チュートリアル

CafeOBJ 入門 (5) 認証プロトコルの検証

緒方 和博　二木 厚吉　中村 正樹

代数仕様言語 CafeOBJ による認証プロトコルの検証の例を示す．NSLPK 認証プロトコルは，公開鍵暗号方式に基づく，2 つの主体間の相互認証のためのプロトコルである．NSLPK 認証プロトコルに定められたメッセージ送受信を終えると，2 つの主体はある情報を共有する．その情報が，たとえ悪意のある主体が存在していたとしても第三者にもれることはない，という性質を秘匿性という．NSLPK 認証プロトコルが秘匿性を満たすことの証明譜による検証を解説する．

An example of verification of authentication protocols with CafeOBJ algebraic specification language is shown. The NSLPK authentication protocol is based on the public-key cryptosystem. Two principals can use the protocol to achieve the mutual authentication between them. The successful completion of the message exchanges specified by the protocol lets two principals share some information. The secrecy property is as follows: even when there exist malicious principals, the information is never leaked to any third parties. Described is the verification with proof score that the protocol satisfies the (nonce) secrecy property.

1 はじめに

本チュートリアルも残すところ 2 回である．前回，待ち行列を用いる相互排除プロトコル Qlock が相互排除性を満たす検証を例に，等式のみを公理とする CafeOBJ の等式仕様を対象とする，証明譜による検証法を解説した．今回と次回で，証明譜による検証法の 2 つの事例を解説する．今回は認証プロトコルの，次回は通信プロトコルの検証の事例を取り上げる．

ここで扱う認証プロトコルは NSLPK 認証プロトコル [4] [5] である．NSLPK 認証プロトコルに定められたメッセージ送受信を終えると，2 つの主体はある情報を共有する．その情報が，たとえ悪意のある主体が存在していたとしても第三者にもれることはない，という性質を秘匿性という．NSLPK 認証プロトコルが秘匿性を満たすことの証明譜による検証を解説する．

検証の概略は以下のとおりである．(1) まず，NSLPK 認証プロトコルを観測遷移システム $\mathcal{S}_{\mathrm{NSLPK}}$ としてモデル化し，$\mathcal{S}_{\mathrm{NSLPK}}$ の CafeOBJ 仕様を作成する．(2) 次に，秘匿性を定式化するための状態述語 secret を定義し，`Bool` の項で表現する．秘匿性は，secret が $\mathcal{S}_{\mathrm{NSLPK}}$ の不変性として定式化できる．(3) そして，証明譜の作成・確認をとおしてその不変性の証明を行う．

今回のチュートリアルは大きく 2 つの部分から構成されている．1 つは，NSLPK 認証プロトコルのモデル化と仕様記述 (3 節) で，もう 1 つは，NSLPK 認証プロトコルの秘匿性に関する検証 (4 節) である．第 1 部 (モデル化と仕様記述) では，読者が今回のチュートリアルに目をとおすだけで，NSLPK 認証プロトコルの CafeOBJ 仕様を再構築できるくらいに詳細にかつわかりやすく解説することを心がけた．第 2 部 (検

Introducing CafeOBJ (5) : Verification of an Authentication Protocol.

Kazuhiro Ogata, Kokichi Futatsugi, 北陸先端科学技術大学院大学情報科学研究科, Graduate School of Information Science, Japan Advanced Institute of Science and Technology (JAIST).

Masaki Nakamura, 金沢大学理工学域電子情報学類, School of Electrical and Computer Engineering, Kanazawa University.

コンピュータソフトウェア, Vol.26, No.1 (2009), pp.71–83.
[解説論文] 2007 年 3 月 28 日受付.

証) が今回のチュートリアルの主題である．補題を必要とする帰納段階の証明節等，秘匿性の検証にとって重要と思われる箇所を中心に解説している．特に，実例をとおして，以下のことについて詳細にかつわかりやすく解説することを心がけた．

- 場合を特徴づける適切な等式
- 場合分けの基準の選択方法 (理由)
- 補題発見法 1 と 2，およびその組合せによる補題発見
- 補題発見に伴う (連立) 帰納スキーマの変更

2 NSLPK 認証プロトコル

認証プロトコルは，ネットワーク越しに主体が互いに認証するための仕組みである．NSLPK 認証プロトコルは，1983 年に Needham と Schroeder により提案された公開鍵方式に基づく NSPK 認証プロトコル [6] の不具合を修正することで，1995 年に Lowe により提案された認証プロトコル [4] [5] である．

NSLPK 認証プロトコルは以下のように記述できる．

$$\begin{array}{lcl} \text{Msg 1} & p \longrightarrow q & : \quad \mathcal{E}_q(n_p, p) \\ \text{Msg 2} & q \longrightarrow p & : \quad \mathcal{E}_p(n_p, n_q, q) \\ \text{Msg 3} & p \longrightarrow q & : \quad \mathcal{E}_q(n_q) \end{array}$$

各主体には，私有鍵と公開鍵の組が割り当てられているとする．私有鍵は，割り当てられた主体のみが知っており，公開鍵は，プロトコルに参加するすべての主体が知っているとする．$\mathcal{E}_p(m)$ は，主体 p の公開鍵で暗号化されたメッセージ m である．この暗号文は，対応する私有鍵を保有する主体 p だけが復号できる．n_p は，主体 p により生成されるノンスである．ノンスは，高々1 回使われる値である．このプロトコルでは，さらに，ノンスは類推不能であるとする．ノンスとして，乱数を用いることができる．

プロトコルの基本的な振舞： 主体 p が，別の主体 q と相互認証をしたい場合，ノンス n_p を生成し，n_p と識別子 p の組を q の公開鍵で暗号化した $\mathcal{E}_q(n_p, p)$ を q に送る．q が，Msg 1 の種類と思われるメッセージを受け取ると，まず復号を試みる．復号によりノンス n_p と主体の識別子 p を取り出すことができたら，新しいノンス n_q を生成し，n_p，n_q と識別子 q の 3 組を p の公開鍵で暗号化したメッセージ $\mathcal{E}_p(n_p, n_q, q)$ を p に送る．p が，$\mathcal{E}_q(n_p, p)$ を q に送った後，Msg 2 の種類と思われるメッセージを受け取ると，まず復号を試みる．復号により 2 つのノンスと識別子が得られ，ノンスの 1 つが n_p と等しく，識別子が q であれば，p の通信相手は確かに q であることを確信することができ，p は q を認証したことになる．この後，もう 1 つのノンス n_q を q の公開鍵で暗号化したメッセージ $\mathcal{E}_q(n_q)$ を q に送る．q が，$\mathcal{E}_p(n_p, n_q, q)$ を送った後，Msg 3 の種類と思われるメッセージを受け取ると，まず復号を試みる．復号によりノンスが得られ，それが n_q と等しければ，q の通信相手は確かに p であることを確信することができ，q は p を認証したことになる．以上で，p と q の相互認証ができたことになる．このとき，2 つのノンス n_p と n_q は p と q にのみ共有される秘密の情報であると，p と q は信じている．

