修 士 論 文

匿名性プロトコルのOTS/CafeOBJ法に基づく形式化

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

程　剣

2008年3月

修 士 論 文

匿名性プロトコルのOTS/CafeOBJ法に基づく形式化

指導教官　緒方和博 特任助教授

審査委員主査　緒方和博 特任助教授
審査委員　二木厚吉 教授
審査委員　小川瑞史　教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

610058 程　剣

提出年月: 2008 年 2 月

概 要

近年，匿名性プロトコルに対する関心が高まっており，その安全性の検証が重要となっている．特に，インターネットを用いた電子投票システム，内部告発などへの応用が期待されているプロトコルの検証においては，形式手法が有効であると考えられている．

そこで本研究では，代数仕様言語 CafeOBJ を用いて匿名性プロトコルの仕様記述を行い，その検証の検討を行う．形式手法による匿名性プロトコルの検証に期待がなされ，いくつかの形式化が提案されている．その中から，NTT コンミュニケーション科学基礎研究所の河辺らが提案した入出力オートマトンで匿名性プロトコルをモデル化し，モデルを代数仕様言語 Larch で記述し，定理証明器 (Larch Prover) を使って匿名性シミュレーションと呼ばれる関係を用いたトレース匿名性の検証方法を参考にして，入出力オートマトンと観測遷移システムの類似性に着目し，観測遷移システムによる匿名性プロトコルのモデル化の方法を考案する．

そして，入出力オートマトンと観測遷移システムの類似点では，入出力オートマトンの状態の集合は観測遷移システムの状態空間 Y に対応している．入出力オートマトンの初期状態の集合は観測遷移システムの初期状態の集合に対応している．また入出力オートマトンのアクションの集合と遷移の集合は観測遷移システムの条件付遷移規則の集合に対応している．入出力オートマトンと観測遷移システムの相違点では，入出力オートマトンのアクションには外部と内部の区別があり，観測遷移システムの遷移には外部と内部の区別がないことがわかった．このため，観測遷移システムの遷移にも外部と内部の区別を与え，観測遷移システムのトレースを定義する必要がある．また，河辺らの匿名性シミュレーションと呼ばれる関係を用いたトレース匿名性の検証方法を参考にして，匿名性プロトコルの CafeOBJ 仕様に基づく証明譜による匿名性検証の可能性についても考察した．

OTS/CafeOBJ 法では，観測遷移システム (OTS：Observational Transition System) としてシステムやプロトコルの数学モデルを作成する．観測遷移システムは，代数仕様言語 CafeOBJ で記述され．観測遷移システムの CafeOBJ 仕様をもとに，システムやプロトコルが望みの性質を満たすことを検証する．匿名性プロトコルの OTS/CafeOBJ 法による仕様記述では，直感的に理解しやすいものとなっており，その可読性も高いものとなっている．記述した仕様は書き換え規則により自動的に状態遷移が行われる．その状態遷移の変化はシーケンスチャートに自然に対応させることができるため，遷移の様子を把握することは容易である．また，OTS/CafeOBJ 法は，ユニークな特徴 (システムやプロトコルの外部から観測可能な値の変化に着目したモデル化や可読性等に優れた証明譜による検証法など) を持つ．そして，本研究では，河辺らの提案方法を参考にして，OTS/CafeOBJ 法のユニークな特徴を活かした，匿名性プロトコルを観測遷移システムとしてモデル化し，CafeOBJ でその仕様を作成する方法を提案した．また匿名性プロトコルの CafeOBJ 仕様に基づく，証明譜による匿名性検証の可能性があることを確認した．

目 次

第1章　はじめに

匿名性の考え方は，投票，寄付，内部告発など，実世界のさまざまな場面に現れている．匿名性は，本人の発言や行動により本人が不利益を被らないように本人の身元を隠すことである．匿名性を利用すると，自由に発言ができたり，発言に対する責任が無くなる等が管理者により保障される．その人に関する情報が他の情報とつなぐことができるかで決まってくる．これは物理世界でも同じことである．特にインターネット上で見ているのは，互いの痕跡だけでありその人の身体的外見にアクセスできないから，それだけでは誰であるかということは特定できない．また，自分がもっている，相手の個人情報だけではその人がどんな人物なのかは分からない．その意味である特定の匿名性があると言える．

しかし，やはり他の点では匿名性はかなり薄くなっている．例えばスパイウェアを使えるものは，ユーザーのインターネット上の痕跡をたどることができるからだ．この意味では匿名性は失われていると言える．匿名性プロトコルは，安全な通信の実現が期待されたプロトコルであるが，仮にその安全性に不備な点があり，匿名要件が満たされなかった場合，その適用が期待される領域を考えると，社会経済に大きな損失を与える可能性がある．そのため，匿名性プロトコルがその匿名要件を満たし，安全であることの保証が必要とされる．

現在，匿名性を形式的に記述しその正しさを計算機を使って検証する研究が行われ始めた．しかし，分散システムが持つ匿名性を厳密に証明する手法は，まだ確立されていない．

ソフトウェア工学の分野では，プログラムや仕様を形式的に記述し，その正しさを計算機を使って検証する方法が知られている．同様の考え方に基づいて，匿名性などを形式的に記述し証明する研究も現れている．しかし，定理証明器やモデル検査器など，計算機を用いた検証の例は少ない．

ロンドン大学の Steve Schneider らは，FDR(failures-divergences refinement) を用いた有限状態システムのトレース匿名性のモデル検査器による検証方法 [1] を提案している．

NTT コミュニケーション科学基礎研究所の河辺ら [2][3][4][5] は，匿名性シミュレーションと呼ばれる関係を用いたトレース匿名性の検証方法を提案している．この検証では，入出力オートマトンでプロトコルをモデル化し，モデルを代数仕様元言語 Larch で記述し，定理証明器 LP(Larch Prover) を用いてプロトコルがトレース匿名性を満たしていることを証明する．

一般に，ソフトウェアの開発は，仕様を基に行なわれるため，開発全体のコストを減らすためには誤りのない仕様を作成することが非常に重要である．仕様記述の記法に形式手法がある．形式手法とは，コンピュータサイエンスにおける数学を基盤としたシステムの

仕様記述，開発，検証の技術である．そのアプローチは高度な安全性などを求められるシステムでは特に重要であり，開発段階で誤りのないことを保証する．

匿名性プロトコルを実用化する場合，あらかじめ，その安全性を保証する必要がある．しかし，それはしばしば人間の直感に頼っていた．形式手法による検証にとって，良く知られたプロトコルの不具合が報告され，匿名性プロトコルの検証方法として，形式手法への期待が高まっている．

本稿では，はじめに匿名性プロトコルの具体例として $JukeBox$ についての説明を行い，$JukeBox$ 上における匿名要件を明らかにし，これらの問題に対処するためのプロトコルである，匿名性プロトコルについて説明する．次に，形式手法および本研究で匿名性プロトコルの仕様記述に用いる代数仕様言語 CafeOBJ について説明する．また，観測遷移システム (OTS) による匿名性プロトコルのモデル化・仕様作成方法について説明する．

そして，観測遷移システムで匿名性プロトコルをモデル化し，CafeOBJ でその仕様を作成する方法を提案し，また，匿名性プロトコルの CafeOBJ 仕様に基づく証明譜による匿名性検証の可能性についても考察する．

第2章　準備

本章では，分散システムを状態機械として表現する観測遷移システムと，それを記述し，検証するための言語 CafeOBJ を説明する．

2.1　観測遷移システム (OTS)

コンピュータ化されたシステムを安全かつ安心に構築することを部分的に支援する方法の一つに，形式検証がある．形式検証により，システムが望みの性質を満足していること等をきちんと確認することができる．システムが望みの性質を満足していること等を形式検証により確認するには，システムをモデル化する必要がある．システムのモデルとして，観測遷移システム (Observational Transition Systems)[6] を用いる．

観測遷移システム (Observational Transition Systems) [6] は，システムの実行に伴い外部から観測可能な値がどのように変化するかに着目することで，システムをモデル化する．観測遷移システム (Observational Transition Systems) は，観測関数が観測等価を保存する振舞仕様であり，CafeOBJ による抽象機械の仕様作成・検証に有効な概念である．振舞仕様が OTS であるためには，観測関数が観測等価を保存する必要がある．すなわち，観測等価な状態 s, s' と遷移関数 t に対し，状態 $t(s)$ と状態 $t(s')$ は観測等価でなければならない．状態 s と s' の観測等価は，任意の観測関数 o に対して $o(s) = o(s')$ が成り立つことで定義される．さらに，初期状態 init に対して，すべての観測関数 o で $o(init)$ が定義されていれば，抽象機械における初期状態からの任意の実行列 $t_1, t_2, \ldots, t_n$ に対して，実行列を適用後の状態 $t_n(\ldots(t_1(init))\ldots)$ の観測値が決定する．

観測遷移システム S は三つ組 $OTS_S = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ で定義する．なお，以下の定義中，ソート (または型) として，S の状態空間 Y，データ $D, D_K(k = x_1, \ldots, x_m, y_1, \ldots, y_n)$ を用いる:

$\mathcal{O}$ **：観測の集合:** 各 $o \in \mathcal{O}$ は，状態空間を引数にとり，データを返す関数 $\mathcal{O}$: Y → D である (観測ごとに値域は異なってもよい)．S の 2 つの状態 $u_1, u_2 \in$ Y の等価性 $u_1 =_s u_2$ は以下のように定義する: $u_1 =_s u_2$ iff $\forall$ o $\in \mathcal{O} \in .\ o(u_1) = o(u_2)$ ただし，$o(u_1) = o(u_2)$ における'=' は各 $o \in \mathcal{O}$ の値域ごとに定義されているとする．

$\mathcal{I}$ **：初期状態の集合:** $\mathcal{I}$ は状態空間の部分集合である．

$\mathcal{T}$ **：条件付遷移規則の集合:** 各 $t \in \mathcal{T}$ は，ある状態から次の状態を返す関数 t : Y → Y である．

類似した観測や遷移規則を，添字を伴った $o_{x_1,\ldots,x_m}$ や $t_{y_1,\ldots,y_n}$ として定義することがある (ただし,$m,n \geq 0$) ．これらの演算は，集合 $\{o_{x_1,\ldots,x_m} : \mathrm{Y} \to D | x_1 \in D_{x_1}, \ldots, x_m \in D_{x_m}\}$ 内の観測や，集合 $\{t_{y_1,\ldots,y_n} : \mathrm{Y} \to \mathrm{Y} \ | y_1 \in D_{y_1}, \ldots, y_m \in D_{y_m}\}$ 内の遷移規則を表している．

遷移規則 $t \in \mathcal{T}$ の意味は，効果と効力条件の組で定義する．効果は, 各 $o \in O$ の変化である．効力条件は，述語 $\text{c-t}_{y_1,\ldots,y_n} : \mathrm{Y} \to \{true, false\}$ であり，$\text{c-t}_{y_1,\ldots,y_n}$ が真である状態で t が適用されると，各 $o \in O$ は t の効果で定義した値に変化する．そうでない時は，各 $o \in \mathcal{O}$ は変化しない．

振舞遷移機械の計算は，次の 3 つの条件を満たす状態の無限列 $u_0, u_1, \ldots$ である．

開始性: 状態 u_0 において，各 $o \in O$ は $\mathcal{I}$ を満たす．

一貫性: すべての $i \in 0, 1, \ldots$ において，$u_{i+1} =_s t(u_i)$ となる遷移規則 $t \in \mathcal{T}$ が存在する．

公平性:各 $t \in \mathcal{T}$ に対し，$u_{i+1} =_s t(u_i)$ となる $i \in 0, 1, \ldots$ が無限に存在する．

振舞遷移機械の安全性，およびその定義で用いる到達可能性は，以下の定義とする．

到達可能性: S の計算に状態 $u \in \mathrm{Y}$ が現れる時，u は到達可能である．

安全性: S のすべての到達可能な状態 $u \in \mathrm{Y}$ において，述語 $p : \mathrm{Y} \to \{true, false\}$ が真である時，S は安全性 p を有する．

本稿では，S が匿名性 p を有することを，遷移規則の適用回数に関する帰納法を用いて検証する．帰納法の手続きにおいては，以下のことを確かめる．

基底: $\mathcal{I}$ を満たす任意の状態 $u \in \mathrm{Y}$ で，$p(u)$ が成立する．

帰納段階: 任意の到達可能な状態 $u \in \mathrm{Y}$ について，$p(u)$ が成立するならば，各 $t \in \mathcal{T}$ を適用後の状態 $t(u)$ においても $p(t(u))$ が成立する．

簡単な銀行口座 (図 2.1) を例として用い，観測遷移システムでのモデル化を例示する．銀行口座は，観測関数として口座の金額を調べる $balance$，観測関数として口座の預金をする $deposit$ と口座から引き出す $withdraw$ がある．$Account$ を表す状態空間 Υ の存在を仮定する．銀行口座をモデル化した観測遷移システム S_{BA} は以下のとおりである．

観測 $\mathcal{O}$ は次のように表現できる．

$S_{BA} \triangleq \{balance : \Upsilon \to Int\}$

銀行口座の現在金額を金額表す値 Int で観測する．初期状態 $\mathcal{I}$ は，任意の状態 $\upsilon \in \mathrm{Y}$ に対し，次のように表現できる．

