool_ss [],
      RL_TAC [new_file_def,file_file_def],
      SLICE_TAC, OBJ_SIMP_TAC,
      RW_TAC list_ss []]]];;

val case2_tac =
EVERY
[RL_TAC [storage_addAudrec_def],
```

```
 ASM_SIMP_TAC bool_ss [],
 ASM_SIMP_TAC bool_ss [storage_getHeadFile_storage_createFile],
 RL_TAC [storage_getHeadFile_def],
 RL_TAC [file_addAudrec_def],
 SLICE_TAC,
 RL_TAC [storage_createFile_def],
 RW_TAC bool_ss []] THENL
[EVERY
 [RL_TAC [new_file_def,file_file_def],
  SLICE_TAC, OBJ_SIMP_TAC,
  RW_TAC list_ss []],
 EVERY
 [RL_TAC [new_file_def,file_file_def],
  SLICE_TAC, OBJ_SIMP_TAC,
  RW_TAC list_ss []],
 ASM_UNDISCH_TAC 2 THEN
 Cases_on `storage_usingFullFiles st
 (SND (new_file (audrec_getDate ar s) s))` THENL
  [EVERY
   [ASM_SIMP_TAC bool_ss [],
    RL_TAC [new_file_def,file_file_def],
    SLICE_TAC, OBJ_SIMP_TAC,
    RW_TAC list_ss []],
   EVERY
   [ASM_SIMP_TAC bool_ss [],
    RL_TAC [new_file_def,file_file_def],
    SLICE_TAC, OBJ_SIMP_TAC,
    RW_TAC list_ss []]]];;
val storage_getHeadFile_storage_addAudrec_1 = prove
    (``!st s ar.
     storage_ex st s ==>
     let h = storage_getHeadFile st s in
     let max = storage_get_fileSize st s in
 storage_isEmpty st s \/ file_isFull h max s ==>
  (storage_getHeadFile st (storage_addAudrec st ar s) = FST(file_new s))``,
     RW_TAC list_ss [] THENL [case1_tac,case2_tac]);;
val storage_getHeadFile_storage_addAudrec_2 = prove
( ``!st s ar.
```

```
     storage_ex st s ==>
     let h = storage_getHeadFile st s in
     let max = storage_get_fileSize st s in
 ~storage_isEmpty st s /\  ~file_isFull h max s ==>
  (storage_getHeadFile st (storage_addAudrec st ar s) =
   storage_getHeadFile st s)‘‘,
EVERY[
(RL_TAC [storage_addAudrec_def]),
(RW_TAC bool_ss []),
(RL_TAC [storage_getHeadFile_def]),
(RL_TAC [file_addAudrec_def]),
(SLICE_TAC)]
);;
val file_get_audrecList_storage_createFile_new_audrec = prove
    (‘‘!st d t ev id s.
     file_get_audrecList (FST (file_new s))
     (storage_createFile st d (SND (new_audrec (d,t,ev,id) s))) =
     []‘‘,
     EVERY
     [RL_TAC [storage_createFile_def],
      RW_TAC bool_ss [],
      RL_TAC [new_file_def,file_file_def],
      RL_TAC [new_audrec_def,audrec_audrec_def],
      SLICE_TAC,
      OBJ_SIMP_TAC]);;

val audrec_get_event_file_addAudrec = prove
(‘‘!ar f ar2 s.
audrec_get_event ar (file_addAudrec f ar2 s) =
audrec_get_event ar s‘‘,
EVERY[
(RL_TAC [file_addAudrec_def]),
(SLICE_TAC)
]
);;

val audrec_get_event_storage_createFile = prove
(‘‘!ar f d s.
```

```
audrec_get_event ar (storage_createFile f d s) =
audrec_get_event ar s‘‘,
EVERY[
(RL_TAC [storage_createFile_def]),
(RW_TAC bool_ss []),
(RL_TAC [new_file_def, file_file_def]),
(SLICE_TAC)
]
);;