秘匿性： NSLPK が満たすべき性質の 1 つは (ノンス) 秘匿性である．このことは，p と q にのみ共有されると，p と q が信じている，2 つのノンスは p と q 以外の第三者にもれることはない，ということである．以降で，NSLPK がこの性質を満たすことを CafeOBJ [1] [2] で検証する．検証にさきがけ，NSLPK の CafeOBJ 仕様を作成する．

3 NSLPK の CafeOBJ 仕様の作成

NSLPK のモデルを観測遷移システム [7] [2] [8] として作成し，その観測遷移システムの CafeOBJ 仕様を作成する．

3.1 NSLPK のモデルの作成

侵入者の存在の仮定： 検証では，プロトコルにのみ従って行動する主体に加え，プロトコルを攻撃する主体も存在するとする．プロトコルを攻撃する主体をまとめて，汎用の侵入者 [3] としてモデル化する．汎用の侵入者の存在を仮定して，NSLPK のモデルとして観測遷移システム $\mathcal{S}_{\text{NSLPK}}$ を作成する．

NSLPK **のモデル化：** $\mathcal{S}_{\text{NSLPK}}$ は，観測関数の集合 $\mathcal{O}_{\text{NSLPK}}$，初期状態の集合 $\mathcal{I}_{\text{NSLPK}}$ それに遷移関数の集合 $\mathcal{T}_{\text{NSLPK}}$ から成る．$\mathcal{O}_{\text{NSLPK}}$，$\mathcal{I}_{\text{NSLPK}}$ および $\mathcal{T}_{\text{NSLPK}}$ は以下のとおりである．

$\mathcal{O}_{\text{NSLPK}} \triangleq \{\text{rand} : \text{Sys} \to \text{Rand},$

$$\begin{aligned}
&\text{nw} : \text{Sys} \to \text{Network},\\
&\text{nonces} : \text{Sys} \to \text{NonceBag}\}\\
\mathcal{I}_{\text{NSLPK}} \triangleq \{&s_0 \mid \text{rand}(s_0) = \text{seed} \wedge\\
&\text{nw}(s_0) = \text{empty} \wedge\\
&\text{nonces}(s_0) = \text{empty}\}\\
\mathcal{T}_{\text{NSLPK}} \triangleq \{&\text{send1} : \text{Sys Prin Prin} \to \text{Sys},\\
&\text{send2} : \text{Sys Prin Prin Nonce} \to \text{Sys},\\
&\text{send3} : \text{Sys Prin Prin Nonce Nonce} \to \text{Sys},\\
&\text{fake1} : \text{Sys Prin Prin Nonce} \to \text{Sys},\\
&\text{fake2} : \text{Sys Prin Prin Nonce Nonce} \to \text{Sys},\\
&\text{fake3} : \text{Sys Prin Nonce} \to \text{Sys}\}
\end{aligned}$$

ここで，Sys，Rand，Network，NonceBag，Prin および Nonce は，それぞれ，$\mathcal{S}_{\text{NSLPK}}$ の状態，乱数，ネットワーク，ノンスの多重集合，主体およびノンスの型である．次節で記述するが，ネットワークはメッセージの多重集合としてモデル化する．さらに，侵入者の存在のため，一度送信されたメッセージはネットワークに留まり続けるとしている．というのは、一度送信されたメッセージは，侵入者により何度も送信される可能性があるからである．また，seed は初期状態で利用できる乱数を，empty は空の多重集合を表す．

観測関数： 状態 s が与えられると，観測関数 rand，nw および nonces は，その状態で利用可能な乱数 ($\text{rand}(s)$)，その状態までに送信されたメッセージの多重集合 ($\text{nw}(s)$)，およびその状態までに侵入者により収集されたノンスの多重集合 ($\text{nonces}(s)$) を返す．

遷移関数： 遷移関数 send1，send2 および send3 は，それぞれ，主体がプロトコルに従い，Msg 1，Msg 2 および Msg 3 を送ることに対応する．これに対し，遷移関数 fake1，fake2 および fake3 は，それぞれ，侵入者が，収集したノンスを用いて，Msg 1，Msg 2 および Msg 3 を偽造することに対応する．

$\mathcal{S}_{\text{NSLPK}}$ の定義を完成するには，任意の状態 s が与えられたとき，各遷移関数の適用後に，各観測関数の返す値がどのように変化するかを定めなければならない．このことは，$\mathcal{S}_{\text{NSLPK}}$ の CafeOBJ 仕様を記述するときに示す (3.3 小節参照)．

$\mathcal{S}_{\text{NSLPK}}$ の CafeOBJ 仕様を作成する．その前に，データに関する型 Rand，Network，NonceBag，Prin，Nonce および Cipher の CafeOBJ 仕様を作成する．Cipher は，暗号文の型である．NSLPK におけるメッセージは暗号文である．このため，ネットワークは暗号文の多重集合となる．

3.2 データ型の CafeOBJ 仕様の作成

主体の仕様 `Prin`： 主体の型 Prin のモジュールは以下のとおりである．

```
mod* PRIN {
  [Prin]
  op intr : -> Prin
  op _=_ : Prin Prin -> Bool {comm}
  var P : Prin
  eq (P = P) = true .
}
```

定数 `intr` は汎用の侵入者を表す．

乱数の仕様 `Rand`： 乱数の型 Rand のモジュールは以下のとおりである．

```
mod! RAND {
  [Rand]
  op seed : -> Rand
  op next : Rand -> Rand
  op _=_ : Rand Rand -> Bool {comm}
  vars R1 R2 : Rand
  eq (R1 = R1) = true .
  eq (seed = next(R1)) = false .
  eq (R1 = next(R1)) = false .
  eq (next(R1) = next(R2)) = (R1 = R2) .
}
```

定数 `seed` は，最初に利用可能な乱数を表す．演算 next は，与えられた乱数を基に，それまでに生成されていない乱数を返す関数である．つまり，$\texttt{next}^0(x)$ を x，$\texttt{next}^{n+1}(x)$ を $\texttt{next}(\texttt{next}^n(x))$ と定義すると，$\texttt{next}^0(\texttt{seed})$，$\texttt{next}^1(\texttt{seed})$，$\texttt{next}^2(\texttt{seed})$，... は，すべて異なる乱数を返す (表す)．このことを等式で記述している．