$S_{BA} \triangleq \{\upsilon_{init} \in \Upsilon \mid balance(\upsilon_{init}) = o\}$

観測規則 $\mathcal{T}$ は，2 つの遷移規則の集合で表現する．

$S_{BA} \triangleq \{deposit : \Upsilon \to \Upsilon, withdraw : \Upsilon \to \Upsilon\}$

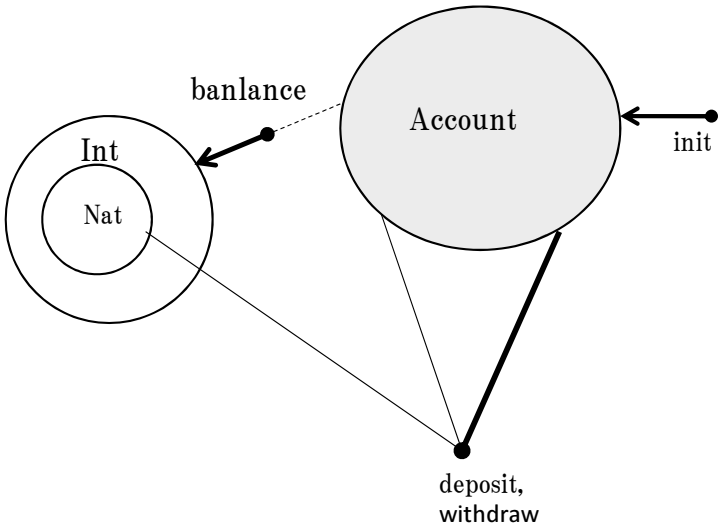

図 2.1: *Account*

2.2 代数仕様言語 (CafeOBJ)

CafeOBJ 言語 [8][9][10] は主に始代数 (initial algebra) と隠蔽代数 (hidden algebra) に基づく仕様記述言語である．始代数は整数やスタックなどの抽象データ型の記述に，隠蔽代数は抽象機械の記述に用いる．

2.2.1 代数仕様による抽象データ型の仕様

抽象データ型は，データとデータに対して行うことができる操作の集合についての記述である．このようなデータ型は様々な実装から独立しているという意味で抽象である。定義は数理的なものか，インタフェースとして記述することができる．抽象データ型は，その使用者に対して型名の参照と提供している操作の呼び出しのみを許し，使用上必要な情報のみをインターフェースとして公開する．そのため，内部の実現方法については外部から完全に隠蔽される．代数仕様では，抽象データ型のインターフェースを表すことができる．

例として，自然数を CafeOBJ で記述する．

```
mod! NAT-NUMBERS {
[Nat]
op zero : -> Nat
op succ : Nat -> Nat
op _+_ : Nat Nat  -> Nat
```

```
 op _=_: Nat Nat  -> Bool {comm}
 vars X Y : Nat
 eq  zero + X = X .
 eq  succ(Y) + X  = succ(Y + X) .
 eq (X = X) = true .
 eq (succ(X) = zero) = false .
 eq (succ(X) = succ(Y)) = (X = Y) .
}
```

CafeOBJ では，仕様をモジェールを単位として記述する．モジェールの宣言は，mod, mod!，mod*のいずれかで行うが，抽象データ型の仕様を記述するときには，mod!を用いる．mod!の後に続く NAT-NUMBERS はこのモジェールの名前である．[ ] で囲まれた Nat が型を表すソート名である．このモジェールは，シグネチャと公理からなる．シゲネチャ：[...] はソートおよびその上の順序関係の宣言である．ソート Nat が宣言されている．op はオペレータ宣言のキーワードである．op の後に，演算記号，:，引数ソート，− >，結果ソートの順で宣言する．この引数ソートと結果ソートの組をランクと呼ぶ．CafeOBJ では，オペレータの引数の位置を自由に指定することができる．例えば，中置演算子の + や，より複雑な演算子を宣言できる．オペレータの位置はオペレータ名の中の __で表される．`op _+_: Nat Nat -> Nat` の意味は二つの引数ソート Nat を相加し，結果ソート Nat になる．オペレータには，assoc：結合律，comm：交換律など属性を指定できる．例えば，自然数の仕様に現れる，= が交換律を持つことが証明できたならば，= は属性指定を使って次のように宣言し直すことができる: `op _=_: Nat Nat  -> Bool {comm}`．また，予約語 var で等式の外で変数を宣言する．同じソート上の複数の変数は vars を使って宣言できる．var(vars) の後に (変数名，: ，ソート名) の順で宣言する．ここで各等式中の X，Y を X:Nat，Y:Nat と宣言したのと同じ意味を持つ．

CafeOBJ では等式に条件を付けることができる．等式は，二つの項上 t と t' の間の等価関係を次のように宣言する．

```
eq t = t'.
```

一つの等式毎に「.」で宣言の終わりを表す必要がある．また，条件付き等式を表す予約語として ceq がある．条件付き等式は，ceq l = r if C の形をしており，l = r は等式,C は BOOL 項である．すべてのモジェールは，暗黙に BOOL を輸入しているため，条件付き等式を記述することができる．次のように宣言する:

```
ceq t = t' if C .
```

仕様の実行

CafeOBJ は, 対話型の処理系を持つ．CafeOBJ 起動後にプロンプト "CafeOBJ> "が表示されるので，そこにモジェールを直接またはファイル経由で読み込ませることができ

る. select コマンドでモジェールを選択できる（プロンプトは ”モジェール名 > ”に変わる). 選択中のモジェールにおいて，項の構文解析コマンド parse が使える. 実行例を示す.

```
-- defining module! NAT-NUMBERS......_.....* done.
CafeOBJ> select NAT-NUMBERS .
NAT-NUMBERS> parse 0:Nat .
(0):Nat
NAT-NUMBERS> parse X:Nat .
(X):Nat
NAT-NUMBERS> parse succ(X) .
(succ X):Nat
NAT-NUMBERS> parse X + Y:Nat .
(X + Y):Nat
NAT-NUMBERS> parse X + succ(X) .
(X + (succ X)):Nat
```

CafeOBJ は，実行可能な仕様言語である. 仕様の実行は, 項の簡約として行われる. 項の簡約は，公理の等式を左辺から右辺への書換規則と見なし，規則を順次適用して項を書き換えていくことで行われる. 簡約は，Cafeobj 処理系の簡約コマンド red により行われる. 次の例では，仕様 NAT-NUMBERS に対する red コマンドを使って 3+4 を表す項を簡約する.

```
NAT-NUMBERS> red succ(succ(succ(zero))) + succ(succ(succ(succ(zero)))) .
-- reduce in NAT-NUMBERS : succ(succ(succ(zero))) +
                           succ(succ(succ(succ(zero))))
(succ (succ (succ (succ (succ (succ (succ (zero)))))))):Nat
(0.000 sec for parse, 4 rewrites(0.000 sec), 7 matches)
```

2.3 CafeOBJ による観測遷移システムの記述

観測遷移システム S は，代数仕様言語 CafeOBJ で記述する. 以下では,S の記述に用いる各データ型 D および $D_K(k = x_1, \ldots, x_m, y_1, \ldots, y_n)$ は，始代数に基づいて適切に記述されており，それらを表す可視ソート V および $V_K(k = x_1, \ldots, x_m, y_1, \ldots, y_n)$ の存在を仮定する. 状態空間 Y を表す状態ソート H を $*[H]*$ と宣言する.

S の観測 $o_{x_1,\ldots,x_m} \in \mathcal{O}$ は CafeOBJ の観測演算で次のように表現する:

bop o : $HV_{x_1} \ldots V_{x_n}$ -> V

初期条件 $\mathcal{I}$ は，状態定数を宣言し，初期条件が表す各観測値を等式で表現する. 初期状態を表す状態定数 init を次のように宣言する.

bop init : -> H

観測 $o_{x_1}, \ldots, x_m \in \mathcal{O}$ に対して，初期値が $f(x_1, \ldots, x_m)$ で与えられると，初期状態の観測値を定義する等式を次のように宣言する．

eq $o(init, x_1, \ldots, x_m,) = f(x_1, \ldots, x_m)$.

S の遷移規則 $t_{y_1,\ldots,y_n} \in \mathcal{T}$ は CafeOBJ の操作演算で次のように宣言する．

bop $t : H\ V_{y_1} \ldots V_{y_n}$ -> H

各 $t_{y_1,\ldots,y_n} \in \mathcal{T}$ の効力条件は，次のように表現する．

op $c_t : H\ V_{y_1} \ldots V_{y_n}$ -> Bool

各 $t_{y_1,\ldots,y_n} \in \mathcal{T}$ が各 $o_{x_1,\ldots,x_m} \in \mathcal{O}$ に与える効果は，次のように宣言する．

以下の等式中，$x_k(k = x_1, \ldots, x_m, y_1, \ldots, y_n)$ は，可視ソート V_k 上の，h は状態ソート H 上の CafeOBJ 変数とする．効力条件が真である状態における遷移規則 $t_{y_1,\ldots,y_n}$ の実行による観測 $o_{x_1,\ldots,x_m}$ の変化は，c_t と $e_{(t,o)}$ を用いて，次のように記述する:

ceq $o(t(h, x_{j_1}, \ldots, x_{j_n}), x_{i_1}, \ldots, x_{i_m})$
$= e_{(t,o)}(h, x_{j_1}, \ldots, x_{j_n}, x_{i_1}, \ldots, x_{i_m})$
if c-t$(h, x_{j_1}, \ldots, x_{j_n})$.

同様に，効力条件が偽である状態においては，$t_{y_1,\ldots,y_n}$ は観測値を変化させないので，次のように記述する．

ceq $t(h, x_{j_1}, \ldots, x_{j_n}) = h$
if not c-t$(h, x_{j_1}, \ldots, x_{j_n})$.

以上の手続きにより，振舞遷移機械 S から CafeOBJ の仕様を得ることができる．

例として，2.1 節で紹介した銀行口座をモデル化した観測遷移システム S_{BA} を CafeOBJ で記述する．

```
mod* ACCOUNT
  protecting(INT)

  *[ Account ]*
   op init : -> Account
   bop balance : Account -> Int
   bop deposit  : Account Nat -> Account
   bop withdraw  : Account Nat -> Account

   var N : Nat
```

```
var A : Account
eq balance(deposit(A,N)) = balance(A) + N .
ceq balance(withdraw(A,N)) = balance(A) - N
                             if N <= balance(A) .
ceq balance(withdraw(A,N)) = balance(A)
                             if not(N <= balance(A)) .
```

モジェール ACCOUNT は，モジェール NAT-NUMBERS を輸入している．輸入を含むモジェールに対応する仕様は輸入されている仕様の要素を含む．CafeOBJ では，組み込みのモジェールや既に定義されたモジェールを他のモジェールに輸入して再利用が可能である．モジェールを輸入するための予約語として，protecting，extending，using が用意されそれぞれ，輸入先のモジェールでどのような意味を持つかという点で異なっている．*[ ]*で囲まれた Account が隠蔽ソートを表し，S_{BA} の状態集合を表す．定数 init は，S_{BA} の任意の初期状態を表す．観測演算と操作演算は，キーワード'op' ではなく'bop' を用いて定義する他は，通常の演算と同様に定義する．状態定数は，ある条件を満たす状態の集合の代表元を表す，状態ソート上の定数である．主に，初期状態の定義や，検証時に議論の対象とする状態空間を特定するために用いる．状態定数も，通常の定数と同様にキーワード'op' を用いて宣言する．

演算 balance は，S_{BA} の観測 balance に対応し，観測演算と呼ばれる．演算 withdraw は，S_{BA} の遷移 withdraw に対応し，遷移演算と呼ばれる．

初期状態の定義: 任意の初期状態を表す定数 init は以下のとおりに定義される．

```
op init : -> Account
eq balance(init) = 0 .
```

この等式は $\mathcal{I}$ に対応する．

遷移演算 withdraw の定義:遷移演算 withdraw を定義する等式は以下のとおりである．

```
ceq balance(withdraw(A,N)) = balance(A) - N
                             if N <= balance(A) .
ceq balance(withdraw(A,N)) = balance(A)
                             if not(N <= balance(A)) .
```

N <= balance(A) は，遷移演算 withdraw の効力条件である．最初の等式は，条件を満たされた場合に何が起きるかについて定義されている．条件が満たされれば，事後状態 withdraw(A,N) において，balance は，自然数 N に対して，balance(A) - N を返す．最後の等式で，条件が満たされない場合，何も起きないことが定義されている．

第3章　検証の手法

現在，計算機による検証に関する研究は注目されている．証明対象も数学，ソフトウェアの正しさ，システムの正しさなど様々である．検証するには，対象とするものの仕様を作成する必要がある．仕様は，あるシステムが何をすべきかを記述するものであり，どのように実装すべきかを記述する必要はない．そのような仕様を与えることにより，対象システムが仕様に照らして正しいかどうかを形式的検証技法で判定することが可能となる．このようなシステム実装を開発する際に使用される数学的記述を形式的仕様と呼ぶ．計算機を利用し，この形式仕様がある命題を満たすかどうかについて証明する．この証明方法は大きく 2 つに分けることができる．

一つは形式仕様に対して，演繹的手続きで命題の真偽を導出することで証明する方法である．この証明方法は帰納法による検証の手法と呼ぶ．もう一つは有限な状態遷移系が，時相論理式で表された性質を満足するかどうかを判定する数理的手法である．この検証方法はモデル検査の手法と呼ぶ．モデル検査の特徴では，与えられた性質が成り立つかどうかの判定する際に，システムのとりうる全状態を網羅的に探索することである．