val storage_getHeadFile_new_audrec = prove
(‘‘!st d t ev id s.
 storage_getHeadFile st (SND (new_audrec (d,t,ev,id) s)) =
 storage_getHeadFile st s‘‘,
 EVERY[
  (R_TAC [storage_getHeadFile_def]),
  (LET_TAC),
  (R_TAC [new_audrec_def]),
  (LET_PAIR_TAC),
  (R_TAC [audrec_audrec_def]),
  (LET_TAC),
  (SLICE_TAC)
 ]
);;

val audrec_getDate_new_audrec = prove
(‘‘!d t ev id s.
audrec_getDate (FST (audrec_new s)) (SND (new_audrec (d,t,ev,id) s)) = d‘‘,
EVERY[
(RL_TAC [audrec_getDate_def]),
(RL_TAC [new_audrec_def, audrec_audrec_def]),
(SLICE_TAC),
(OBJ_SIMP_TAC)]
);;

val file_get_audrecList_storage_addAudrec_new_audrec_1 = prove
    (‘‘!am ev id s.
 storage_ex (am_get_storage am s) s
```

```
==>
 storage_isEmpty (am_get_storage am s) s \/
 file_isFull (storage_getHeadFile (am_get_storage am s) s)
 (storage_get_fileSize (am_get_storage am s) s) s
==>
(file_get_audrecList (FST (file_new s))
 (storage_addAudrec (am_get_storage am s) (FST (audrec_new s))
  (SND (new_audrec (am_getDate am s,am_getTime am s,ev,id) s))) =
   [FST (audrec_new s)])``,
     EVERY
     [RL_TAC [storage_addAudrec_def],
      RL_TAC [storage_isEmpty_new_audrec],
      RL_TAC [storage_getHeadFile_new_audrec],
      RL_TAC [storage_get_fileSize_new_audrec],
      RL_TAC [file_isFull_new_audrec],
      RW_TAC bool_ss []] THEN
      EVERY
      [RL_TAC [audrec_getDate_new_audrec],
       ASM_SIMP_TAC bool_ss
       [REWRITE_RULE [storage_ex_new_audrec,file_new_new_audrec]
(SPECL [``am_get_storage am s``,``am_getDate am s``,
``SND (new_audrec
       (am_getDate am s,am_getTime am s,ev,id) s)``]
storage_getHeadFile_storage_createFile)],
RL_TAC [file_addAudrec_def],
RL_TAC [file_get_audrecList_storage_createFile_new_audrec],
RL_TAC [storage_createFile_def],
RW_TAC bool_ss []] THEN
EVERY
[SLICE_TAC,
 RL_TAC [new_file_def,file_file_def],
 RL_TAC [new_audrec_def,audrec_audrec_def],
 SLICE_TAC, OBJ_SIMP_TAC]);;

val storage_getHeadFile_ex_Inv = Define
    `storage_getHeadFile_ex_Inv am s =
    let st = am_get_storage am s in
~storage_isEmpty st s ==>
```

```
    file_ex (storage_getHeadFile st s) s‘;;

val file_get_audrecList_storage_addAudrec_new_audrec_2 = prove
(‘‘!am ev id s.
 storage_ex (am_get_storage am s) s /\
  storage_getHeadFile_ex_Inv am s
==>
 ~(storage_isEmpty (am_get_storage am s) s \/
 file_isFull (storage_getHeadFile (am_get_storage am s) s)
 (storage_get_fileSize (am_get_storage am s) s) s)
==>
(file_get_audrecList (storage_getHeadFile (am_get_storage am s) s)
 (storage_addAudrec (am_get_storage am s) (FST (audrec_new s))
  (SND (new_audrec (am_getDate am s,am_getTime am s,ev,id) s))) =
   (FST (audrec_new s))::
    (file_get_audrecList (storage_getHeadFile (am_get_storage am s) s) s))‘‘,
EVERY[
RL_TAC [storage_getHeadFile_ex_Inv],
RL_TAC [storage_addAudrec_def],
RL_TAC [storage_isEmpty_new_audrec],
RL_TAC [storage_getHeadFile_new_audrec],
RL_TAC [storage_get_fileSize_new_audrec],
RL_TAC [file_isFull_new_audrec],
SIMP_TAC bool_ss [],
REPEAT STRIP_TAC,
RES_TAC,
RL_TAC [new_audrec_def],
RL_TAC [audrec_audrec_def],
RL_TAC [file_addAudrec_def],
SLICE_TAC,
OBJ_SIMP_TAC]
);;
```

8.2 A.2 命題の証明