ノンスの仕様 `Nonce`： ノンスの型 Nonce のモジュールは以下のとおりである．

```
mod! NONCE { pr(PRIN + RAND)
```

```
  [Nonce]
  op n : Prin Prin Rand -> Nonce
  op p1 : Nonce -> Prin
  op p2 : Nonce -> Prin
  op r  : Nonce -> Rand
  op _=_ : Nonce Nonce -> Bool {comm}
  vars P1 P2 P12 P22 : Prin
  vars R1 R2 : Rand
  var N : Nonce
  eq p1(n(P1,P2,R1)) = P1 .
  eq p2(n(P1,P2,R1)) = P2 .
  eq r(n(P1,P2,R1))  = R1 .
  eq (N = N) = true .
  eq (n(P1,P2,R1) = n(P12,P22,R2))
     = (P1 = P12 and P2 = P22
                 and R1 = R2) .
}
```

演算 n は，ノンスの構成子である．2 つの主体 $p1$ と $p2$，それに乱数 r が与えられると，項 $\mathtt{n}(p1, p2, r)$ は，主体 $p2$ を認証するために主体 $p1$ により生成されたノンスを表す．そのノンスの唯一性は，乱数 r に依存する．ノンスを表す項が与えられると，演算 p1，p2 および r は，それぞれ，その項の第 1，第 2 および第 3 引数を返す．これらの演算とノンスの等価性を判定する述語_=_の定義は，等式により与えられている．

暗号文の仕様 Cipher：暗号文の型 Cipher のモジュールは以下のとおりである．

```
mod! CIPHER { pr(NONCE)
  [Cipher]
  op enc1 : Prin Nonce Prin -> Cipher
  op enc2 : Prin Nonce Nonce Prin -> Cipher
  op enc3 : Prin Nonce -> Cipher
  op k    : Cipher -> Prin
  op n1   : Cipher -> Nonce
  op n2   : Cipher -> Nonce
  op p    : Cipher -> Prin
  op _=_ : Cipher Cipher -> Bool {comm}
  vars K1 K2 : Prin  vars P1 P2 : Prin
  vars N1 N2 N11 N21 N12 N22 : Nonce
  var C : Cipher
  eq k(enc1(K1,N1,P1)) = K1 .
  eq k(enc2(K1,N1,N2,P1)) = K1 .
  eq k(enc3(K1,N1)) = K1 .
  eq n1(enc1(K1,N1,P1)) = N1 .
  eq n1(enc2(K1,N1,N2,P1)) = N1 .
  eq n1(enc3(K1,N1)) = N1 .
  eq n2(enc2(K1,N1,N2,P1)) = N2 .
  eq p(enc1(K1,N1,P1)) = P1 .
  eq p(enc2(K1,N1,N2,P1)) = P1 .
  eq (C = C) = true .
  eq (enc1(K1,N11,P1) = enc1(K2,N12,P2))
     = (K1 = K2 and N11 = N12
                and P1 = P2) .
  eq (enc1(K1,N11,P1)
      = enc2(K2,N12,N22,P2)) = false .
  eq (enc1(K1,N11,P1) = enc3(K2,N12))
     = false .
  eq (enc2(K1,N11,N21,P1)
      = enc2(K2,N12,N22,P2))
     = (K1 = K2 and N11 = N12 and
        N21 = N22 and P1 = P2) .
  eq (enc2(K1,N11,N21,P1) = enc3(K2,N12))
     = false .
  eq (enc3(K1,N11) = enc3(K2,N12))
     = (K1 = K2 and N11 = N12) .
}
```

演算 enc1，enc2 および enc3 は，それぞれ，Msg 1，Msg 2 および Msg 3 の暗号文の構成子である．主体 p と q，ノンス n_p と n_q が与えられると，項 $\mathtt{enc1}(q, n_p, p)$，$\mathtt{enc2}(p, n_p, n_q, q)$ および $\mathtt{enc3}(n_q)$ は，それぞれ，暗号文 $\mathcal{E}_q(n_p, p)$，$\mathcal{E}_p(n_p, n_q, q)$ および $\mathcal{E}_q(n_q)$ を表す．暗号文を表す項が与えられると．演算 k と n1 は，それぞれ，その項の第 1 および第 2 引数を返す．Msg 2 の暗号文を表す項が与えられると，演算 n2 は，その項の第 3 引数を返す．演算 p は，Msg 1 の暗号文を表す項が与えられると，その項の第 3 引数を返し，Msg 2 の暗号文を表す項が与えられると，その項の第 4 引数を返す．演算_=_は，2 つの暗号文を表す項が等しいか否かを判定する述語である．これらの演算の定義は，等式で与えられている．

汎用の多重集合の仕様 BAG： 暗号文の多重集合の型 Network とノンスの多重集合の型 NonceBag のモジュールを作成するため，まず，汎用の多重集合のモジュールを作成する．汎用の多重集合のモジュールは以下のとおりである．

```
mod! BAG (M :: EQTRIV) {
  [Elt.M < Bag]
  op empty : -> Bag
  op __ : Bag Bag -> Bag {assoc comm}
  op _\in_ : Elt.M Bag -> Bool
  vars X X' : Elt.M  vars S S' : Bag
  eq (empty S) = S .
  eq X \in empty = false .
  eq X \in X' = (X = X') .
  eq X \in (X' S) = (X = X') or (X \in S) .
}
```

仮引数 M を規定するモジュール EQTRIV は以下のとおりである．

```
mod* EQTRIV {
  [Elt]
  op _=_ : Elt Elt -> Bool {comm}
}
```

ソート Elt.M は，ソート Bag の下位ソートとして宣言されている．このことは，多重集合の各要素は，その要素のみを持つ多重集合とみなせる，ということを意味する．定数 empty は，空の多重集合を表しており，演算__は空でない多重集合の構成子である．演算__は，交換律 (comm) と結合律 (assoc) を満たすと宣言されている．演算_\in_は，与えられた要素が与えられた多重集合に含まれるか否かを判定する述語である．最後の 3 つの等式により定義が与えられている．最初の等式 (empty S) = S は，定数 empty は，演算__の単位元であることを規定している．

ノンスの多重集合の仕様 NONCE-BAG： 汎用の多重集合からノンスの多重集合を具現化するためのビューを作成する．

```
view EQTRIV2NONCE from EQTRIV to NONCE {
  sort Elt -> Nonce,
  op _=_ -> _=_
}
```

すると，ノンスの多重集合のモジュールは以下のとおりとなる．

```
mod! NONCE-BAG {
  pr(BAG(M <= EQTRIV2NONCE)
       *{sort Bag -> NonceBag})
}
```