Spin はソフトウェアのモデル検査のためのツールである．検査対象のシステムは Promela (Process Meta Language) で記述される．Promela は非同期分散アルゴリズムを非決定性オートマトンとしてモデル化する．検証される属性は線形時相論理の論理式で表され，その否定形を Buchi オートマトンに変換してモデル検査アルゴリズムの一部とする．

本章では主に帰納法による検証の手法を述べる．

3.1　帰納法による検証の手法

帰納法による検証の手法による検証は，無限状態を持つ問題に対しても可能であるなど，非常に強力で広範な問題を取り扱うことができる．帰納法による検証では，一般に，人間のユーザーがシステムにヒントを与える必要がある．証明中に誤りなどがあったら，仕様書を修正する必要等がある．そして修正した仕様に対して検証を再度行う．このような作業を繰り返し行うことで，正し証明を構築していく．証明検証系は，公理と推論規則から演繹的に証明していく．

以下では帰納法による検証の手法による検証器についていくつが取り上げ，簡単に説明する．

PVS(Prototype Verification System) は，計算機上で特定の数学的議論を形式化し，形式化された命題を機械的に証明するためのシステムである．PVS は，数学的概念を表現

する機能を持つ. 人間が数学的概念を実装しやすく，理解しやすい言語と形式化された命題を計算機上で形式的に証明するための環境を提供することできる．特に，対話的な証明支援ツールとして，証明の作成において，各作業ごとにユーザがツールに与える指示がチェックされ，ユーザは自分の指示の間違いをすぐに修正できる，という利点を持っている．

Coq は人間と協力して数学的な定理を定式化し，証明することができる. Coq は 2 つの部分からなり，1 つは証明をつくる部分，もう 1 つは証明が正しいかどうかチェックする部分である. 証明をチェックする部分はラムダ計算の型理論に基づいていて，そのチェックが通ればその正しさが保証されることが証明されている. Coq を応用すればユーザの仕様を数学の命題として定式化し，証明することができる. また，Coq では仕様が証明できたら，ワンタッチでユーザを取り出すことができます。このとき得られたユーザは完全に仕様を満たすことが保証されます。

CafeOBJ は代数に基づく検証器である. 仕様はシグニチャと等式からなるモジェール単位で記述する. シグニチャは項を構成する定数と演算からなり，等式は項の間の等価関係を記述する. CafeOBJ では証明したい性質を項として表現し，等式を簡約規則として項を繰り返し簡約させる. 証明が成功する場合，簡約の結果 true が得られる. true が得られず簡約が途中で停止した場合，必要な補題を等式として追加し簡約を再度試みる.

3.2 場合分け

本研究は，証明譜を用いて CafeOBJ で帰納法を行う. 帰納法で証明したい性質を $prop$: Y $\rightarrow \{true, false\}$ で表す．帰納法は，基底と帰納段階の 2 つの段階に分けられる．基底については，初期状態を表す各状態定数 s において $prop(s)$ が成立することを確かめればよい．帰納段階については，まず，場合分けを行う．帰納段階の証明節を作成するときは，まず，効力条件に基づき場合分けを行う必要がある. さらに，効力条件が成り立つ場合に CafeOBJ が true でも false でもない Bool の項を返せば，さらに場合分けをする必要がある.

場合分けとは，状態空間に関する適当な述語を用いて状態空間を分割することである. 場合とは，分割された各状態空間のことである．ある条件を満たす任意の状態を表す状態ソートの定数を状態定数と呼ぶ．1 つの状態定数は，1 つの場合を表現している.

場合分けは，構成子に基づいて行われる. ソート S の構成子は以下のとおり宣言されているとする.

op C_1 : $S_{11} \ldots S_{1_{m_1}}$ -> S
$\ldots$
op C_n : $S_{n1} \ldots S_{n_{m_n}}$ -> S

ソート S の項 t に基づき証明節を分割すると，n 個の証明節が得られる. i 番目の証明節には，以下の等式が宣言される.

$$\text{eq t} = c_i(d_{i1}, \ldots d_{i_{m_i}}) \ .$$

ここで, 各 d_{ij} は，ソート S_{ij} の定数で，そのソートの任意の値を表す. ソート S が Bool の場合，2 つの証明節に分割される. それぞれの証明節で，t = true と t = false が宣言される. t が l = r の形をしていれば，(l = r)= true の代わりに，l = r(あるいは r = l) を使うことが多い.

恒真式をもとに場合わけを行うこともある. $t_1$or ... or t_n の形をした Bool の項が恒真であるとき，この項に基づき場合分けを行うと，n 個の証明節が得られる. i 番目の証明節には t_1 = true が宣言される.

恒真式に基づく場合分けは，Bool の構成子 (true と false) に基づく場合分けの一般化とみなせる. というのは，t を Bool の項とすると，(t = true)or(t = false) は恒真であるからである. また，恒真式に基づく場合分けは,Bool の構成子に基づく場合分けとして説明できる. i 番目の証明節に i 個の等式 t_1 = false，...,t_{i-1} = false，t_i = true を宣言すると，Bool の構成子に基づく場合分けになる. i 番目の証明節から最初の $i-1$ 個の等式をコメントしたとしても，証明節の役割に本質的な支障はない. そして，そうすると，恒真式に基づく場合分けになる.

3.3 帰納法による検証の例

本節は，帰納法を用いた証明の例を紹介する.

証明 Theorem ∀x, y :Nat. (x + y = y + x).

この定理を証明するとき，**lemma1 ∀x，:Nat.(zero + x = x)** と **Lemma2 ∀x, y : Nat. (succ(x) + y = succ(x + y)** を必要とする. そこで，まず，これら 2 つの補題を証明する.

- lemma ∀x, :Nat.(zero + x = x) まず以下のモジェールを宣言する.

```
mod LEMMA1 {
    pr(NAT-NUMBERS)
    -- arbitrary values
    op x : -> Nat
    -- lemmas
    op lemma1 : Nat -> Bool
    -- CafeOBJ variables
    var X  : Nat
    -- definitions of the lemmas
    eq lemma1(X) = (X + zero = X ) .
}
```

各帰納段階で証明すべき論理式を記述したモジェールを以下のとおりに宣言する.

```
mod ISTEP1 {
   pr(LEMMA1)
   -- arbitrary values
   op x' : -> Nat
   -- formulas to prove in induction cases
   op istep1 : -> Bool
   -- CafeOBJ variables
   var X  : Nat
   -- definitions of the formulas
   eq istep1 = lemma1(x) implies lemma1(x') .
}
```

定数 x は任意の自然数を表し，定数 x' は自然数 y の事後自然数を表す. 項 lemma1(x') は各帰納段階で証明すべき論理式である. 項 lemma1(X) は帰納法の仮定としてよく使うため，演算 istep1 を上記のように定義する.

lemma1 の証明譜

- 基底段階の証明節
  open LEMMA1
  red lemma1(zero) .
  close

  この証明節に対し，CafeOBJ は true を返す.
- 帰納段階の証明節
  open ISTEP1
  – aribitrary values
  – assumptions
  – next natural number
  eq x' = succ(x) .
  – check
  red istep1 .
  close

- Lemma2 $\forall$x, y : Nat. (succ(x) + y = succ(x + y)).

まず以下のモジェールを宣言する.

```
mod LEMMA2 {
    pr(NAT-NUMBERS)
    -- arbitrary values
    ops x y : -> Nat

    -- lemmas
    op lemma2 : Nat -> Bool

    -- CafeOBJ variables
    vars X Y : Nat

    -- definitions of the lemmas
    eq lemma2(Y) = (Y + succ(x)  = succ(Y + x)) .

}
```

各帰納段階で証明すべき論理式を記述したモジェールを以下のとおりに宣言する.

```
mod ISTEP2 {
   pr(LEMMA2)
   -- arbitrary values
   op y' : -> Nat
   -- formulas to prove in induction cases
   op istep2 : -> Bool

   -- CafeOBJ variables
   vars X Y : Nat

   -- definitions of the formulas
   eq istep2 = lemma2(y) implies lemma2(y') .
}
```

定数 y は任意の自然数を表し，定数 y' は自然数 y の事後自然数を表す. 項 lemma2(y') は各帰納段階で証明すべき論理式である. 項 lemma2(y) は帰納法の仮定としてよく使うため，演算 istep2 を上記のように定義する.

lemma2 の証明譜

– 基底段階の証明節
  open LEMMA2

red lemma2(zero) .
close

この証明節に対し，CafeOBJ は true を返す.

– 帰納段階の証明節
open ISTEP2
– aribitrary values
– assumptions
– next natural number
eq y' = succ(y) .
– check
red istep2 .
close

基底段階は，lemma2(zero) の証明である. 帰納段階では，eq y' = succ(y) を満たす任意の y' について lemma2(y') が成り立つをことを示している. これは帰納法の仮定 lemma2(y) のもとで lemma2(y') を示したことと等しい. よって，任意の自然数 m について lemma2(m) が証明できた.

続いて，2 つの補題を用いて，**Theorem** ∀x, y :Nat. (x + y = y + x). を証明する.

- Theorem ∀x, y :Nat. (x + y = y + x).
まず以下のモジェールを宣言する.

```
mod THEOREM {
    pr(NAT-NUMBERS + LEMMA)
    -- arbitrary values
    ops x y : -> Nat
    -- theorems
    op theorem : Nat Nat -> Bool
    -- CafeOBJ variables
    vars X Y : Nat
    -- definitions of the theorems
    eq theorem(X,Y) = ( X + Y = Y + X ) .
}
```

各帰納段階で証明すべき論理式を記述したモジェールを以下のとおりに宣言する.

```
mod ISTEP3 {
   pr(THEOREM)
   -- arbitrary values
   op y' : -> Nat
   -- formulas to prove in induction cases
   op istep3 : -> Bool
   -- CafeOBJ variables
   vars X Y : Nat
   -- definitions of the formulas
   eq istep3 = theorem(x,y) implies theorem(x,y') .
}
```

定数 y は任意の自然数を表し，定数 y' は自然数 y の事後自然数を表す．項 theorem(x,y') は各帰納段階で証明すべき論理式である．項 theorem(x,y) は帰納法の仮定としてよく使うため，演算 istep3 を上記のように定義する．

Theorem の証明譜

- 基底段階の証明節
  open THEOREM
  red lemma1(x) implies theorem(x,zero) .
  close

  この証明節に対し，CafeOBJ は true を返す．
- 帰納段階の証明節
  open ISTEP3 + LEMMA2
  – aribitrary values
  – assumptions
  – next natural number
  eq y' = succ(y) .
  – check
  red istep3 .
  close

例として，定理証明器 CafeOBJ を使って帰納法による数学定理を証明した．このあとの章では，匿名性プロトコルを対象として，その仕様を形式論理の命題として記述し，それを定理証明器 CafeOBJ を使って匿名性プロトコルが仕様を満たすことを数学的に証明して行く．

第4章　匿名性プロトコル

この章では，匿名性を解説し，代表的ないくつの匿名性プロトコルを紹介する．また，匿名性を確保するための匿名性プロトコルについて説明する．

4.1　匿名性

匿名（とくめい）とは，何らかの行動をとった人物が誰であるのかが分からない状態をさす．自分の実名・正体を明かさない事を目的とする．匿名性が高いインターネットでは，自分の名前や身元を隠すことで，自由な発言ができるメリットがある．自分がどこの誰かわからない，というのは，プライバシー保護の観点からも重要である．プライバシーとは，他人の干渉を許さない，各個人の私生活上の自由のことであるが，現在は「自己情報コントロール権」として自己の情報をコントロールする権利である．

しかし一方では，その匿名性を悪用された場合，例えば自身の情報を偽って登録することが可能となる．それをネット上の掲示板などで他人の誹謗中傷を書き込んだり，電子商取引の場で悪用して架空の注文をしたり，ということもある．インターネットの場合，直接相手方と対等で事を行うことがほとんどなく，現在のインターネットプロトコルおよび多くのアプリケーションでは，掲示板やブログなどの情報発信元を正確に指摘することが困難であるため，トラブルが発生した場合，その相手方を容易には特定できず，当事者間での解決が難しくなる．以下，代表的ないくつの匿名性プロトコルを紹介する．

電子投票プロトコル [3] では投票者のプライバシーを保証する為，匿名性が要求される．次の 2 つの要件が数学的に保証されるとき，電子投票プロトコルは，安全であるという．一つはどの投票者が誰に投票したのかは誰にも分からない．もう一つの要件は投票結果は正しく集計される。

I2P とは，ネットワークによる通信の始点と終点を匿名化し，端点間の通信内容も暗号化するという方法で匿名化されたネットワークを構成するプロトコルである．パソコン通信では，通常各個人に対して一つの ID が発行されていた．この環境で，ID を使用しないで活動することは難しかった．また，通常他人は書き込み者の ID も分かるようになっており，最終的にはその ID のもとで自身の責任を取ることになっていた．

I2P における通信では端点は暗号化された識別子によってネットワーク上で一意に区別される．TCP/IP による通信がホスト名とポート番号によって一意に識別される事と類似している．この I2P の端点識別子からは IP アドレスを知る事ができないため，ネットワークの利用者，サービス提供者ともに匿名での通信が可能になっている．

I2P は既存の TCP/IP ネットワークの上にオーバレイ (overlay) されたネットワークとして機能する. 現在アプリケーションを I2P ネットワークを通信路として利用できるようにするためのブリッジソフトウェア I2PTunnel が提供されている. この I2PTunnel を利用すると，IP アドレスを公開する事なく WWW サーバや IRC サーバを提供する事ができ，利用者は IP アドレスを秘匿してサービスを受ける事ができる。

4.2 *JukeBox*

本研究では，匿名性を説明するために，単純な例 *JukeBox* を考える.*JukeBox* を図示したものを (図 4.1) に示す. *JukeBox* はある建物に置かれているとする. 誰かが n 個のコインを *JukeBox* に入れたら，*JukeBox* はワイヤレスで n 個の歌のデジタルデータを配信する. 音楽を聞くために，建物の人々は PDA のような装置を使う. ここで，あるニ人 *Alice* および *Bob* がいる. 彼らは匿名でコインを *JukeBox* に入れて音楽を聞く. 彼らは *JukeBox* を利用したことをだれにも知られたくない. *Alice* は最後必ず *jazz* を選択する. *Bob* は最後必ず *rock* を選択する.

