チュートリアル

CafeOBJ 入門 (6) 通信プロトコルの検証

緒方 和博　二木 厚吉　中村 正樹

代数仕様言語 CafeOBJ による通信プロトコルの検証の例を示す．SCP 通信プロトコルは，ABP 通信プロトコルの簡略版で，送受信者間の通信路として信頼できないセルを用いる．受信者が N 個のパケットを受け取ったとき，それらは送信者が送った最初の N 個のパケットで，順番を保存している，という性質を通信信頼性という．SCP 通信プロトコルが通信信頼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 communication protocols with CafeOBJ algebraic specification language is shown. The SCP communication protocol is a simplified version of the ABP communication protocol, which uses unreliable cells as communication channels between senders and receivers. The reliable communication property is as follows: when a receiver receives N packets, they are the first N packets that a sender has sent and the order in which the N packets has been sent is preserved. Described is the verification with proof score that the SCP communication protocol satisfies the reliable communication property.

1　はじめに

今回が，本チュートリアルの最終回である．前回，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．最終回である今回は，通信プロトコルが通信信頼性を満たすことの証明譜による検証を解説する．

ここで扱う通信プロトコルは，ABP(Alternating Bit Protocol) 通信プロトコル [1] の簡略版である．簡略版を SCP(a Simple Communication Protocol) 通信プロトコルと呼ぶ．ABP が送受信者間の通信路として信頼できない待ち行列を用いるのに対し，SCP は信頼できないセルを用いる．受信者が N 個のパケットを受け取ったとき，それらは送信者が送った最初の N 個のパケットで，順番を保存している，という性質を通信信頼性という．SCP 通信プロトコルが通信信頼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，SCP 通信プロトコルを観測遷移システム $\mathcal{S}_{\mathrm{SCP}}$ としてモデル化し，$\mathcal{S}_{\mathrm{SCP}}$ の CafeOBJ 仕様を作成する．(2) 次に，通信信頼性を定式化するための状態述語 rc を定義し，`Bool` の項で表現する．通信信頼性は，rc が $\mathcal{S}_{\mathrm{SCP}}$ の不変性として表現する．(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．

前回と同じく，今回のチュートリアルも大きく 2 つの部分から構成されている．1 つは，SCP 通信プロトコルのモデル化と仕様記述 (3 節) で，もう 1 つは，SCP 通信プロトコルの通信信頼性に関する検証 (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，SCP 通信プロトコルの CafeOBJ 仕様を再構築できるくらいに詳細にかつわかりやく解説することを心がけた．第 2 部 (検証) 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，通信信頼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

Introducing CafeOBJ (6) : Verification of a Communication 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.2 (2009), pp.93–106.
[解説論文] 2007 年 3 月 28 日受付.

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．

- 場合分けの基準の選択方法 (理由)
- 模擬実験による SCP プロトコルの理解と補題発見
- 場合分けと補題発見のどちらを試みるかについて
- 補題発見に伴う (連立) 帰納スキーマの変更
- データ型の補題

2 SCP 通信プロトコル

セルを介して送信者 (sender) から受信者 (receiver) にパケットを送る仕組み (プロトコル) を考える (図 1 参照)．ただし，セルは完全に信頼できるものではなく，蓄えている情報が消失するかもしれないとする．2 つのセルを用いる．1 つ (cell1) は，送信者から受信者に情報を伝えるために，もう 1 つ (cell2) は，受信者から送信者に情報を伝えるために用いる．送信者はブール型の変数 bit1 を持ち，受信者はブール型の変数 bit2 と受理したパケットを保存するリスト list を持つ．

プロトコルの基本的な振舞： 送信者と受信者は以下のことを繰り返し行う．

- send1：送信者は，組 $\langle \mathrm{bit1}, \mathrm{p}_i \rangle$ を cell1 に入れる．ここで，p_i は i 番目に送りたいパケットである．
- rec1：送信者は，cell2 が空でなければ，その中のビットを取り出し，bit1 と比較する．等しければ何もしない．そうでなければ，bit1 を補完し，i の値を 1 増やす．
- send2：受信者は，bit2 を cell2 に入れる．
- rec2：受信者は，cell1 が空でなければ，その中の組を取り出し，組の最初の要素と bit2 を比較する．等しくなければ何もしない．等しければ，組の 2 番目の要素を list に追加し，bit2 を補完する．