ソート名 Bag を NonceBag に名前変えしている．

暗号文の多重集合の仕様 NETWORK： つづいて，汎用の多重集合から暗号文の多重集合を具現化するためのビューを作成する．

```
view EQTRIV2CIPHER from EQTRIV to CIPHER {
  sort Elt -> Cipher,
  op _=_ -> _=_
}
```

すると，暗号文の多重集合のモジュールは以下のとおりとなる．

```
mod! NETWORK {
  pr(BAG(M <= EQTRIV2CIPHER)
       *{sort Bag -> Network})
}
```

ソート名 Bag を Network に名前変えしている．

3.3 観測遷移システムの CafeOBJ 仕様の作成

観測遷移システム $\mathcal{S}_{\mathrm{NSLPK}}$ で用いるデータ型の CafeOBJ 仕様を作成したので，つづいて $\mathcal{S}_{\mathrm{NSLPK}}$ の CafeOBJ 仕様を作成する。

$\mathcal{S}_{\mathrm{NSLPK}}$ の仕様 NSLPK： $\mathcal{S}_{\mathrm{NSLPK}}$ のモジュールは以下のとおりである．

```
mod* NSLPK { pr(NONCE-BAG + NETWORK)
  *[Sys]*
  op init : -> Sys
  bop rand   : Sys -> Rand
  bop nw     : Sys -> Network
  bop nonces : Sys -> NonceBag
  bop send1 : Sys Prin Prin -> Sys
  bop send2 : Sys Prin Prin Nonce -> Sys
  bop send3 : Sys Prin Prin
                        Nonce Nonce -> Sys
  bop fake1 : Sys Prin Prin Nonce -> Sys
  bop fake2 : Sys Prin Prin
```

```
                          Nonce Nonce -> Sys
  bop fake3 : Sys Prin Nonce -> Sys
  var S : Sys  vars P1 P2 P3 : Prin
  vars N1 N2 : Nonce
  ...
}
```

定数 init は $\mathcal{S}_{\mathrm{NSLPK}}$ の任意の初期状態を表す．その他の演算 (観測関数と遷移関数) は，それぞれ，$\mathcal{S}_{\mathrm{NSLPK}}$ の観測関数と遷移関数に対応する．... の箇所に，$\mathcal{S}_{\mathrm{NSLPK}}$ の初期状態および振舞いを定義する等式が宣言される．それらを以降で記述する．

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．

```
  eq rand(init)   = seed .
  eq nw(init)     = empty .
  eq nonces(init) = empty .
```

これらの等式は $\mathcal{I}_{\mathrm{NSLPK}}$ に対応する．

遷移関数 send1 の定義： 遷移関数 send1 を定義する等式は以下のとおりである．

```
  eq rand(send1(S,P1,P2)) = next(rand(S)) .
  eq nw(send1(S,P1,P2))
     = (enc1(P2,n(P1,P2,rand(S)),P1)
        nw(S)) .
  eq nonces(send1(S,P1,P2))
     = (if P2 = intr
        then (n(P1,P2,rand(S)) nonces(S))
        else nonces(S) fi) .
```

どの主体も，いずれかの主体と相互認証をしたいときにはいつでも Msg 1 を送ることができるので，遷移関数 send1 の効力条件は常に成り立つ．このため，上記の定義では効力条件を省略している．

この遷移関数により生成される暗号文 enc1(P2,n(P1,P2,rand(S)),P1) を侵入者が復号できるとき，つまり P2 が intr のとき，そこに含まれるノンス n(P1,P2,rand(S)) を侵入者は収集する．つまり，そのノンスをノンスの多重集合 nonces(S) に加える．

遷移関数 send2 の定義： 遷移関数 send2 を定義する等式は以下のとおりである．

```
  op c-send2 : Sys Prin Prin Nonce -> Bool
  eq c-send2(S,P1,P2,N1)
     = enc1(P1,N1,P2) \in nw(S) .
  ceq rand(send2(S,P1,P2,N1))
      = next(rand(S))
      if c-send2(S,P1,P2,N1) .
  ceq nw(send2(S,P1,P2,N1))
      = (enc2(P2,N1,n(P1,P2,rand(S)),P1)
         nw(S)) if c-send2(S,P1,P2,N1) .
  ceq nonces(send2(S,P1,P2,N1))
      = (if P2 = intr
         then (N1 n(P1,P2,rand(S))
               nonces(S))
         else nonces(S) fi)
      if c-send2(S,P1,P2,N1) .
  ceq send2(S,P1,P2,N1)
      = S if not c-send2(S,P1,P2,N1) .
```

この遷移関数の効力条件は，主体 P1 宛の Msg 1 がネットワークに存在している，である．もし存在していれば，そのメッセージを P1 は受け取ることができるため，効力条件をこのようにしている．この遷移関数により生成される暗号文が侵入者により復号できるとき，そこに含まれる 2 つのノンスを侵入者は収集する．

遷移関数 send3 の定義： 遷移関数 send3 を定義する等式は以下のとおりである．

```
  op c-send3 : Sys Prin Prin
                   Nonce Nonce -> Bool
  eq c-send3(S,P1,P2,N1,N2)
     = enc2(P1,N1,N2,P2) \in nw(S) and
       enc1(P2,N1,P1) \in nw(S) .
  eq  rand(send3(S,P1,P2,N1,N2))
      = rand(S) .
  ceq nw(send3(S,P1,P2,N1,N2))
      = (enc3(P2,N2) nw(S))
      if c-send3(S,P1,P2,N1,N2) .
  ceq nonces(send3(S,P1,P2,N1,N2))
      = (if P2 = intr then (N2 nonces(S))
                      else nonces(S) fi)
      if c-send3(S,P1,P2,N1,N2) .
  ceq send3(S,P1,P2,N1,N2)
```

```
    = S if not c-send3(S,P1,P2,N1,N2) .
```

この遷移関数の効力条件は，主体 P1 からの主体 P2 宛の Msg 1 とそのメッセージに対する返答の Msg 2 がネットワークに存在している，である．このことは，最初のメッセージを P2 が受け取り，その返答を P1 に返したことを意味する．この遷移関数により生成される暗号文が侵入者により復号できるとき，そこに含まれるノンスを侵入者は収集する．

遷移関数 fake1 の定義： 遷移関数 fake1 を定義する等式は以下のとおりである．

```
op c-fake1 : Sys Prin Prin Nonce -> Bool
eq c-fake1(S,P1,P2,N1)
   = N1 \in nonces(S) .
eq  rand(fake1(S,P1,P2,N1)) = rand(S) .
ceq nw(fake1(S,P1,P2,N1))
    = (enc1(P2,N1,P1) nw(S))
    if c-fake1(S,P1,P2,N1) .
eq  nonces(fake1(S,P1,P2,N1))
    = nonces(S) .
ceq fake1(S,P1,P2,N1)
    = S if not c-fake1(S,P1,P2,N1) .
```