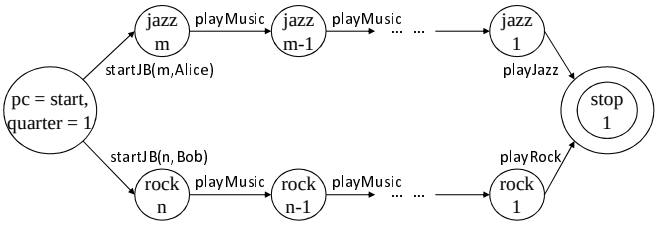

図 4.1: *JukeBox*

第5章　入出力オートマトンによる匿名性プロトコルのモデル化

この章では，まず$JukeBox$を入出力オートマトンで定義し，続いて混乱の原理 (principle of confusion) に基づく匿名性の定義を導入する．最後トレース匿名性の定義を紹介する．

5.1　入出力オートマトン

オートマトンでは，入力に対して内部の状態に応じた処理を行ない，結果を出力する仮想的な自動機械である．複数の状態と，それぞれの状態で入力結果に対してどのような処理を行うかを定めた関数とで構成されている．コンピュータの数学的なモデルとも言えるもので，入出力オートマトンもオートマトンの一種である．

入出力オートマトンXは，アクションの集合$sig(X)$，状態集合$state(X)$，初期状態集合$start(X) \subset state(X)$および遷移の集合$trans(X) \subset state(X) \times sig(X) \times state(X)$から成る．アクションには，入力，出力，内部の3種類があり，それぞれの集合を$in(x), out(x), int(X)$と書く (これらは，互いに素な集合とする)．また，入力アクションと出力アクションを，外部アクションと呼ぶ．Xの遷移$(s, a, s') \in trans(X)$を,$s \xrightarrow{a}_X s'$とも書く．また，aが内部アクションのとき，$s \longrightarrow_X s'$とも書く．関係$\rightsquigarrow$を，関係$\longrightarrow$の反射の推移の閉包とする．

任意の$a \in sig(X)$および$s, s' \in states(X)$に関して，$s \stackrel{a}{\Rightarrow}_X s'$は以下を表す．

- aが内部アクションのときは，$s \rightsquigarrow_X s'$．
- aが外部アクションのときは，ある s1,s2 ∈ states(X) が存在して，$s \rightsquigarrow_X s_1 \xrightarrow{a}_X s_2 \rightsquigarrow_X s'$．

遷移列（ ただし，s0 ∈ start(X)）$a \equiv s_1 \xrightarrow{a_1}_X s_1 \xrightarrow{a_2}_X \ldots \xrightarrow{a_n}_X s_n$ に関して，aに現われるすべての外部アクションを順に並べたものをaのトレースと呼ぶ．Xが持つすべてのトレースの集合を traces(X) と書く．

ここで，$JukeBox$を入出力オートマトンで定義すると．以下のようになる．

アクション集合:	{startjb(X,n)|X∈{Alice,Bob},n∈Nat}∪{playjazz,playmusic,playrock};
入力アクション集合:	in(JB) = ϕ;
出力アクション集合:	{startjb(X,n)|X∈{Alice,Bob},n∈Nat}∪{playjazz,playmusic,playrock};
内部アクション集合:	ϕ;
外部アクション集合:	{startjb(X,n)|X∈{Alice,Bob},n∈Nat}∪{playjazz,playmusic,playrock};
初期状態集合:	{(start,0)};
状態集合:	{(pc,quarter)|pc∈{start,jazz,rock,stop},quarter∈Nat};
遷移集合:	{((start,0),startjb(Alice,n),(jazz,n))|n∈Nat,n≠0} ∪{((jazz,n),playmusic,(jazz,n-1))|n∈Nat,n>1} ∪{((jazz,1),playjazz,(stop,0))} ∪{((start,0),startjb(Bob,n)(rock,n))|n∈n≠0} ∪{((jazz,n),playmusic,(jazz,n-1))|n∈Nat,n>1} ∪{((jazz,1),playjazz,(stop,0))}

5.2 匿名性の形式化

混乱の原理 (principle of confusion) に基づく匿名性の定義を説明する.

定義:

あるシステムのあるユーザ A が，ある (観測可能な) トレースを引き起し，ほかのユーザも同じトレースを引き起すことができれば，そのシステムは匿名である.

例:

A に関する任意のトレース（たとえば abc）を

$s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow s_4 \longrightarrow s_5 \longrightarrow$ （A の遷移列）

他のすべてのユーザ（たとえば B）も引き起すことができれば,

$s_0 \longrightarrow s'_1 \longrightarrow s'_2 \longrightarrow s'_3 \longrightarrow s'_4 \longrightarrow s'_5 \longrightarrow$（B の遷移列）

このシステムは匿名である.

以上のように，外部から観測されるアクション系列 (トレース) やその集合 (トレース集合) を考えることで，匿名性を論議することができる. 次では，トレース集合に関する匿名性を，形式的に定義する.

入出力オートマトン $anonym_A(X)$ の定義:

入出力オートマトン X および出力アクションの集合族 A を考える. (ただし A'A"' $\in$ A かつ A' $\neq$ A"' ならば, A' と A" は互いに素である). A' の要素を，匿名アクションと呼ぶ. 入出力オートマトン $anonym_A(X)$ を次のように定める.

$states(anonym_A(X)) = states(X)$

$start(anonym_A(X)) = start(X)$

$in(anonym_A(X)) = in(X)$

$out(anonym_A(X)) = out(X)$

$int(anonym_A(X)) = int(X)$

$$trans(anonym_A(X)) = \{(s_1, a, s_2) | (s_1, a, s_2) \in trans(X) \wedge a \notin \cup_{A' \in A} A'\} \cup \{(s_1, a, s_2) | (s_1, a', s_2) \in trans(X) \wedge A' \in A \wedge a' \in A' \wedge a \in A'\}.$$

$anonym_A(X)$ は，次の意味で匿名的なシステムである．
ある $someone \in A'$ (ただし A' ∈ A) に関して
$s \overset{someone}{\longrightarrow}_{anonym_A(X)} s'$
ならば，任意の everyone ∈ A' に関して，
$s \overset{everyone}{\longrightarrow}_{anonym_A(X)} s'$
である．

トレース匿名性の定義:

入出力オートマトンXおよび匿名アクションの集合族Aに関して，$trans(anonym_A(X)) = trans(X)$ のとき，XはAに関してトレース匿名的という．

一つ JukeBOX の例では，startjb(n，Alice) の発生と startjb(n，Bob) のそれが敵に見分けがつかないと仮定する，そして，JukeBox のアクタアクションの集合族として $A = \{\{startJB(n, Alice), startjb(n, Bob)\} \mid n \in \{1, 2, \ldots\}\}$ を使用する．

$anonym_A(JukeBox)$ の (図5.1) と $JukeBox$ の図を比較すると，$anonym_A(JukeBox)$ のトレース $startjb(n, Alice).playmusic.playrock$ が $trace(JukeBox)$ に現れないので，$trans(anonym_A(JukeBox)) \neq trans(JukeBox)$ であることがわかる．それゆえに，$JukeBox$ は匿名的なシステムではない．

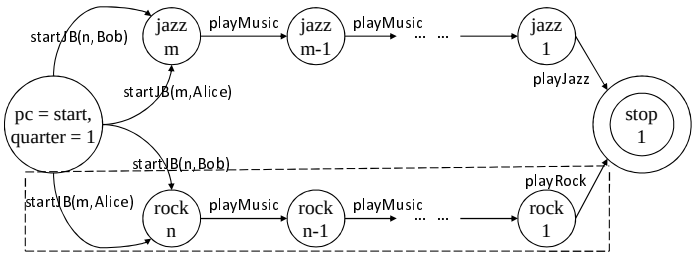

図 5.1: $anonymA(JukeBox)$

もう一つ $\overline{JukeBox}$ の例 (図5.2) では，$playJazz$ と $playRock$ は内部アクションとして隠

し，$\overline{JukeBox} = JukeBox \backslash \{playJazz, playRock\}$ とする．この操作を個人的なチャンネルで $playJazz$ と $playRock$ を送ると見なすことができる．ここで，$trans(anonymA(\overline{JukeBox}))) = trans(\overline{JukeBox})$ であるので，$JukeBox$ は匿名的なシステムである．この観測によって，動作 $playJazz$ と $playRock$ が敵から目に見えないときに，$JukeBox$ は匿名である．

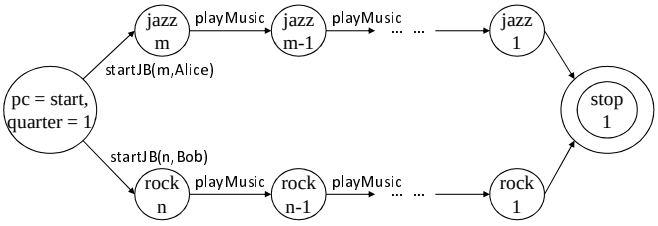

図 5.2: $\overline{JukeBox}$

第6章　観測遷移システムによる匿名プロトコルのモデル化

この章では，まず入出力オートマトンと観測遷移システムの類似点と相違点を紹介し，続いて観測遷移システムによるトレース匿名性の定義を導入する．最後前述した $JukeBox$ を，観測遷移システムでモデル化する．

6.1　入出力オートマトンと観測遷移システムの類似点と相違点

本節は，入出力オートマトンと観測遷移システムの類似点と相違点 (図 6.1) について議論する．入出力オートマトンの状態の集合は観測遷移システムの状態空間 Y と対応つけている．入出力オートマトンの状態の集合の各状態は各変数の値で特徴付けられる．これらの変数は観測遷移システムの遷移の集合と対応つけている．入出力オートマトンのアクションの集合と遷移の集合は観測遷移システムの条件付遷移規則の集合と対応つけている．

また，入出力オートマトンのアクションには外部と内部の区別がある，観測遷移システムの遷移には外部と内部の区別がない．このため，観測遷移システムの遷移にも外部と内部の区別を与え，観測遷移システムのトレースを定義する必要がある．

以下，観測遷移システム S は三つ組 $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ で定義され，各要素は以下のとおりである:

$\mathcal{O}$ **：観測の集合:**各観測 $o \in \mathcal{O}$ は，状態空間 Y から自然数等のデータ型 D への関数 o : Y → D である．観測の返す値を観測値と呼ぶ．

o および 2 つの状態 u_1, u_2 が与えられた場合，2 つの状態の s に関する等価性 ($u_1 =_s u_2$) を以下のように定義する:

$u_1 =_s u_2$ iff $\forall\ o \in \mathcal{O} \in .\ o(u_1) = o(u_2)$

ただし，$o(u_1) = o(u_2)$ における' = 'は各 $o \in \mathcal{O}$ の値域ごとに定義されているとする．

$\mathcal{I}$ **：初期状態の集合:**$\mathcal{I} \in$ Y である．

$\mathcal{T}$ **：条件付遷移規則の集合:**外部の状態遷移規則の集合 $\mathcal{T}_\mathcal{E}$ と内部の状態遷移規則の集合 $\mathcal{T}_\mathcal{I}$ からなる．

各 $t \in \mathcal{T}$ に付随する条件 $c - t$: Y → $\{true, false\}$ を効力条件と呼ぶ．s の実行は，初期状態からはじまり，次の通りに永遠的に遷移規則を適用することにより得られる状態の無

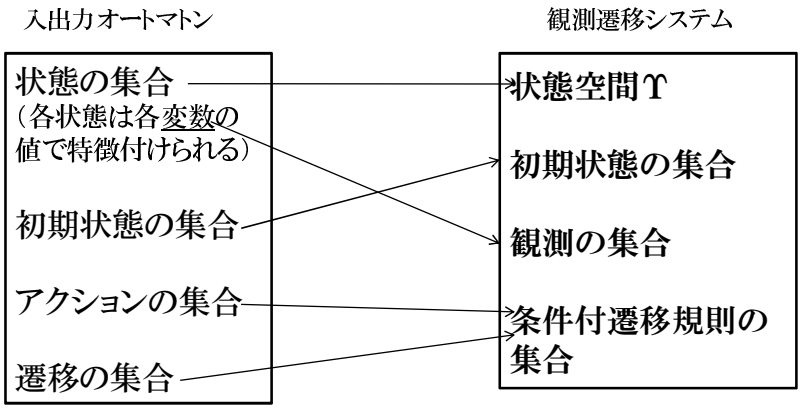

図 6.1: 入出力オートマトンと観測遷移システムの類似点と相違点

限列である; 実行の各段階で，遷移規則の一つが非決定的に選択される (事後状態が決定される). ただし，この非決定的選択は次の公平性を満たすものとする; すべての遷移規則は際限なく選択される. s が与えられると，このような状態の無限列の集合 (s の実行) が得られる. より正確には，s の実行は，以下の条件を満たす状態の無限列 $u_1, u_2 \ldots$ である.