初期状態では，bit1 と bit2 は共に f(alse) で，i は 0 で，cell1 と cell2 は共に空で，list は空である．

この通信プロトコルを SCP(a Simple Communication Protocol) と呼ぶ．

通信信頼性： SCP が満たすべき性質の 1 つは以下のとおりである．受信者が n 個のパケットを受け取ったとき，n 個のパケットは送信者により送られた最初の n 個のパケットで，受け取った順序と送られた順序は完全に一致する．つまり，受信者がパケットを受け取るときは，過不足無く，順序も保存している，ということである．この性質をここでは通信信頼性と呼ぶ．以降で，SCP がこの性質を満たすことを CafeOBJ [2] [3] で検証する．検証にさきがけ，SCP の CafeOBJ 仕様を作成する．

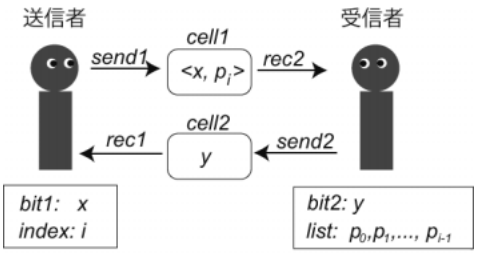

図 1 SCP 通信プロトコルの模式図

3 SCP の CafeOBJ 仕様の作成

SCP のモデルを観測遷移システム [4] [3] [5] として作成し，観測遷移システムの CafeOBJ 仕様を作成する．

3.1 SCP のモデルの作成

SCP のモデル化： SCP のモデルとして観測遷移システム $\mathcal{S}_{\mathrm{SCP}}$ を作成する．$\mathcal{S}_{\mathrm{SCP}}$ は，観測関数の集合 $\mathcal{O}_{\mathrm{SCP}}$，初期状態の集合 $\mathcal{I}_{\mathrm{SCP}}$ それに遷移関数の集合 $\mathcal{T}_{\mathrm{SCP}}$ から成る．$\mathcal{O}_{\mathrm{SCP}}$，$\mathcal{I}_{\mathrm{SCP}}$ および $\mathcal{T}_{\mathrm{SCP}}$ は以下のとおりである．

$$\begin{aligned}\mathcal{O}_{\mathrm{SCP}} \triangleq \{&\mathrm{bit1} : \mathrm{Sys} \to \mathrm{Bool}, \mathrm{bit2} : \mathrm{Sys} \to \mathrm{Bool},\\ &\mathrm{cell1} : \mathrm{Sys} \to \mathrm{PCell}, \mathrm{cell2} : \mathrm{Sys} \to \mathrm{BCell},\\ &\mathrm{index} : \mathrm{Sys} \to \mathrm{Nat}, \mathrm{list} : \mathrm{Sys} \to \mathrm{List}\}\end{aligned}$$

$$\begin{aligned}\mathcal{I}_{\mathrm{SCP}} \triangleq \{&s_0 \mid \mathrm{bit1}(s_0) = \mathrm{false} \wedge \mathrm{bit2}(s_0) = \mathrm{false} \wedge\\ &\mathrm{cell1}(s_0) = \mathrm{empty} \wedge \mathrm{cell2}(s_0) = \mathrm{empty} \wedge\\ &\mathrm{index}(s_0) = 0 \wedge \mathrm{list}(s_0) = \mathrm{nil}\}\end{aligned}$$

$$\begin{aligned}\mathcal{T}_{\mathrm{SCP}} \triangleq \{&\mathrm{send1} : \mathrm{Sys} \to \mathrm{Sys}, \mathrm{rec1} : \mathrm{Sys} \to \mathrm{Sys},\\ &\mathrm{send2} : \mathrm{Sys} \to \mathrm{Sys}, \mathrm{rec2} : \mathrm{Sys} \to \mathrm{Sys},\\ &\mathrm{drop1} : \mathrm{Sys} \to \mathrm{Sys}, \mathrm{drop2} : \mathrm{Sys} \to \mathrm{Sys}\}\end{aligned}$$

ここで，Sys，Bool，PCell，BCell，Nat および List

は，それぞれ，$\mathcal{S}_{\mathrm{SCP}}$ の状態，ブール値，自然数とブール値の組のセル，ブール値のセル，自然数および自然数のリストの型である．また，empty は空のセルを，nil は空のリストを表す．

観測関数： 観測関数 bit1，bit2，cell1，cell2 および list は，それぞれ，前節に現れる bit1，bit2，cell1，cell2 および list に対応する．観測関数 index は，パケットの識別番号 (p_i の i) に対応する．

遷移関数： 遷移関数 send1，rec1，send2 および rec2 は，それぞれ，前節に現れる send1，rec1，send2 および rec2 に対応する．drop1 および drop2 は，セルは完全に信頼できないことを表現するための遷移関数である．この 2 つは，2 つのセルに対して以下のことを繰り返し行う．

- drop1：cell1 が空でなければ，その中の組を取り出し，cell1 を空にする．
- drop2：cell2 が空でなければ，その中のビットを取り出し，cell2 を空にする．