```
g‘!am s ev id.
```

```
storage_getHeadFile_ex_Inv am s
/\
  (am_ex am s)
/\
  (storage_ex (am_get_storage am s) s)
/\
  (storage_getFileSize (am_get_storage am s) s > 0)
==>
  let s' = (am_addAudrec am (ev,id) s) in
  (am_isAuditEvent am ev s)
==>
  ((am_getHeadEvent am s') = ev)‘;;

e(REPEAT STRIP_TAC);;

e(RL_TAC [am_addAudrec_def]);;

e(RL_TAC [FST_new_audrec_FST_audrec_new]);;

e(RL_TAC [am_getHeadEvent_def]);;

e(RL_TAC [storage_getHeadEvent_def]);;

e(RL_TAC [audrec_getEvent_def]);;

e(RL_TAC [file_getHeadAudrec_def]);;

e(ASM_SIMP_TAC bool_ss []);;

e(RL_TAC [am_get_storage_storage_addAudrec]);;

e(RL_TAC [am_get_storage_new_audrec]);;

e (Cases_on ‘
     storage_isEmpty (am_get_storage am s) s \/
     file_isFull (storage_getHeadFile (am_get_storage am s) s)
     (storage_get_fileSize (am_get_storage am s) s) s‘);;
```

```
c_get_a i (d_set_b j x s) = c_get_a i s

e (ASSUME_TAC (REWRITE_RULE [storage_ex_new_audrec, storage_isEmpty_new_audrec,
file_isFull_new_audrec, storage_getHeadFile_newaudrec,
storage_get_fileSize_new_audrec, file_new_new_audrec,
new_audrec_audrec_new]
(SPECL
[‘‘am_get_storage am s‘‘,
 ‘‘SND (new_audrec (am_getDate am s,am_getTime am s,ev,id) s)‘‘,
 ‘‘FST (new_audrec (am_getDate am s,am_getTime am s,ev,id) s)‘‘]
(LET_RULE storage_getHeadFile_storage_addAudrec_1))));;

e (ASM_UNDISCH_TAC 6);;

e (ASM_SIMP_TAC bool_ss []);;

e STRIP_TAC;;

e (ASSUME_TAC
   (SPECL [‘‘am:am‘‘,‘‘ev:string‘‘,‘‘id:string‘‘,‘‘s:store‘‘]
    file_get_audrecList_storage_addAudrec_new_audrec_1));;

e (ASM_UNDISCH_TAC 7);;

e (ASM_SIMP_TAC bool_ss []);;

e(SIMP_TAC list_ss []);;

e(STRIP_TAC);;

e (RL_TAC [storage_addAudrec_def]);;

e (RW_TAC bool_ss []);;

e (RL_TAC [audrec_get_event_file_addAudrec]);;

e (RL_TAC [audrec_get_event_storage_createFile]);;
```

```
e (RL_TAC [new_audrec_def,audrec_audrec_def]);;

e(SLICE_TAC);;

e(OBJ_SIMP_TAC);;

e (RL_TAC [audrec_get_event_file_addAudrec]);;

e (RL_TAC [audrec_get_event_storage_createFile]);;

e (RL_TAC [new_audrec_def,audrec_audrec_def]);;

e(SLICE_TAC);;

e(OBJ_SIMP_TAC);;

e (RL_TAC [audrec_get_event_file_addAudrec]);;