開始性: 状態 u_0 において，各 $o \in O$ は $\mathcal{I}$ を満たす.

一貫性: すべての $i \in 0, 1, \ldots$ において，$u_{i+1} =_s t(u_i)$ となる遷移規則 $t \in \mathcal{T}$ が存在する.

公平性:各 $\tau \in \mathcal{T}$ に対し，$u_{i+1} =_s t(u_i)$ となる $i \in 0, 1, \ldots$ が無限に存在する.

OTS と二つの状態 u，u' を与えて，$t_{y_1,\ldots,y_n}(u) =_s u'$ を満たす遷移 $t \in \mathcal{T}$ と $k \in D_K (k = x_1, \ldots, x_m, y_1, \ldots, y_n)$ であるデータ型 D_k が存在すれば，$\tau_{y_1,\ldots,y_n}(u) =_s u'$ を $u \longrightarrow_s^\tau u'$ と書く. $\tau \in \mathcal{T}_I$ のとき,$u \longrightarrow u'$ とも書く. それが前後関係から明白であるならば，下に書かれた s は省略する. 関係 $\rightsquigarrow$ を，関係 $\longrightarrow$ の反射の推移の閉包とする. $\tau \in \mathcal{T}_E$ のときは，s から s' の方へ動作している遷移は，$u \longrightarrow_s^\tau u'$ と書く.　また，外部遷移規則が存在しないときは，$u \rightsquigarrow_s u'$ と書く.

定義: ある観測遷移システム $S = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ を与えて，到達可能の状態は次のように定義される. (1) 任意の $u_0 \in \mathcal{I}$ は到達可能の状態である. (2) 任意の $u, u' \in$ Υ において，$u \longrightarrow_s^\tau u'$ を満たす $\tau \in \mathcal{T}$ が存在する，もし u が到達可能状態であるならば，u' も到達可能状態である. S のすべての到達可能状態の集合を，Rs によって定義されます.

定義: ある観測遷移システム $S = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ を与えて，遷移列は次のように定義される．(1) 任意の $u_0 \in \mathcal{I}$ は遷移列である．(2) 任意の遷移列 a に関しては，遷移 $last(a) \longrightarrow_s^\tau u'$ があれば，$a \longrightarrow_s^\tau u'$ は遷移列である．遷移列での関数 last は以下のように定義される．$(1) last(u_0) = u_0$ $(2) last(a \longrightarrow_s^\tau u) = u_1$．ただし a は遷移列である．

トレースの定義: ある観測遷移システム $S = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ の遷移列 $u_0 \longrightarrow_s^{\tau_0} u_1 \longrightarrow_s^{\tau_1} \ldots u_{n-1} \longrightarrow_s^{\tau_{n-1}} u_n \ldots$ が与えて，この遷移列に現われるすべての外部遷移を順に並べたもの $(\tau_0 \tau_1 \ldots \tau_{n-1} \ldots)$ をこの遷移列のトレースと呼ぶ．観測遷移システム S のトレースの集合を trace(S) と書く．

観測遷移システム $anonym_A(S) = \langle \mathcal{O}_{anonym_A(S)}, \mathcal{I}_{anonym_A(S)}, \mathcal{T}_{anonym_A(S)} \rangle$ の定義:
観測遷移システム $S = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ および出力アクションの集合族 A を考える．(ただし $A', A, \ldots \in A$ かつ $A' \neq A, \ldots$ ならば，A' と A'' は互いに素である)．A' の要素を，匿名アクションと呼ぶ．観測遷移システム $OTS_{S_A} =< \mathcal{O}_{\mathcal{S}_\mathcal{A}}, \mathcal{I}_{\mathcal{S}_\mathcal{A}}, \mathcal{T}_{\mathcal{S}_\mathcal{A}} >$ を次のように定める．
$\mathcal{O}_{anonym_A(S)} = \mathcal{O}_\mathcal{S}$
$\mathcal{I}_{anonym_A(S)} = \mathcal{I}_\mathcal{S}$
$\mathcal{T}_{anonym_A(S)} = \{u \longrightarrow_{s_A}^\tau u' | u \longrightarrow_{s_A}^\tau u' \in \mathcal{T}_S \wedge T \notin \cup_{A' \in A} A'\} \cup \{u \longrightarrow_{s_A}^\tau u' | u \longrightarrow_{s_A}^{\tau'} u' \in \mathcal{T}_S \wedge A' \in A \wedge \tau' \in A' \wedge \tau \in A'\}$

トレース匿名性の定義:
観測遷移システム $S = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ および匿名アクションの集合族 A(ただし $A', A, \ldots \in A$ かつ $A' \neq A, \ldots$ ならば，A' と A'' は互いに素である) に関して，$trans(anonym_A(S)) = trans(S)$ のとき，$anonym_A(S) = \langle \mathcal{O}_{anonym_A(S)}, \mathcal{I}_{anonym_A(S)}, \mathcal{T}_{anonym_A(S)} \rangle$ は A に関してトレース匿名的という．

6.2 JukeBoxのモデル化

ここでは，前述した $JukeBox$ を，観測遷移システムでモデル化する．最初に, データとして

label = {start,jazz,rock,stop}	JukeBox の識別子
Nat = {0,1,2,...}	コインの個数を表す値
Pid = {ALICE, BOB}	人の識別子

これらのデータを用いて，$JukeBox$ を表す観測遷移システム $\langle \mathcal{O}_{JukeBox}, \mathcal{I}_{JukeBox}, \mathcal{T}_{JukeBox} \rangle$ を定義する．JukeBox を表す状態空間の存在を仮定する．

観測 $\mathcal{O}$ は次のように表現できる．

$$\mathcal{O}_{JukeBox} \triangleq \{pc : \Upsilon \to Label, quarter : \Upsilon \to Nat\}$$

$JukeBox$ の現在状態を $JukeBox$ 状態の識別子 $label$ で観測する．コインの個数をコインの個数を表す値で観測する．初期状態 $\mathcal{I}$ は，任意の状態 u $\in$ Y に対し，次のように表現できる．

$$\mathcal{I}_{JukeBox} \triangleq \{\upsilon_{init} \in \Upsilon \mid pc(\upsilon_{init}) = start \wedge quarter(\upsilon_{init}) = 0\}$$

観測規則 $\mathcal{T}$ は，3 つの添字付き遷移規則の集合で表現する．ここでは，観測規則が適用されたときの状態 $u \in$ Y，遷移規則が適用された後の状態を $u' \in$ Y と表記する．

$\mathcal{T}_{JukeBox} \triangleq \mathcal{T}_{E_{jukeBox}} \cup \mathcal{T}_{I_{jukeBox}}$ where,

$\mathcal{T}_{E_{JukeBox}} \triangleq \{startJB_{n:Nat,p:Pid} : \Upsilon \to \Upsilon,$
$playMusic : \Upsilon \to \Upsilon\}$

$\mathcal{T}_{I_{JukeBox}} \triangleq \{playJazz : \Upsilon \to \Upsilon, playRock : \Upsilon \to \Upsilon\}$

$\mathcal{T}_{E_{JukeBox}} \triangleq \{startJB_{n:Nat,p:Pid} : \Upsilon \to \Upsilon\}$

- $startJB_{n:Nat,p:Pid} : \Upsilon \to \Upsilon$

効力条件は，pc(ν) = start and quarter(ν) = 0 である．これは，状態 ν においては，$JukeBox$ は pc が start で quarter が 0 であることを意味している．効力条件が真であるとき，各遷移関数は効果で定義した値に変化する．ここでは，遷移規則が適用された後の観測関数は pc(ν ') = (if X = ALICE then jazz else rock), quarter(ν ') = X である．効力条件が真でないとき，各遷移関数は変化しない．

遷移関数 startjb は，以下のように定義される．

```
c- startjb(U,X, ν )  = (pc( ν ) = start and quarter( ν ) = 0)
pc(startjb(U,X, ν ) = (if U = ALICE then jazz else rock)
                                  if c-startjb(U,X, ν ) .
quarter(startjb(U,X, ν )) = X   if  c-startjb(U,X, ν ) .
startjb(U,X, ν ) =  ν            if ¬  c-startjb(U,X, ν ) .
```

- $playMusic : \Upsilon \rightarrow \Upsilon$

効力条件は (pc(ν) = jazz or pc(ν) = rock) and quarter(ν) > succ(0) である. これは，$JukeBox$ が状態 jazz または rock に存在するとき，コインの数は 1 より大きなことを意味している. 効力条件が真であるとき，各遷移関数は効果で定義した値に変化する. ここでは，遷移規則が適用された後の観測関数は pc(ν ') = (if pc(ν) = jazz then jazz else rock)，quarter(ν ') = quarter (ν) - succ(0) である. 効力条件が真でないとき，各遷移関数は変化しない.

遷移関数 playmusic は，以下のように定義される.

```
c- playmusic( ν ) = (pc( ν ) = jazz or pc( ν ) = rock)
                          and  quarter( ν ) > succ(0)
pc(playmusic( ν ))=(if pc( ν ) = jazz then jazz else rock)
                                  if  c-playmusic ( ν ) .
quarter(playmusic ( ν )) = quarter ( ν ) - succ(0)
                                  if  c-playmusic( ν )  .
playmusic ( ν ) = ν   if ¬ c-playmusic( ν )
```

- $playJazz : \Upsilon \rightarrow \Upsilon$

効力条件は pc(ν) = jazz and quarter(ν) = succ(0) である. これは，jazz は最後の曲として，$JukeBox$ を閉じることを表す遷移規則の集合である. 効力条件が真であるとき，各遷移関数は効果で定義した値に変化する. ここでは，遷移規則が適用された後の観測関数は pc(ν ') = stop，quarter(ν ') = n である. 効力条件が真でないとき，各遷移関数は変化しない.

遷移関数 playjazz は，以下のように定義される.

```
c- playjazz( ν ) = pc( ν ) = jazz  and  quarter( ν ) = succ(0)
pc(playjazz ( ν )) = stop          if  c-playjazz   ( ν ) .
quarter(playjazz( ν )) = n         if  c-playjazz   ( ν ) .
playjazz( ν ) = ν   if ¬ c-playjazz( ν )
```

- $playrock : \Upsilon \rightarrow \Upsilon$

効力条件は pc(ν) = rock and quarter(ν) = succ(0) である. これは，rock は最後の曲として，$JukeBOX$ を閉じることを表す遷移規則の集合である. 効力条件が真であるとき，各遷移関数は効果で定義した値に変化する. ここでは，遷移規則が

適用された後の観測関数は pc(ν ') = stop ,quarter(ν ') = n である. 効力条件が真でないとき，各遷移関数は変化しない.

遷移関数 playrock は，以下のように定義される.

```
c- playrock( ν ) = pc( ν ) = rock  and  quarter( ν ) = succ(0)
pc(playrock ( ν )) = stop          if  c-playrock   ( ν ) .
quarter(playrock( ν )) = n         if  c-playrock   ( ν ) .
playrock( ν ) = ν   if ¬ c-playrock( ν )
```

以上のように,JukeBox をモデル化した.

6.3 CafeOBJ による仕様記述

観測遷移システムでモデルした *JukeBox* を CafeOBJ で記述した. 自然数，状態の識別子および使用者の識別子を記述するモジェール MYNAT，LABEL，USER と，一方 *JukeBOX* システム本体を記述するモジェール *JukeBox* の 4 つのモジェールからなる. 仕様は，全体で 119 行であり，完全なコードは，付録 A に示す.

6.3.1 抽象データ型の記述

モジェール MYNAT は自然数を定義するモジェールである. モジェール LABEL は，状態の識別子を定義するモジェールである. モジェール USER は使用者の識別子を定義するモジェールである.