$\mathcal{S}_{\mathrm{SCP}}$ 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，$\mathcal{S}_{\mathrm{SCP}}$ の CafeOBJ 仕様を記述するときに示す (3.3 小節参照)．

$\mathcal{S}_{\mathrm{SCP}}$ の CafeOBJ 仕様を作成する．その前に，データに関する型 PCell，BCell，Nat および List の CafeOBJ 仕様を作成する．

3.2 データ型の CafeOBJ 仕様の作成

まず，汎用のセル，リストそれに組のパラメータ化モジュールを作成する．

汎用のセルの仕様 CELL： 汎用のセルのパラメータ化モジュールは以下のとおりである．

```
mod! CELL(M :: EQTRIV) {
  [Cell]
  op empty : -> Cell
  op c : Elt.M -> Cell
  op get : Cell -> Elt.M
  op empty? : Cell -> Bool
  op _=_ : Cell Cell -> Bool {comm}
  var C : Cell  vars X Y : Elt.M
  eq get(c(X)) = X .
  eq empty?(empty) = true .
  eq empty?(c(X)) = false .
  eq (C = C) = true .
  eq (empty = c(X)) = false .
  eq (c(X) = c(Y)) = (X = Y) .
}
```

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}
}
```

定数 empty は空のセルを表し，演算 c は空でないセルの構成子である．演算 get，empty?および_=_は，それぞれ，空でないセルの中身を取り出す関数，与えられたセルが空であるか否かを判定する述語，および与えられた 2 つのセル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．

汎用のリストの仕様 LIST： 汎用のリストのパラメータ化モジュールは以下のとおりである．

```
mod! LIST(M :: EQTRIV) {
  [List]
  op nil : -> List
  op __ : Elt.M List -> List
  op hd : List -> Elt.M
  op tl : List -> List
  op _=_ : List List -> Bool {comm}
  vars L L1 L2 : List  vars X Y : Elt.M
  eq hd(X L) = X .  eq tl(X L) = L .
  eq (L = L) = true .
  eq (nil = X L) = false .
  eq (X L1 = Y L2) = (X = Y and L1 = L2) .
}
```

定数 nil は空のリストを表し，演算__は空でないリストの構成子である．演算 hd，tl および_=_は，それぞれ，空でないリストの先頭を返す関数，空でないリストの先頭を取り除いた残りを返す関数，および与えられた 2 つのリスト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．

汎用の組の仕様 PAIR： 汎用の組のパラメータ化モジュールは以下のとおりである．

```
mod! PAIR(M :: EQTRIV,N :: EQTRIV) {
  [Pair]
  op <_,_> : Elt.M Elt.N -> Pair
  op fst : Pair -> Elt.M
  op snd : Pair -> Elt.N
  op _=_ : Pair Pair -> Bool {comm}
  var P : Pair
  vars X1 X2 : Elt.M  vars Y1 Y2 : Elt.N
  eq fst(< X1,Y1 >) = X1 .
  eq snd(< X1,Y1 >) = Y1 .
  eq (P = P) = true .
  eq (< X1,Y1 > = < X2,Y2 >)
     = (X1 = X2) and (Y1 = Y2) .
}
```

演算<_,_>は組の構成子である．演算 fst，snd および_=_は，それぞれ，組の最初の要素を返す関数，組の 2 番目の要素を返す関数，および与えられた 2 つの組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．

自然数の仕様 PNAT： つづいて自然数のモジュールを作成する．

```
mod! PNAT {
  [Nat]
  op 0 : -> Nat
  op s : Nat -> Nat
  op _=_ : Nat Nat -> Bool {comm}
  vars X Y : Nat
  eq (X = X) = true .
  eq (0 = s(X)) = false .
  eq (s(X) = s(Y)) = (X = Y) .
}
```

定数 0 はゼロを表し，演算 s は非ゼロの自然数の構成子である．項 s(n) は，自然数 n より 1 大きい自然数 $n+1$ を表す．このため，演算 s は後者関数 (successor function) と呼ばれる．演算_=_は，与えられた 2 つの自然数が等しいか否かを判定する述語であり，その定義は等式で与えられている．

等価判定を持つブール値の仕様 EQBOOL： ブール値としては組込みモジュール BOOL を用いるが，2 つのブール値が等しいか否かを判定する述語_=_を用いたいので，以下のモジュールを作成する．

```
mod! EQBOOL {
  op _=_ : Bool Bool -> Bool {comm}
  var B : Bool
  eq (B = B) = true .
  eq (true = false) = false .
}
```

モジュール BOOL はデフォルトで輸入されるので，明示的に輸入 (pr(BOOL)) する必要はない．

自然数のリストの仕様 PNAT-LIST： 汎用のリストから自然数のリストを具現化するためのビューを作成する．