この遷移関数は，侵入者により収集されたノンスを基に Msg 1 を偽造する．この遷移関数により生成される暗号文に含まれるノンスは，すでに侵入者により収集されているものであるから，たとえ暗号文が侵入者により復号できたとしても収集する必要はない．

遷移関数 fake2 の定義： 遷移関数 fake2 を定義する等式は以下のとおりである．

```
op c-fake2 : Sys Prin Prin
                 Nonce Nonce -> Bool
eq c-fake2(S,P1,P2,N1,N2)
   = N1 \in nonces(S) and
     N2 \in nonces(S) .
eq  rand(fake2(S,P1,P2,N1,N2))
    = rand(S) .
ceq nw(fake2(S,P1,P2,N1,N2))
    = (enc2(P1,N1,N2,P2) nw(S))
   if c-fake2(S,P1,P2,N1,N2) .
eq  nonces(fake2(S,P1,P2,N1,N2))
    = nonces(S) .
ceq fake2(S,P1,P2,N1,N2)
    = S if not c-fake2(S,P1,P2,N1,N2) .
```

遷移関数 fake3 の定義： 遷移関数 fake3 を定義する等式は以下のとおりである．

```
op c-fake3 : Sys Prin Nonce -> Bool
eq c-fake3(S,P1,N1) = N1 \in nonces(S) .
eq  rand(fake3(S,P1,N1)) = rand(S) .
ceq nw(fake3(S,P1,N1))
    = (enc3(P1,N1) nw(S))
    if c-fake3(S,P1,N1) .
eq  nonces(fake3(S,P1,N1)) = nonces(S) .
ceq fake3(S,P1,N1)
    = S if not c-fake3(S,P1,N1) .
```

4 NSLPK の秘匿性に関する検証

秘匿性を定式化し，NSLPK が秘匿性を満たしていることを検証する．

4.1 秘匿性の定式化

侵入者により収集されるノンスの多重集合を用いて秘匿性を定式化する．もし，その中に，侵入者以外の主体 q を認証するために侵入者以外の主体 p により生成されたノンス $\mathrm{n}(p,q,r)$ が含まれていれば，秘匿性に反することになる．というのは，このノンスが，p でも q でもない第三者である侵入者に洩れたからである．秘匿性は，そのようなノンスを侵入者が取得できない，ということで表すことができる．

そこで，状態述語 secret を以下のとおりに定義する．

$$\mathrm{secret}(s) \triangleq \forall n \in \mathrm{Nonce}.\,(n \in \mathrm{nonces}(s)) \Rightarrow \mathrm{p1}(n) = \mathrm{intr} \vee \mathrm{p2}(n) = \mathrm{intr})$$

すると，秘匿性は，$\forall s : \mathcal{R}_{\mathcal{S}_{\mathrm{NSLPK}}}.\mathrm{secret}(s)$ として定式化できる．ここで，$\mathcal{R}_{\mathcal{S}_{\mathrm{NSLPK}}}$ は，$\mathcal{S}_{\mathrm{NSLPK}}$ の到達可能状態すべての集合である．つまり，秘匿性とは，状態述語 secret が $\mathcal{S}_{\mathrm{NSLPK}}$ の不変性である，ということである．このことは，侵入者が収集できるノンスは，侵入者自身により生成されたか，侵入者を認証するために他の主体により生成されたかのいずれかである，ことを意味する．つまり，侵入者以外の 2 つの主体により共有されるノンスが，侵入者に洩れることはない，ということである．

NSLPK が秘匿性を満たすことの検証は，$\forall s$: $\mathcal{R}_{\mathcal{S}_{\mathrm{NSLPK}}}.\mathrm{secret}(s)$ を証明することである．以降で，CafeOBJ による $\forall s : \mathcal{R}_{\mathcal{S}_{\mathrm{NSLPK}}}.\mathrm{secret}(s)$ の証明について記述する．

4.2 秘匿性の検証

まず以下のモジュールを宣言する．

```
mod INV { pr(NSLPK)
  op n1 : -> Nonce
  op inv1 : Sys Nonce -> Bool
  var S : Sys  var N1 : Nonce
  eq inv1(S,N1) = ((N1 \in nonces(S))
         implies (p1(N1) = intr or
                  p2(N1) = intr)) .
}
```

演算 inv1 は状態述語 secret に対応する．定数 n1 は任意のノンスを表す．

$\forall s : \mathcal{R}_{\mathcal{S}_{\mathrm{NSLPK}}}.\mathrm{secret}(s)$ の証明は，$\mathcal{R}_{\mathcal{S}_{\mathrm{NSLPK}}}$ 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 (IS1 と呼ぶ) は図 1 のように記述できる (本チュートリアルの第 4 回を参照)．前提は，7 つの言明 (1 つの帰納基底と 6 つの帰納段階) から構成される．7 つの言明のそれぞれの証明譜 (節) を作成・確認することで，$\forall s : \mathcal{R}_{\mathcal{S}_{\mathrm{NSLPK}}}.\mathrm{rc}(s)$ の証明を行う．

各帰納段階で証明すべき論理式を記述したモジュールを以下のとおりに宣言する．

```
mod ISTEP { pr(INV)
  ops s s' : -> Sys
  op istep1 : Nonce -> Bool
  var N1 : Nonce
  eq istep1(N1)
     = inv1(s,N1) implies inv1(s',N1) .
}
```

定数 s は任意の状態を表し，定数 s' は状態 s の事後状態を表す．項 inv1(s',N1) は各帰納段階で証明すべき論理式である．項 inv1(s,N1) 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．

```
{[1-init],
[1-send1]*, [1-send2]*, [1-send3]*,
[1-fake1]*, [1-fake2]*, [1-fake3]*}
implies [inv1]*
```

図 1　帰納スキーマ 1

4.2.1 inv1 の証明譜

帰納基底の証明節： 帰納基底 ([1-init]) の証明節は以下のとおりである．

```
open INV
  red inv1(init,n1) .
close
```

この証明節に対し，CafeOBJ は true を返す．

遷移関数 send2 に関する帰納段階： 遷移関数 send2 に関する帰納段階 ([1-send2]*) について考える．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．

前者の証明節を，場合分けにより分割する．以下の Bool の 6 つの項に基づいて，前者の証明節を以下のとおり 7 つに分割する．

- intr = p2
- n1 = n(p1,p2,rand(s))
- n1 = m
- p1(m) = p2
- p2(m) = p2
- m \in nonces(s)