e (RL_TAC [audrec_get_event_storage_createFile]);;

e (RL_TAC [new_audrec_def,audrec_audrec_def]);;

e(SLICE_TAC);;

e(OBJ_SIMP_TAC);;

e (RL_TAC [audrec_get_event_file_addAudrec]);;

e (RL_TAC [audrec_get_event_storage_createFile]);;

e (RL_TAC [new_audrec_def,audrec_audrec_def]);;

e(SLICE_TAC);;

e(OBJ_SIMP_TAC);;

e (RL_TAC [audrec_get_event_file_addAudrec]);;
```

```
e (RL_TAC [audrec_get_event_storage_createFile]);;

e (RL_TAC [new_audrec_def,audrec_audrec_def]);;

e(SLICE_TAC);;

e(OBJ_SIMP_TAC);;

e (RL_TAC [audrec_get_event_file_addAudrec]);;

e (RL_TAC [audrec_get_event_storage_createFile]);;

e (RL_TAC [new_audrec_def,audrec_audrec_def]);;

e(SLICE_TAC);;

e(OBJ_SIMP_TAC);;

val storage_getHeadFile_storage_addAudrec_2 = prove
( ``!st s ar.
     storage_ex st s ==>
     let h = storage_getHeadFile st s in
     let max = storage_get_fileSize st s in
 ~storage_isEmpty st s /\  ~file_isFull h max s ==>
  (storage_getHeadFile st (storage_addAudrec st ar s) =
   storage_getHeadFile st s)``,
EVERY[
(RL_TAC [storage_addAudrec_def]),
(RW_TAC bool_ss []),
(RL_TAC [storage_getHeadFile_def]),
(RL_TAC [file_addAudrec_def]),
(SLICE_TAC)]
);;

e (ASSUME_TAC (
REWRITE_RULE [storage_ex_new_audrec, storage_isEmpty_new_audrec,
file_isFull_new_audrec, storage_getHeadFile_newaudrec,
storage_get_fileSize_new_audrec, file_new_new_audrec,
```

```
new_audrec_audrec_new]
(SPECL
[‘‘am_get_storage am s‘‘,
 ‘‘SND (new_audrec (am_getDate am s,am_getTime am s,ev,id) s)‘‘,
 ‘‘FST (new_audrec (am_getDate am s,am_getTime am s,ev,id) s)‘‘]
(LET_RULE storage_getHeadFile_storage_addAudrec_2))));;

e (ASM_UNDISCH_TAC 5);;

e(SIMP_TAC bool_ss []);;

e(STRIP_TAC);;

e(RES_TAC);;

e (ASM_SIMP_TAC bool_ss []);;

e (ASM_SIMP_TAC bool_ss [file_get_audrecList_storage_addAudrec_new_audrec_2]);;

e (SIMP_TAC list_ss []);;

e (RL_TAC [storage_addAudrec_def]);;

e (RL_TAC [storage_isEmpty_new_audrec,
          file_isFull_new_audrec,
          storage_getHeadFile_newaudrec,
          storage_get_fileSize_new_audrec]);;

e(ASM_SIMP_TAC bool_ss []);;

e (RL_TAC [file_addAudrec_def]);;

e(SLICE_TAC);;

e(RL_TAC [new_audrec_def, audrec_audrec_def]);;

e(SLICE_TAC);;
```

```
e(OBJ_SIMP_TAC);;
```

参考文献

[1] 矢竹健朗，青木利晃，片山卓也 コラボレーションに基づくオブジェクト指向理論の検証, コンピュータソフトウェア, Vol. 22, No. 1(2005), pp58–76.

[2] 矢竹健朗，青木利晃，片山卓也 定理証明システム HOL におけるオブジェクト指向理論の構築, 情報処理学会, ソフトウェア工学研究会　研究報告, 2002–, SE–138

[3] 徳田　拓 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工学専攻, 2006

[4] HOL4 http://hol.sourceforge.net/