6.3.2 抽象機械の記述

一方 *JukeBOX* を表す抽象機械は，モジェール *JukeBox* に記述した. 観測は，次のように記述した.

```
bop pc :  State -> Label
bop quarter :  State -> Nat
```

初期条件は，次のように記述した.

```
op  init : -> State
eq pc(init) = start .
eq quarter(init) = 0 .
```

遷移規則と，効力条件と効果を記述するための演算は，次のように記述する.

```
bop startjb :   User   Nat  State -> State
bop playmusic :   State -> State
bop playjazz :    State -> State
bop playrock :    State -> State
```

startjb の効力条件は，次のように記述した.

```
eq c-startjb(U, X, S) = pc(S) = start and quarter(S) = 0 .
```

startjb の効果は，次のように記述する.

```
ceq pc(startjb(U, X, S)) = (if U = ALICE then jazz  else rock  fi)
                                        if c-startjb(ALICE, X, S) .
ceq quarter(startjb(U, X, S)) =  X      if c-startjb(U, X, S) .
ceq startjb(U, X, S) = S                if not c-startjb(U, X, S) .
```

playmusic の効力条件は，次のように記述した.

```
eq c-playmusic(S) = (pc(S) = jazz or pc(S) = rock)
                                        and quarter(S) > succ(0) .
```

playmusic の効果は，次のように記述する.

```
ceq    pc(playmusic(S)) = (if pc(S) = jazz then jazz else rock fi)

                                        if c-playmusic(S) .
ceq quarter(playmusic(S)) = quarter(S) - succ(0)
                                        if c-playmusic(S) .
ceq playmusic(S) = S                    if not c-playmusic(S) .
```

playjazz の効力条件は，次のように記述した.

```
eq c-playjazz(S) = pc(S) = jazz and quarter(S) = succ(0) .
```

playjazz の効果は，次のように記述する.

```
ceq      pc(playjazz(S)) = stop         if c-playjazz(S) .
ceq quarter(playjazz(S)) = 0            if c-playjazz(S) .
ceq playjazz(S) = S                     if not c-playjazz(S) .
```

playrock の効力条件は，次のように記述した.

```
eq c-playrock(S) = pc(S) = rock and quarter(S) = succ(0) .
```
playrock の効果は，次のように記述する．

```
ceq pc(playrock(S)) = stop              if c-playrock(S) .
ceq quarter(playrock(S)) = 0            if c-playrock(S) .
ceq playrock(S) = S                     if not c-playrock(S) .
```

第7章　検証

7.1　匿名性の検証方法

本章は，観測遷移システム OTS のトレース匿名性を示すための手法を導入する．この手法では，匿名シミュレーションと呼ばれる，状態集合上の 2 項関係を用いる．

定義

観測遷移システム OTS S および出力アクションの集合族 A(ただし A'A"' $\in$ A かつ A' $\neq$ A"' ならば，A' と A" は互いに素である) を考える．以下の条件を満たすなら，$as_A : R_s R_s \rightarrow Bool$ は A における S の匿名のシミュレーション関係である．

1. 任意の $u_0 \in I$ に関して，$as_A(u_0, u_0)$ が成り立つ．

2. 任意の $u_1, u_2, u_1' \in R_s$ に関して，$as_A(u_1, u_1')$ かつ $u_1 \xrightarrow{t}_s u_2$ ならば，ある A' $\in$ A に関して $t \in$A' ならば, すべての $t' \in$A' に関して，ある $u_2' \in R_s$ が存在して，$as_A(u_2, u_2')$ かつ $u_1 \xrightarrow{t'}_s u_2$ である; すべての A' $\in$ A に関して $t \in$A' ならば:(a) $t \in T_E$ ならば，ある $u_2' \in R_s$ が存在して，$as_A(u_2, u_2')$ かつ $u_1 \xrightarrow{t}_s u_2$ である．(b)$t \in T_I$ ならば，ある $T \in T_I$ に関して，ある $u_2' \in R_s$ が存在して，$as_A(u_2, u_2')$ かつ $u_1 \xrightarrow{t}_s u_2$ である．条件 2 は (図 7.1) で表す．

定理

観測遷移システム OTS S および出力アクションの集合族 A(ただし A'A"' $\in$ A かつ A' $\neq$ A"' ならば, A' と A" は互いに素である) を考える．$S_{anonym_A(S)}$ から S のフォワードシミュレーション as_A が存在すれば，S は A に関してトレース匿名的である．

証明: $S_{anonym_A(S)}$ の定義から，$traces(anonym_A(S) \supseteq traces(S)$ は容易，関係 $as_A : R_{anonym_A(S)} R_s \rightarrow Bool$ が $anonym_A(S)$ から S へのフォワードシミ ュレーションであることを証明する．ここで，$O_{anonym_A(S)} = O_S$ および $I_{anonym_A(S)} = I_S$ である．

フォワードシミ ュレーション関係の定義の条件 1 より，$as_A(u_0, u_0)$ を満たす任意の初期状態 $u_0 \in I_{anonym_A(S)} (= I_S)$ が存在する．したがって，フォワードシミュレーションのための初期状態の条件は満たされる．

次に，$as_A(u_1, u_1')$ および $u_1 \xrightarrow{t'}_s u_2$ を満たす任意の $u_1, u_2 \in O_{anonym_A(S)}, u_1' \in O_S$ および t を考える．$O_{anonym_A(S)} = O_S$ より, $u_1, u_2 \in O_S$ である．ここで，ある A' $\in$ A に関して $t \in$A' か否かで分ける．

ある A'$\in$A に関して $t \in$A' のときは，$anonym_A(S)$ の定義から，フォワードシミ ュレーション関係の定義条件 2 より，ある $u_2' \in O_s$ が存在して，$as_A(u_1, u_1')$ かつ $u_1 \xrightarrow{t}_s u_2$ である．したがって，$t \in$A' の場合，フォワードシミュレーションのためのステップが成り

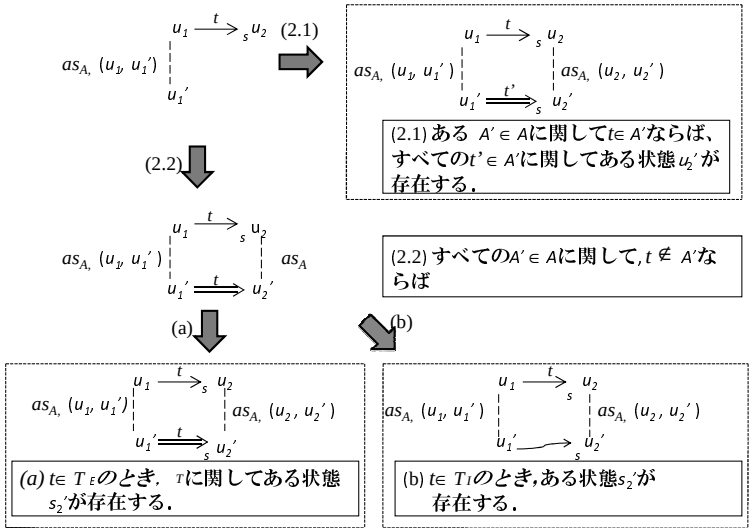

図 7.1: 匿名のシミュレーション関係

立つ.

すべての A' ∈ A に関して $t \notin$A' ならば，$t \in T_E$ のときは，フォワードシミ ュレーション関係の定義条件 2 より，ある $u_2' \in O_s$ が存在して，$as_A(u_2, u_2')$ かつ $u_1 \xrightarrow{t}_s u_2$ である. $t \in T_I$ のときは，フォワードシミ ュレーション関係の定義条件 2 より，ある $u_2' \in O_s$ が存在して，$as_A(u_2, u_2')$ かつ $u_1 \xrightarrow{t}_s u_2$ である.

7.2 匿名性の検証

本研究では,CafeOBJ を用いて，帰納法を用いた匿名性の検証を行う. 場合分けで各証明節について，証明譜と呼ばれるコードをそれぞれ記述する. これらの証明譜を CafeOBJ の処理系を用いて簡約することで，各証明節が表している推論を行うことができる. これらの各簡約結果が，すべて希望の結果となるならば, その検証は成功する.

まず CafeOBJ でトレース匿名性を証明するために，最初に，匿名のシミュレーションの候補関係 r を定義する. JekeBOX は外部アクション: startjbALICE，， startjbBOB，， playmusic，内部アクション: playjazz，playrock とする場合は，匿名シミュレーションは次のように定義する.

asA(s, s') $\Leftrightarrow$(quarter(s) = quarter(s') $\wedge$ ((pc(s) = pc(s')) $\vee$ (pc(s) = jazz $\wedge$ pc(s')=rock) $\vee$ (pc(s)=rock $\wedge$ pc(s')=jazz)))

それから，r が匿名シミュレーション関係の条件を満たすことを示す. まず，定義の条件 1 を満たすことを示す. 任意の s $\in$ start(JukeBOX) に関して，asA(s2，s'2) が成り立

つ. 次に，定義の条件 2 を満たすことを示す.
まず以下のモジェールを宣言する.

```
mod ASIM {
 pr(JUKEBOX)
 ops s1 s1' s2 s2' : ->  State
 vars M N : State
 op asA : State State  -> Bool
 eq asA(M,N) = (quarter(M) = quarter(N) and
               (pc(M) = pc(N) or (pc(M) = jazz and pc(N) = rock)
                              or (pc(M) = rock and pc(N) = jazz)))  .
}
```

演算 asA は匿名性シミュレーション関係に対応する. シミュレーション関係の証明は, シミュレーションが初期状態を表す定数および遷移演算により帰納的に定義できるという事実から得られる帰納スキーマに基づいて行う. その帰納スキーマは,

```
{[1-init],
 [1-startjb_[ALICE,n]]*,[1-startjb_[BOB,n]]*,[1-playmusic]*,
 [1-playjazz]*,[1-playrock]*}
 implies[asA]*
```

のように記述できる. 前提は，6 つの言明 (1 つの基底段階と 5 つの帰納段階) から構成される. 6 つの言明のそれぞれの証明譜を作成確認することで，匿名性シミュレーション関係の証明を行う. もし，証明節がすべて成り立ち，上の帰納スキーマから，言明 [asA]*が成り立つことが証明される.　各帰納段階で証明すべき論理式を記述したモジェールを以下のように宣言する.

```
mod STEP {
 pr(ASIM)
 op stepA : -> Bool
 eq stepA = asA(s1,s1') implies asA(s2,s2') .
}
```

定数 s1，s1' は任意の状態を表す，定数 s2，s2' は状態 s1,s1' の事後状態を表す. 項 asA(s2, s2') は各帰納段階で証明すべき論理式である. 項 asA(s1，s1') は帰納法の仮定としてよく使うため，演算 stepA を上記のように定義する. 完全なコードは，付録 B に示す.

7.2.1 証明譜

基底段階の証明節

基底段階 (init) の証明節は以下のとおりである.

```
open ASIM .
 red asA(init,init) .
close .
```

遷移演算 startjb __ [ALICE，n] に関する帰納段階

遷移演算 startjb __ [ALICE，n] に関する帰納段階 (図 7.2) について考える．まず a = startjb __ [ALICE，n] のとき，startjb __ [ALICE，n] は外部アクションであるので，シミュレーション関係の定義より，a' = startjb __ [ALICE，n] と a" = startjb __ [BOB，n] の二つの場合に分けて証明する必要がある.

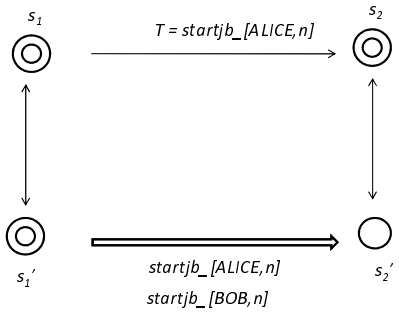

図 7.2: 遷移演算 startjb __ [ALICE，n]

ここで，a' = startjb[ALICE，n] のときを考えると，まず，効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は，前者に対し，true でも false でもない Bool の項を返し，後者に対し，true を返す.

前者の証明節を，効力条件 c-startjb に基づく場合わけにより，以下の Bool の 2 つの項 (bp1，bp2) に基づいて，前者の証明節を以下のとおり 3 つ (¬bp1，bp1∧bp2，bp1∧¬bp2) に分割する.

bp1 $\triangleq$ `pc(s1') = start`,
bp2 $\triangleq$ `quarter(s1') = 0`.

3 つの証明節

- ¬bp1

```
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq (pc(s1') = start) = false .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .
```

- bp1∧bp2

```
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq quarter(s1') = 0 .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .
```

- bp1∧¬bp2

```
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq (quarter(s1') = 0) = false .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .
```

帰納段階における各証明節は，4 つの部分から構成されている：(1) 任意の値を表す定数宣言，(2) 仮定を表す等式宣言，(3) 事後状態の定義，(4) 証明すべき論理式が (2) の仮定のもとで成り立つことの確認. 最初の証明節 ¬bp1 では，(1) は op n : − > Nat . である. (2) は asA(s1,s1') である. (3) は eq s2' = startjb(ALICE,n,s1') . である. (4) は証明すべき論理式 asA(s2,s2') が asA(s1,s1') の仮定のもとで成り立つことを確認する. ほかの証明節も証明節 ¬bp1 と同じ方法で作成されている.
a" = startjb __ [BOB,n] のときを考えると，a' = startjb __ [ALICE,n] のときと同じく効力条件 c-startjb に基づく場合わけにより，次の 3 つの証明節 (¬bp1,bp1∧bp2，bp1∧¬bp2) を得ることである.

3 つの証明節

- ¬bp1

```
 open STEP .
-- arbitrary values
op n : -> Nat .
-- assumptions
-- eq c-startjb(ALICE,n,s1) = true .
eq pc(s1) = start .
```

```
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq (pc(s1') = start) = false .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .
```

- bp1∧bp2

```
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq quarter(s1') = 0 .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .
```

- bp1∧¬bp2

```
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
```

```
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq (quarter(s1') = 0) = false .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .
```

遷移演算 startjb __ [BOB，n] に関する帰納段階

遷移演算 startjb __ [BOB，n] に関する帰納段階 (図 7.3) について考える. startjb[BOB, n] は外部アクションであるので，シミュレーション関係の定義より，a' = startjb __ [BOB, n] と a" = startjb __ [ALICE，n] の二つの場合に分けて証明する必要がある. ここで，a' = startjb[BOB，n] のときを考えると，まず，効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する. CafeOBJ は，前者に対し，true でも false でもない Bool の項を返し，後者に対し，true を返す.