ここで，定数 p1 と p2 は任意の主体を，定数 m は任意のノンスを表す．これら 3 つの定数は，定数 s と共にアクション send2 の引数に用いられる．つまり，事後状態 s' は，send2(s,p1,p2,m) と定義される．

分割された 7 つの場合は以下のとおりである．

1. not(intr = p2)
2. intr = p2, n1 = n(p1,p2,rand(s))
3. intr = p2, not(n1 = n(p1,p2,rand(s))), not(n1 = m)
4. intr = p2, not(n1 = n(p1,p2,rand(s))), n1 = m, p1(m) = p2
5. intr = p2, not(n1 = n(p1,p2,rand(s))),

`n1 = m, not(p1(m) = p2), p2(m) = p2`

6. `intr = p2, not(n1 = n(p1,p2,rand(s))), n1 = m, not(p1(m) = p2), not(p2(m) = p2), m \in nonces(s)`
7. `intr = p2, not(n1 = n(p1,p2,rand(s))), n1 = m, not(p1(m) = p2), not(p2(m) = p2), not(m \in nonces(s))`

1 番目から 6 番目までのいずれの証明節に対しても，CafeOBJ は `true` を返すが，7 番目の証明節に対しては，`false` を返す．`false` を返すからといって証明に失敗したというわけではない．7 番目の仮定が起こり得ない状態 (到達不可能状態) であることを示すことができればよい．

場合分けの基準の選択理由： 6 つの項を場合分けの規準に選ぶ理由は以下のとおりである．

- `intr = p2`：この項が簡約結果に含まれる条件付選択の条件部分に現れることと，この項が `false` に等しいとき (1 番目の証明節に対し)，CafeOBJ が `true` を返すからである．この項が `true` に等しいとき，`p2 = intr` ではなく，`intr = p2` を用いる．というのは，効力条件と等価である等式

  ```
  eq enc1(p1,m,p2) \in nw(s) = true .
  ```

  の左辺に `p1` が含まれているからである．`intr = p2` の代わりに，`p2 = intr` を用いることもできるが，上記の等式の左辺を `enc1(p1,m,intr)` にしなければならないかもしれない．
- `n1 = n(p1,p2,rand(s))`：この項が簡約結果 (の先頭) に現れることと，この項が `true` に等しいとき (2 番目の証明節に対し)，CafeOBJ が `true` を返すからである．
- `n1 = m`：この項が簡約結果 (の先頭) に現れることと，この項が `false` に等しいとき (3 番目の証明節に対し)，CafeOBJ が `true` を返すからである．
- `p1(m) = p2`：この項が簡約結果 (の先頭) に現れることと，この項が `true` に等しいとき (4 番目の証明節に対し)，CafeOBJ が `true` を返すからである．
- `p2(m) = p2`：この項が簡約結果 (の先頭) に現れることと，この項が `true` に等しいとき (5 番目の証明節に対し)，CafeOBJ が `true` を返すからである．
- `m \in nonces(s)`：この項そのものが簡約結果であるからである．もちろん，CafeOBJ は，`true` に等しいとき (6 番目の証明節に対し) `true` を，`false` に等しいとき (7 番目の証明節に対し) `false` を返す．

補題発見法 1 による補題発見 (inv2)： 7 番目の仮定が起こり得ない状態であることを示す補題を類推する．ここで用いる補題発見法は，NSLPK および観測遷移システム $\mathcal{S}_{\mathrm{NSLPK}}$ のある程度の理解を必要とする補題発見法 1(前回のチュートリアル参照) である．7 番目の仮定のもとでの，遷移関数 `send2` の適用により，以下のメッセージ (暗号文) が追加される (7 番目の証明節において，`nw(s')` を簡約することで確認できる)．

```
enc2(p2,m,n(p1,p2,rand(s)),p1)
```

7 番目の仮定から，`p2` は侵入者である．このため，この暗号文に含まれる 2 つのノンス `m` と `n(p1,p2,rand(s))` は，侵入者により収集される (7 番目の証明節において，`nonces(s')` を簡約することで確認できる)．また，7 番目の仮定では，ノンス `m` は，侵入者以外の主体を認証するために侵入者以外の主体により生成されたことになっており，侵入者の収集するノンスの多重集合には含まれていないことになっている．このため，7 番目の証明節に対し，CafeOBJ は `false` を返すことになる．しかし，遷移関数 `send2` の効力条件は，暗号文 `enc1(p1,m,p2)` がネットワークに存在する，ということである．この暗号文は，主体 `p1` により侵入者 `p2` に送られたものである．このことから，ノンス `m` は，侵入者を認証するために生成されたものであることがわかる．このことは 7 番目の仮定と矛盾している．以上の観察から，以下の補題を類推できる．

```
op inv2 : Sys Nonce Prin -> Bool
eq inv2(S,N1,Q1)
   = ((enc1(Q1,N1,intr) \in nw(S))
      implies (p1(N1) = intr or
               p2(N1) = intr or
```

```
                N1 \in nonces(S))) .
```

これをモジュール INV に追加する．これに伴い，ソート Prin の変数 Q1 と定数 q1 も追加する．定数 q1 は，任意の主体を表す．

この補題を用いることで，7 番目の証明節に対し CafeOBJ が true を返すようにできる．最終的な 7 番目の証明節は以下のとおりである．

```
open ISTEP
  ops p1 p2 : -> Prin .  op m : -> Nonce .
  eq enc1(p1,m,p2) \in nw(s) = true .
  eq intr = p2 .
  eq (n1 = n(p1,p2,rand(s))) = false .
  eq n1 = m .
  eq (p1(m) = p2) = false .
  eq (p2(m) = p2) = false .
  eq m \in nonces(s) = false .
  eq s' = send2(s,p1,p2,m) .
  red inv2(s,n1,p1) implies istep1(n1) .
close
```