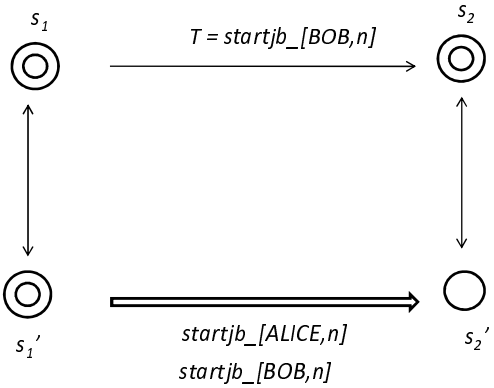

図 7.3: 遷移演算 startjb __ [BOB,n]

前者の証明節を，効力条件 c-startjb に基づく場合わけにより，以下の Bool の 2 つの項

(bp1,bp2) に基づいて，前者の証明節を以下のとおり 3 つ (¬bp1,bp1∧bp2,bp1∧¬bp2) に分割する.
bp1 ≜ `pc(s1') = start`,
bp2 ≜ `quarter(s1') = 0`.

3 つの証明節：

- ¬bp

```
 open STEP .
-- arbitrary values
op n : -> Nat .
-- assumptions
-- eq c-startjb(BOB,n,s1) = true .
eq pc(s1) = start .
eq quarter(s1) = 0 .
--
eq s2 = startjb(BOB,n,s1) .
-- case anaysis
eq (pc(s1') = start) = false .
-- check if there exists s2'
eq s2' = startjb(BOB,n,s1') .
red stepA  .
close .
```

- bp1∧bp2

```
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq quarter(s1') = 0 .
 -- check if there exists s2'
```

```
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .
```

- bp1∧¬bp2

```
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq (quarter(s1') = 0) = false .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .
```

a" = startjb __ [ALICE, n] のときを考えると，a' = startjb __ [BOB, n] のときと同じく効力条件 c-startjb に基づく場合わけにより，次の 3 つの証明節 (¬bp1，bp1∧bp2, bp1∧¬bp2) を得ることである.

3 つの証明節:

- ¬bp

```
open STEP .
-- arbitrary values
op n : -> Nat .
-- assumptions
-- eq c-startjb(BOB,n,s1) = true .
eq pc(s1) = start .
eq quarter(s1) = 0 .
--
eq s2 = startjb(BOB,n,s1) .
-- case anaysis
```

```
 eq (pc(s1') = start) = false .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .
```

- bp1∧bp2

```
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq quarter(s1') = 0 .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .
```

- bp1∧¬bp2

```
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq (quarter(s1') = 0) = false .
```

```
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .
```

遷移演算 playmusic に関する帰納段階

遷移演算 playmusic に関する帰納段階 (図 7.4) について考える. a = playmusic のとき, playmusic は外部アクションであり, しかも出力アクションの集合族 A に含まれてないので, シミュレーション関係の定義より, s1' から外部アクション playmusic により s2' を生成する. まず, 効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する. CafeOBJ は, 前者に対し, true でも false でもない Bool の項を返し, 後者に対し, true を返す.

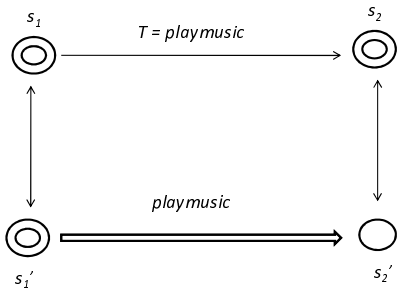

図 7.4: 遷移演算 playmusic

前者の証明節を, 効力条件 c-playmusic に基づく場合わけにより, 以下の Bool の 3 つの項に基づいて, 前者の証明節を以下のとおり 5 つ (¬bp1∧bp2,¬bp1∧bp2∧¬bp3, ¬bp1∧bp2∧bp3, bp1∧¬bp3, bp1∧bp3) に分割する.
bp1 ≜ `pc(s1') = jazz`,
bp2 ≜ `pc(s1') = rock`,
bp3 ≜ `quarter(s1') = quarter(s1)`.

5 つの証明節:

- ¬bp1∧¬bp2

```
    open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
  eq (pc(s1') = jazz) = false .
  eq (pc(s1') = rock) = false .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .
```

- ¬bp1∧bp2∧¬bp3

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false .
 eq pc(s1') = rock .
 eq (quarter(s1') = quarter(s1)) = false .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .
```

- ¬bp1∧bp2∧bp3

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false .
 eq pc(s1') = rock .
 eq quarter(s1') = quarter(s1) .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .
```

- bp1∧¬bp3

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
 eq pc(s1') = jazz .
 eq (quarter(s1') = quarter(s1)) = false .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .
```

- bp1∧bp3

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
 eq pc(s1') = jazz .
 eq quarter(s1') = quarter(s1) .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .
```

遷移関数 playjazz に関する帰納段階

遷移演算 playjazz に関する帰納段階 (図 7.5) について考える． a = playjazz のとき，playjazz は内部アクションであるので，シミュレーション関係の定義より，s1' から任意の内部アクション a により s2' を生成する．

ここで，a = playjazz のとき考えると，まず，効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は前者に対し，true でも false でもない Bool の項を返し，後者に対し，true を返す．

前者の証明節を，効力条件 c-playjazz に基づく場合わけにより，以下の Bool の 3 つの項に基づいて，前者の証明節を以下のとおり 5 つ (bp1∧bp2，bp1∧¬bp2，¬bp1∧¬bp2，¬bp1∧bp2∧
¬bp3，¬bp1∧bp2∧bp3) に分割する．
bp1 ≜ `pc(s1') = jazz`,
bp2 ≜ `quarter(s1') = 1`,
bp3 ≜ `pc(s1') = rock`.

5 つの証明節:

- bp1∧bp2

```
open STEP .
 -- arbitrary values
 -- assumptions
```

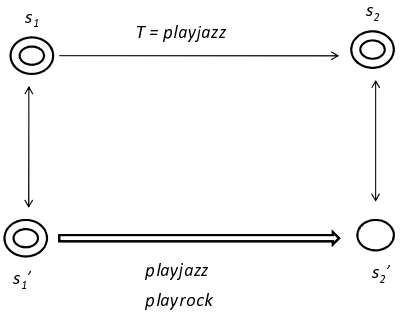

図 7.5: 遷移演算 playjazz

```
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq pc(s1') = jazz .
 eq quarter(s1') = succ(0) .
 -- check if there exists s2'
 eq s2' = playjazz(s1') .
 red stepA  .
close .
```

- bp1∧¬bp2

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
```

```
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq pc(s1') = jazz .
 eq (quarter(s1') = succ(0)) = false .
 -- check if there exists s2'
 eq s2' = playjazz(s1') .
 red stepA  .
close .
```

- ¬bp1∧¬bp2

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false .
 eq (quarter(s1') = succ(0)) = false .
 -- check if there exists s2'
 eq s2' = playjazz(s1') .
 red stepA  .
close .
```

- ¬bp1∧bp2∧¬bp3

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false .
```

```
 eq quarter(s1') = succ(0) .
 eq (pc(s1') = rock) = false .
 -- check if there exists s2'
 eq s2' = playjazz(s1') .
 red stepA  .
close .
```

- ¬bp1∧bp2∧bp3

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false .
 eq quarter(s1') = succ(0) .
 eq pc(s1') = rock .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .
```

遷移関数 playrock に関する帰納段階

遷移演算 playrock に関する帰納段階 (図 7.6) について考える．a = playrock のとき，playrock は内部アクションであるので，シミュレーション関係の定義より，s1' から任意の内部アクションにより s2' を生成する．

ここで，a = playrock のときを考えると，まず，効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は，前者に対し，true でも false でもない Bool の項を返し，後者に対し，true を返す．

前者の証明節を，効力条件 c-playrock に基づく場合わけにより，以下の Bool の 3 つの項に基づいて，前者の証明節を以下のとおり 5 つ (bp1∧bp2，bp1∧¬bp2，¬bp1∧¬bp2，¬bp1∧bp2∧
¬bp3，¬bp1∧bp2∧bp3) に分割する．

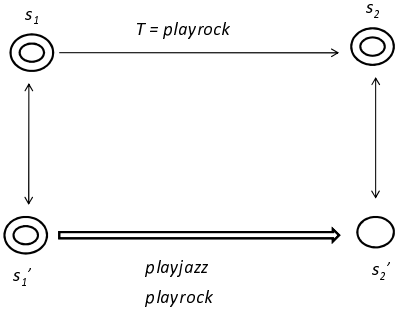

図 7.6: 遷移演算 playrock

bp1 ≜ `pc(s1') = rock`,
bp2 ≜ `quarter(s1') = 1`,
bp3 ≜ `pc(s1') = jazz`.

5 つの証明節:

- bp1∧bp2

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playrock(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
 eq pc(s1') = rock .
 eq quarter(s1') = succ(0) .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .
```

- bp1∧¬bp2

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
 eq pc(s1') = rock .
 eq (quarter(s1') = succ(0)) = false .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .
```

- ¬bp1∧¬bp2

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playrock(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
 eq (pc(s1') = rock) = false .
 eq (quarter(s1') = succ(0)) = false .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .
```

- ¬bp1∧bp2∧¬bp3

```
open STEP .
```

```
 -- arbitrary values
 -- assumptions
 -- eq c-playrock(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
 eq (pc(s1') = rock) = false .
 eq quarter(s1') = succ(0) .
 eq (pc(s1') = jazz) = false .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .
```

- ¬bp1∧bp2∧bp3

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playrock(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
 eq (pc(s1') = rock) = false .
 eq quarter(s1') = succ(0) .
 eq pc(s1') = jazz .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .
```

第8章　まとめと今後の課題

本研究では，NTT の河辺らの方法を参考に，プロトコルを観測遷移システムでモデル化し，代数仕様言語 CafeOBJ で仕様記述し，CafeOBJ 処理系を定理証明器として用いた匿名性検証を行う方法を提案した．不変性検証のために開発された観測遷移システムと CafeOBJ を用いた知見や技術を匿名性検証にも適用できることを確認した．

本研究では，トレース匿名性の証明手法として，フォワードシミュレーションを導入した．また，フォワードシミュレーションの存在が，トレース匿名の十分条件であることを示した．さらに，トレース匿名性の検証例として，$JukeBox$ に対する匿名性の検証を行った．この検証では，OTS に基づく仕様記述言語で $JukeBox$ を記述し，定理証明器を用いて匿名性を証明した．記述した仕様と定理証明について考察を行い，今後の課題について述べる．

8.1　OTS/CafeOBJ 法

OTS/Cafeobj 法では，ユニークな特徴 (システムやプロトコルの外部から観測可能な値の変化に着目したモデル化や可読性等に優れた証明譜による検証法など) を持つ．システムのモデルを観測遷移システム (OTS) として作成し，OTS を代数仕様言語 CafeOBJ で記述する．匿名性プロトコルの OTS/CafeOBJ 法による仕様記述では，直感的に理解しやすいものとなっており，その可読性も高いものとなった．記述した仕様は書き換え規則により自動的に状態遷移が行われる．その状態遷移の変化はシーケンスチャートに自然に対応させることができるため，遷移の様子を把握することは容易である．さらに，OTS/CafeOBJ 法による仕様では，ある目的の状態を規則の中に盛り込むことで，その状態を取り出すことができるため，匿名性の成立を検証することが可能である．

8.2　定理証明

$JukeBox$ の例では，可変な $quarter$ がすべての自然数に渡る時から，$JukeBox$ は無限の状態を持っている．それゆえに，モデル検査法によって直接 $JukeBox$ の例に対処することができない．この研究において，定理証明法を利用した．定理を証明することによって，帰納的に無限の状態を取り扱うことができる．帰納法による検証的手法による証明では，仕様を修正しなければならない理由をシステムから得ることが困難な場合があるなど

問題点を持つ．人間に直感と形式的な証明の間には多きな差がある．たとえば任意な自然数に対して，加法が対称律を満たすことは直感的に理解できるが，これを証明論的手法で公理から証明するためには二つの補題を用いて帰納的に証明しなければならない．

本研究では，シミュレーション関係の証明より，観測遷移システム S の匿名アクション集合族 A に関するトレース匿名性を示すことは，$anonym_A(S)$ から S へのシミュレーションの関係存在を証明することによって行える．しかし，この方法では，擬決定的とは限らない観測遷移システム $anonym_A(S)$ を生成しなければならない．観測遷移システムが擬決定的とは，任意の状態 s および遷移 t に関して，S の t に関する次状態高々一つしか存在しないことをいう．本研究の検証例でも，擬決定的な観測遷移システムとして $JukeBOX$ を記述した．擬決定的な観測遷移システムでは，等式を使って形式化できることが知られている．そのため，CafeOBJ などの等式推論の機能を持つ定理証明器で効率的に匿名性検証できると考える．