補題発見法 2 による補題発見 (inv2')： 補題 inv2 を用いなければ，7 番目の証明節に対し，CafeOBJ は false を返す．このため，補題発見法 2(前回のチュートリアル参照) により，7 番目の場合を特徴づける等式の集合から，機械的に補題を作ることもできる．このように作られる補題は以下のとおりである．

```
  op inv2' : Sys Nonce Prin -> Bool
  eq inv2'(S,N1,Q1)
     = not(enc1(p1,n1,intr) \in nw(s)
        and not(n1 = n(p1,intr,rand(s)))
        and not(p1(n1) = intr)
        and not(p2(n1) = intr)
        and not(n1 \in nonces(s))) .
```

inv2' を用いても 7 番目の証明節に対し CafeOBJ が true を返すようにできる．inv2 と inv2' は以下の関係を満たす．

$$\forall s \in \mathcal{R}_{\mathcal{S}_{\mathrm{NSLPK}}}.\forall n \in \mathrm{Nonce}.\forall p \in \mathrm{Prin}.$$
$$\texttt{inv2}(s,n,p)$$
$$\Downarrow$$
$$\forall s \in \mathcal{R}_{\mathcal{S}_{\mathrm{NSLPK}}}.\forall n \in \mathrm{Nonce}.\forall p \in \mathrm{Prin}.$$
$$\texttt{inv2}'(s,n,p)$$

inv2 と inv2' のいずれを補題として用いてもよいが，不変性の証明は inv2 のほうが少ない労力でできる．

補題発見法 1 と補題発見法 2 の組合せ： このように，不変性の証明の労力の観点からは，補題発見法 1 で求める補題のほうが，補題発見法 2 で求める補題より優れていることのほうがおおい．しかし，補題発見法 1 では，対象となる観測遷移システムについてある程度の理解を必要とする．対象となる観測遷移システムがある不変性を満たすかどうかの検証をするのであるから，その観測遷移システムについてある程度理解しているべきであるし，証明譜を記述することで理解が深まると期待できる．このため，より優れた補題を発見できるのであれば，常に補題発見法 1 だけを用いればよいように思える．しかし，対象となる観測遷移システムの理解を深めるため (どのような不変性を満たすかを知るため) に検証しているのであるから，検証している途中であれば，その観測遷移システムを十分に理解していないことにもなる．つまり，補題発見法 1 だけでは有用な補題を探すことができないかもしれない．このようなとき，補題発見法 1 と補題発見法 2 を組み合わせて用いることを考えることができる．まず，補題発見法 2 で機械的に補題の候補を作成し，対象となる観測遷移システムについての理解に基づき，補題候補を簡略化 (強化) する．上記の例では，まず inv2' を機械的に作成し，観測遷移システム $\mathcal{S}_{\mathrm{NSLPK}}$ の理解に基づき，inv2' から not(n1 = n(p1,intr,rand(s))) を削除することで，inv2 を得ることができる．

遷移関数 send3 に関する帰納段階： 次に，遷移関数 send3 に関する帰納段階 ([1-send3]$*$) について考える．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．

前者の証明節を，場合分けにより分割する．以下の Bool の 5 つの項に基づいて，前者の証明節を以下のとおり 6 つに分割する．

- intr = p2
- n1 = m2
- p1(m2) = p2

- p2(m2) = p2
- m2 \in nonces(s)

分割された 6 つの場合は以下のとおりである.

1. not(intr = p2)
2. intr = p2, not(n1 = m2)
3. intr = p2, n1 = m2, p1(m2) = p2
4. intr = p2, n1 = m2, not(p1(m2) = p2), p2(m2) = p2
5. intr = p2. n1 = m2, not(p1(m2) = p2), not(p2(m2) = p2). m2 \in nonces(s)
6. intr = p2, n1 = m2, not(p1(m2) = p2), not(p2(m2) = p2), not(m2 \in nonces(s))

1 番目から 5 番目までのいずれの証明節に対しても，CafeOBJ は true を返すが，6 番目の証明節に対しては，false を返す．5 つの項を場合分けの規準に選ぶ理由は，遷移関数 send2 のときと同じである．

補題発見 (inv3)**：** 先の補題を類推したのと同じように，6 番目の証明節に関し，以下の補題を類推できる．

```
op inv3 : Sys Nonce Nonce Prin -> Bool
eq inv3(S,N1,N2,Q1)
   = ((enc2(Q1,N1,N2,intr) \in nw(S))
      implies (p1(N2) = intr or
               p2(N2) = intr or
               N2 \in nonces(S))) .
```

これをモジュール INV に追加する．これに伴い，ソート Nonce の変数 N2 と定数 n2 も追加する．定数 n2 は，任意のノンスを表す.

この補題を用いることで，6 番目の証明節に対し CafeOBJ が true を返すようにできる．最終的な 6 番目の証明節は以下のとおりである.

```
open ISTEP
  ops p1 p2 : -> Prin .
  ops m1 m2 : -> Nonce .
  eq enc1(p2,m1,p1) \in nw(s) = true .
  eq enc2(p1,m1,m2,p2) \in nw(s) = true .
  eq intr = p2 .
  eq n1 = m2 .
  eq (p1(m2) = p2) = false .
  eq (p2(m2) = p2) = false .
  eq m2 \in nonces(s) = false .
  eq s' = send3(s,p1,p2,m1,m2) .
  red inv3(s,m1,m2,p1) implies istep1(n1) .
close
```

{[1-init],
[1-3-send1]∗, [1-3-send2]∗, [1-3-send3]∗,
[1-3-fake1]∗, [1-3-fake2]∗, [1-3-fake3]∗,
[2-init],
[2-3-send1]∗, [2-3-send2]∗, [2-3-send3]∗,
[2-3-fake1]∗, [2-3-fake2]∗, [2-3-fake3]∗,
[3-init],
[3-3-send1]∗, [3-3-send2]∗, [3-3-send3]∗,
[3-3-fake1]∗, [3-3-fake2]∗, [3-3-fake3]∗}
implies {[inv1]∗, [inv2]∗, [inv3]∗}

図 2 連立帰納スキーマ 3

inv1 の証明譜の残り：残りの 4 つの帰納段階も同じように証明できる．残りの 4 つの帰納段階の証明では，補題を必要としない.

帰納スキーマから連立帰納スキーマへの変更： 2 つの補題 inv2 と inv3 を導入したため，帰納スキーマ IS1 の代わりに，図 2 のように記述される連立帰納スキーマ (SIS3 と呼ぶ) を用いる.

inv1 の証明譜は，前提の最初の 7 つの言明に，inv2 の証明譜は，つづく 7 つの言明に，そして inv3 の証明譜は，最後の 7 つの言明に対応している．これら 3 つの証明譜のすべての証明節が有効になれば (CafeOBJ が true を返せば)，inv1 が $\mathcal{S}_{\mathrm{NSLPK}}$ にとって不変であるのみならず，inv2 と inv3 もそうであることが証明されたことになる.