8.3 今後の課題

本研究で用いた $JukeBox$ の匿名性は，$Alice$ および Bob は匿名でコインを $JukeBox$ に入れて，$JukeBox$ を利用したことをだれにも知られないことである．そして，JukeBOX の匿名遷移の集合族として $A = \{\{startjb(n, Alice), startjb(n, Bob)\} \mid n \in \{1, 2, \ldots\}\}$ を使用し，$JekeBox$ の外部アクションは $startjb\{m, Alice\}, startjb\{n, Bob\}$ および $playmusic$ とする検証を行った．この検証例から見ると，どの外部遷移を匿名遷移に分類するかは検証したい具体的な匿名性に依存している．提案方法を規模の大きなプロトコルへ適用するために，系統的に分類する方法を考案する必要がある．

今後は，電子投票プロトコルやネットワーク上の匿名性プロトコルなどもっと複雑なプロトコルに対する匿名性解析などにも，本稿の手法を適用したい．

付 録A　JukeBOXの仕様

```
mod! MYNAT {
  [ Nat ]
  op   0  : -> Nat
  op succ : Nat -> Nat
  op _=_ : Nat Nat -> Bool { comm }
  op _+_ : Nat Nat -> Nat
  op _-_ : Nat Nat -> Nat
  op _<_ : Nat Nat -> Bool
  op _>_ : Nat Nat -> Bool
  op _<=_ : Nat Nat -> Bool
  op _>=_ : Nat Nat -> Bool
  vars X X1  X' X1' : Nat

  eq 0 + X =  X .
  eq succ(X) +  X1 = succ(X +  X1) .

  eq X - 0 =  X .
  eq 0 - X =  0 .
  eq succ(X) - succ(X1) = X - X1  .

  eq (X = X)  = true .
  eq (0 = succ(X)) = false .
  eq (succ(X1) = succ(X)) = (X1 = X) .

  eq (X < X)  = false .
  eq (0 < succ(X)) = true .
  eq (succ(X) < 0) = false .
  eq (succ(X1) < succ(X)) = (X1 < X) .

  eq (X > X)  = false .
  eq (0 > succ(X)) = false .
```

```
  eq (succ(X) > 0) = true .
  eq (succ(X1) > succ(X)) = (X1 > X) .

  eq (X <= X) = true .
  eq (0 <= succ(X)) = true .
  eq (succ(X) <= 0) = false .
  eq (succ(X1) <= succ(X)) = (X1 <= X) .

  eq (X >= X) = true .
  eq (0 >= succ(X)) = false .
  eq (succ(X) >= 0) = true .
  eq (succ(X1) >= succ(X)) = (X1 >= X) .
}

mod! LABEL {
  [Label]
  ops start jazz rock stop  : -> Label
  op _=_ : Label Label -> Bool { comm }
  var L : Label
  eq (L = L) = true .
  eq (start = jazz) = false .
  eq (start = rock) = false .
  eq (start = stop) = false .
  eq (jazz  = rock) = false .
  eq (jazz  = stop) = false .
  eq (rock  = stop) = false .
}

mod! USER {
  [User]
  ops ALICE  BOB : -> User
  op _=_ : User User -> Bool { comm }
  var U : User
  eq (U = U) = true .
  eq (ALICE = BOB) = false .
}

mod* JUKEBOX {
```

```
pr (MYNAT + LABEL + USER)
*[ State ]*
op  init : -> State                                  -- initial state .

bop pc :  State -> Label                             -- observation .
bop quarter :  State -> Nat                          -- observation .

bop startjb :   User   Nat  State -> State           -- action .
bop playmusic :   State -> State                     -- action .
bop playjazz :    State -> State                     -- action .
bop playrock :    State -> State                     -- action .

var  X : Nat
var  U : User
var  S : State

eq pc(init) = start .
eq quarter(init) = 0 .

-- for startjb(ALICE, X, S) and for startjb(BOB, X, S) :
op c-startjb : User  Nat  State -> Bool
eq c-startjb(U, X, S) = pc(S) = start and quarter(S) = 0 .
ceq pc(startjb(U, X, S)) = (if U = ALICE then jazz  else rock  fi)
                                          if c-startjb(ALICE, X, S) .
ceq quarter(startjb(U, X, S)) =  X        if c-startjb(U, X, S) .
ceq startjb(U, X, S) = S                  if not c-startjb(U, X, S) .


-- for playmusic(S) :
op c-playmusic : State -> Bool
eq c-playmusic(S) = (pc(S) = jazz or pc(S) = rock)  and quarter(S) > succ(0) .
ceq    pc(playmusic(S)) = (if pc(S) = jazz then jazz else rock fi)
                                                    if c-playmusic(S) .
ceq quarter(playmusic(S)) = quarter(S) - succ(0)    if c-playmusic(S) .
ceq playmusic(S) = S                                if not c-playmusic(S) .


-- for playjazz(S) :
```

```
   op c-playjazz : State -> Bool
   eq c-playjazz(S) = pc(S) = jazz and quarter(S) = succ(0) .
   ceq      pc(playjazz(S)) = stop             if c-playjazz(S) .
   ceq quarter(playjazz(S)) = 0                if c-playjazz(S) .
   ceq playjazz(S) = S                         if not c-playjazz(S) .

   -- for playrock(S) :
   op c-playrock : State -> Bool
   eq c-playrock(S) = pc(S) = rock and quarter(S) = succ(0) .
   ceq pc(playrock(S)) = stop                  if c-playrock(S) .
   ceq quarter(playrock(S)) = 0                if c-playrock(S) .
   ceq playrock(S) = S                         if not c-playrock(S) .
}
```

付 録B　シミュレーション関係の仕様

```
mod ASIM {
  pr(JUKEBOX)
  ops s1 s1' s2 s2' : ->  State
  vars M N : State
  op asA : State State  -> Bool
  eq asA(M,N) = (quarter(M) = quarter(N) and
                 (pc(M) = pc(N) or (pc(M) = jazz and pc(N) = rock)
                                or (pc(M) = rock and pc(N) = jazz)))  .
}

mod STEP {
  pr(ASIM)
  op stepA : -> Bool
  eq stepA = asA(s1,s1') implies asA(s2,s2') .
}
```

付 録C　証明譜

```
--> 外部アクション: startjb_ALICE,_, startjb_BOB,_,playmusic,
--> 内部アクション: playjazz, playrock

--> I)  Initial state condition
open ASIM .
 red asA(init,init) .
close .


-->   ) Step condition
--> 1. a = startjb_[ALICE,n]
-->    Since a is an actor action, we need to check the two actions:
-->    startjb_[ALICE,n] and startjb_[BOB,n].
--> 1.1 a' = startjb_[ALICE,n]
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq (pc(s1') = start) = false .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .

open STEP .
```

```
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq quarter(s1') = 0 .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq (quarter(s1') = 0) = false .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .

--> 1.2 a' = startjb_[BOB,n]
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
```

```
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq (pc(s1') = start) = false .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq quarter(s1') = succ(0) .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(ALICE,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(ALICE,n,s1) .
```

```
 -- case anaysis
 eq pc(s1') = start .
 eq (quarter(s1') = 0) = false .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .


--> 2. a = startjb_[BOB,n]
-->    Since a is an actor action, we need to check the two actions:
-->    startjb_[ALICE,n] and startjb_[BOB,n].
--> a' = startjb_[BOB,n]
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
 -- case anaysis
 eq (pc(s1') = start) = false .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
```

```
 -- case anaysis
 eq pc(s1') = start .
 eq quarter(s1') = 0 .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq (quarter(s1') = 0) = false .
 -- check if there exists s2'
 eq s2' = startjb(BOB,n,s1') .
 red stepA  .
close .


--> 2.2 a' = startjb_[ALICE,n]
open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
 -- case anaysis
 eq (pc(s1') = start) = false .
```

```
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq quarter(s1') = 0 .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 op n : -> Nat .
 -- assumptions
 -- eq c-startjb(BOB,n,s1) = true .
 eq pc(s1) = start .
 eq quarter(s1) = 0 .
 --
 eq s2 = startjb(BOB,n,s1) .
 -- case anaysis
 eq pc(s1') = start .
 eq (quarter(s1') = 0) = false .
 -- check if there exists s2'
 eq s2' = startjb(ALICE,n,s1') .
 red stepA  .
close .
```

```
--> 3. a = playmusic
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
  eq (pc(s1') = jazz) = false .
  eq (pc(s1') = rock) = false .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false .
 eq pc(s1') = rock .
 eq (quarter(s1') = quarter(s1)) = false .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .
```

```
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false .
 eq pc(s1') = rock .
 eq quarter(s1') = quarter(s1) .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
 eq pc(s1') = jazz .
 eq (quarter(s1') = quarter(s1)) = false .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 -- assumptions
```

```
 -- eq c-playmusic(s1) = true .
 -- eq (pc(s1) = jazz or pc(s1) = rock) = true .
 eq pc(s1) = jazz .
 eq (quarter(s1) > succ(0)) = true .
 --
 eq s2 = playmusic(s1) .
 -- case anaysis
 eq pc(s1') = jazz .
 eq quarter(s1') = quarter(s1) .
 -- check if there exists s2'
 eq s2' = playmusic(s1') .
 red stepA  .
close .
```

```
--> 4. a = playjazz
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq pc(s1') = jazz .
 eq quarter(s1') = succ(0) .
 -- check if there exists s2'
 eq s2' = playjazz(s1') .
 red stepA  .
close .

open STEP .
```

```
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq pc(s1') = jazz .
 eq (quarter(s1') = succ(0)) = false .
 -- check if there exists s2'
 eq s2' = playjazz(s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false .
 eq (quarter(s1') = succ(0)) = false .
 -- check if there exists s2'
 eq s2' = playjazz(s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
```

```
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false .
 eq quarter(s1') = succ(0) .
 eq (pc(s1') = rock) = false .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .

open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = jazz .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playjazz(s1) .
 -- case anaysis
 eq (pc(s1') = jazz) = false  .
 eq quarter(s1') = succ(0) .
 eq pc(s1') = rock .
 -- check if there exists s2'
 eq s2' = playjazz(s1') .
 red stepA  .
close .


--> 5. a = playrock
open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playrock(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
```

```
 eq pc(s1') = rock .
 eq quarter(s1') = succ(0) .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .


open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playjazz(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
 eq pc(s1') = rock .
 eq (quarter(s1') = succ(0)) = false .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .



open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playrock(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
 eq (pc(s1') = rock) = false .
 eq (quarter(s1') = succ(0)) = false .
 -- check if there exists s2'
```

```
 eq s2' = playrock(s1') .
 red stepA  .
close .


open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playrock(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
 eq (pc(s1') = rock) = false .
 eq quarter(s1') = succ(0) .
 eq (pc(s1') = jazz) = false .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
 red stepA  .
close .



open STEP .
 -- arbitrary values
 -- assumptions
 -- eq c-playrock(s1) = true .
 eq pc(s1) = rock .
 eq quarter(s1) = succ(0) .
 --
 eq s2 = playrock(s1) .
 -- case anaysis
 eq (pc(s1') = rock) = false .
 eq quarter(s1') = succ(0) .
 eq pc(s1') = jazz .
 -- check if there exists s2'
 eq s2' = playrock(s1') .
```

```
 red stepA  .
close .
```

謝辞

本研究をご指導いただいた, 北陸先端科学技術大学院大学の緒方和博特任准教授, 二木厚吉教授, 小川瑞史教授, 中村正樹助教, 孔維強博士に深く感謝致します. また, 研究に関する論議に付き合っていただいた言語設計学講座の皆様に感謝致します.

参考文献

[1] S.Schneider, A.Sidiropoulos, CSP and anonymity, in: Proc.ESORICS'96, in: Lecture Notes in Comput. Sci., vol. 1146, Springer-Verlag, Berlin, 1996, pp. 198-218.

[2] Yoshinobu Kawabe, Ken Mano, Hideki Sakurada and Yasuyuki Tsukada : Theorem-proving anonymity of infinite-state systems, Information Processing Letters. Vol.101, pp. 46-51，2007.

[3] 河辺義信, 真野健, 櫻田英樹, 塚田恭章: シミュレーション技法によるセキュリティプロトコルの匿名性検証法. コンピュータセキュリティシンポジウム 2005, 情報処理学会シンポジウムシリーズ, Vol.2005, No.13, pp.43-48.

[4] Kawabe, Y., Mano, K., Sakurada, H., and Tsukada, Y.: Backward Simulations for Anonymity. Proc. of the 6th IFIP WG 1.7 and GI FoMSESS Workshop on Issues in the Theory of Security (WITS'06), pp.206-220 (2006).

[5] 河辺義信, 真野健, 櫻田英樹, 塚田恭章: バックワード匿名シミュレーションを用いた匿名性の検証, 日本ソフトウェア科学会第 22 回大会講演論文集 (日本ソフトウェア科学会, September 2005) CD-ROM Publication (ISSN 1348-0901) 7B-2.

[6] 緒方和博, 二木厚吉: 書き換えによるセキュリティプロトコルの帰納的検証, コンピュータソフトウェア, Vol.20,No.3, pp82–94, 1992.

[7] Kazuhiro Ogata and Kokichi Futatsugi. Proof scores in the OTS/CafeOBJ method. In FMOODS 2003, Volume 2884 of LNCS, pages 170-184. Springer,2003.

[8] CafeOBJ web site. http://www.ldl.jaist.ac.jp/cafeobj/.

[9] Diaconescu R. and Futatsugi K.: cafeobj report. Number 6 in AMAST Series in Computing. Word Scientific,1998.

[10] Futatsugi K.: Formal Methods in CafeOBJ. Lecture Notes in Computer Science,2441,Springer, pp1–20, 2002.(invited paper).