本チュートリアルの第 4 回で説明したとおり，IS1 から SIS3 へ変更しても，これまでに作成した inv1 の証明譜は一切修正する必要はない．どちらのスキーマも，inv1 の帰納基底に対応する言明として，同じものを使っている．IS1 と SIS3 の帰納段階に対応する言明の違いは，用いることのできる帰納法の仮定にある．SIS3 のほうが多くの帰納法の仮定を用いることができ，IS1 の帰納法の仮定を含んでいる．このため，IS1 で証明できた帰納段階は，SIS3 でもできることになる.

inv2 と inv3 の証明譜を記述するのに先立ち，以

下をモジュール ISTEP に追加する．

```
op istep2 : Nonce Prin -> Bool
op istep3 : Nonce Nonce Prin -> Bool
eq istep2(N1,Q1) = inv2(s,N1,Q1)
                 implies inv2(s',N1,Q1) .
eq istep3(N1,N2,Q1) = inv3(s,N1,N2,Q1)
                 implies inv3(s',N1,N2,Q1) .
```

これに伴い，ソート Prin の変数 Q1 とソート Nonce の変数 N2 も追加する．

4.2.2 inv2 の証明譜

次に，inv2 の証明譜 (の一部) について記述する．

遷移関数 send1 に関する帰納段階： 遷移関数 send1 に関する帰納段階 ([1-send3]∗) について考える．遷移関数 send1 の効力条件は常に成り立っているため，効力条件に基づく場合分けを行う必要はない．

場合分けを行わなければ，対応する証明節に対し，CafeOBJ は true でも false でもない Bool の項を返す．そこで，以下の Bool の 3 つの項に基づいて，証明節を以下のとおり 4 つに分割する．

- enc1(q1,n1,intr)
  = enc1(p2,n(p1,p2,rand(s)),p1)
- p2 = intr
- n1 = n(p1,intr,rand(s))

分割された 4 つの場合は以下のとおりである．

1. enc1(q1,n1,intr)
   = enc1(p2,n(p1,p2,rand(s)),p1)
2. not(enc1(q1,n1,intr)
   = enc1(p2,n(p1,p2,rand(s)),p1))
   not(p2 = intr)
3. not(enc1(q1,n1,intr)
   = enc1(p2,n(p1,p2,rand(s)),p1))
   p2 = intr
   n1 = n(p1,intr,rand(s))
4. not(enc1(q1,n1,intr)
   = enc1(p2,n(p1,p2,rand(s)),p1))
   p2 = intr
   not(n1 = n(p1,intr,rand(s)))

4 つすべての証明節に対し，CafeOBJ は true を返す．

場合を特徴づける適切な等式： ただし，1 番目の証明節で，1 番目の項をそのまま用いたとすると，つまり，以下の等式

```
eq enc1(q1,n1,intr)
   = enc1(p2,n(p1,p2,rand(s)),p1) .
```

を用いたとすると，CafeOBJ は true でも false でもない Bool の項を返す．この等式の代わりに，この等式と等価な以下の 3 つの等式

```
eq q1 = p2 .
eq n1 = n(p1,p2,rand(s)) .
eq p1 = intr .
```

を用いると，CafeOBJ は true を返す．場合を特徴づける等式の適切さの基準 (前回のチュートリアル参照) からも，最初の 1 つの等式を用いるよりも，あとの 3 つの等式を用いたほうがよいことがわかる．

3 番目と 4 番目の証明節に対しては，3 つの項をそのまま用いたとしても．CafeOBJ は true を返す．このため，場合を特徴づける等式としてより適切なものがあったとしても置き換える必要はない．しかし，場合を特徴づける適切な等式の例示を目的とし，3 番目の場合を特徴づけるより適切な等式を示す．それらは以下のとおりである．

```
eq (enc1(q1,n(p1,intr,rand(s)),intr)
    = enc1(intr,n(p1,intr,rand(s)),p1))
   = false .
eq p2 = intr .
eq n1 = n(p1,intr,rand(s)) .
```

1 番目の等式に現れる 2 番目と 3 番目の等式の左辺を，対応する右辺に置き換えている．4 番目の場合を特徴づけるより適切な等式も同様に得ることができる．

秘匿性検証の残り： 帰納基底は inv1 と同じように証明でき，残りの 5 つの帰納段階も同じように証明できる．inv2 では補題を必要としない．inv3 の証明譜も inv2 と同じように作成できる．inv3 でも補題を必要としない．

補題の導入と連立帰納スキーマ： NSLPK の秘匿性に関する検証では，2 つ以上の状態述語の不変性の証明は相互に依存していない．このため連立帰納スキーマを用いなくてもよいことになる．しかし，2 つ以上の状態述語の不変性の証明が相互に依存していないとき，連立帰納スキーマを用いても用いなくても，作

成される証明譜は同じであり，連立帰納スキーマを用いる弊害は何もない．このため，補題を導入したときは，常に連立帰納スキーマを用いるという戦略を使ってもよいことになる．

5 おわりに

NSLPK 認証プロトコルが秘匿性を満たすことの証明譜による検証を解説した．特に，実例をとおして，以下のことを解説した．

- 場合を特徴づける適切な等式
- 場合分けの基準の選択方法 (理由)
- 補題発見法 1 と 2，およびその組合せによる補題発見
- 補題発見に伴う (連立) 帰納スキーマの変更

謝辞

有益なコメントをいただいた査読者の方々と編集委員の方々に感謝いたします．

参 考 文 献

[1] Diaconescu, R. and Futatsugi, K.: *CafeOBJ report*, AMAST Series in Computing, Vol. 6, World Scientific, 1998.

[2] Diaconescu, R., Futatsugi, K. and Ogata, K.: CafeOBJ: Logical Foundations and Methodologies, *Computing and Informatics*, Vol. 22(2003), pp. 257–283.

[3] Dolev, D. and Yao, A. C.: On the security of public key protocols, *IEEE Transactions on Information Theory*, Vol. IT-29(1983), pp. 198–208.

[4] Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol, *Information Processing Letters*, Vol. 56(1995), pp. 131–133.

[5] Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR, in *2nd Workshop on Tools and Algorithms for Construction and Analysis of Systems* (*2nd TACAS*), LNCS, Vol. 1055, Springer, 1996, pp. 147–166.

[6] Needham, R. M. and Schroeder, M. D.: Using Encryption for Authentication in Large Networks of Computers, *Communication of the ACM*, Vol. 21, No. 12(1978), pp. 993–999.

[7] Ogata, K. and Futatsugi, K.: Proof Scores in the OTS/CafeOBJ Method, in *6th IFIP WG6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems* (*6th FMOODS*), LNCS, Vol. 2884, Springer, 2003, pp. 170–184.

[8] Ogata, K. and Futatsugi, K.: Some Tips on Writing Proof Scores in the OTS/CafeOBJ Method, in *Algebra, Meaning, and Computation: A Festschrift Symposium in Honor of Joseph Goguen*, LNCS, Vol. 4060, Springer, 2006, pp. 596–615.