```
view EQTRIV2PNAT from EQTRIV to PNAT {
  sort Elt -> Nat,
  op _=_ -> _=_
}
```

LIST(M <= EQTRIV2PNAT) で，自然数のリストのモジュールを作成できるが，与えられた自然数 n に対して，n から 0 まで (先頭が n で最後が 0) のリストを作成する演算 mk を用いたいので，以下のモジュールを作成する．

```
mod! PNAT-LIST {
  pr(LIST(M <= EQTRIV2PNAT))
  op mk : Nat -> List
  var X : Nat
  eq mk(0) = 0 nil .
  eq mk(s(X)) = s(X) mk(X) .
}
```

ブール値のセルの仕様： 汎用のセルからブール値のセルを具現化するためのビューを作成する．

```
view EQTRIV2EQBOOL from EQTRIV to EQBOOL {
  sort Elt -> Bool,
  op _=_ -> _=_
}
```

ブール値のセルのモジュールは以下のとおりに作成できる．

```
CELL(M <= EQTRIV2EQBOOL)
  *{sort Cell -> BCell}
```

ソート名 Cell を BCell に変更している．

ブール値と自然数の組のセルの仕様： 汎用のセルからブール値と自然数の組のセルを具現化するが，その前に，汎用の組からブール値と自然数の組を具現化する．汎用の組からブール値と自然数の組を具現化するには 2 つのビューを用いる．それらは，EQTRIV2EQBOOL と EQTRIV2PNAT である．これらを用いると，ブール値と自然数の組のモジュールは以下のとおりに作成できる．

```
mod! BOOL-PNAT-PAIR {
  pr(PAIR(M <= EQTRIV2EQBOOL,
          N <= EQTRIV2PNAT)
       *{sort Pair -> BNPair})
}
```

汎用のセルからブール値と自然数の組のセルを具現化するとき，このモジュールを参照する必要があるので，名前 BOOL-PNAT-PAIR をつけて作成した．名前変えを用いて，ソート名 Pair を BNPair に変更している．

汎用のセルからブール値と自然数の組のセルを具現化するためのビューは以下のとおりである．

```
view EQTRIV2BOOL-PNAT-PAIR
     from EQTRIV to BOOL-PNAT-PAIR {
  sort Elt -> BNPair,
  op _=_ -> _=_
}
```

ブール値と自然数の組のセルのモジュールは以下のとおりに作成できる．

```
CELL(M <= EQTRIV2BOOL-PNAT-PAIR)
  *{sort Cell -> PCell}
```

ソート名 Cell を PCell に変更している．

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 $\mathcal{S}_{\mathrm{SCP}}$ で用いるデータ型の CafeOBJ 仕様を作成したので，つづいて $\mathcal{S}_{\mathrm{SCP}}$ の CafeOBJ 仕様を作成する．

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

```
mod* SCP {
  pr(CELL(M <= EQTRIV2EQBOOL)
       *{sort Cell -> BCell})
  pr(CELL(M <= EQTRIV2BOOL-PNAT-PAIR)
       *{sort Cell -> PCell})
  pr(PNAT-LIST)
  *[Sys]*
  op init : -> Sys
  bop cell1 : Sys -> PCell
  bop cell2 : Sys -> BCell
  bop bit1  : Sys -> Bool
  bop bit2  : Sys -> Bool
  bop index : Sys -> Nat
  bop list  : Sys -> List
  bop send1 : Sys -> Sys
  bop rec1  : Sys -> Sys
  bop send2 : Sys -> Sys
  bop rec2  : Sys -> Sys
  bop drop1 : Sys -> Sys
  bop drop2 : Sys -> Sys
  var S : Sys
  ...
}
```

定数 init は $\mathcal{S}_{\mathrm{SCP}}$ 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，$\mathcal{S}_{\mathrm{SCP}}$ の観測関数と遷移関数に対応する．... の箇所に，$\mathcal{S}_{\mathrm{SCP}}$ 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 cell1(init) = empty .
  eq cell2(init) = empty .
  eq bit1(init)  = false .
  eq bit2(init)  = false .
  eq index(init) = 0 .
  eq list(init)  = nil .
```

これらの等式は $\mathcal{I}_{\mathrm{SCP}}$ 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 cell1(send1(S))
```

```
    = c(< bit1(S),index(S) >) .
  eq cell2(send1(S)) = cell2(S) .
  eq bit1(send1(S))  = bit1(S) .
  eq bit2(send1(S))  = bit2(S) .
  eq index(send1(S)) = index(S) .
  eq list(send1(S))  = list(S) .
```

遷移関数 rec1 の定義： 遷移関数 rec1 を定義する等式は以下のとおりである.

```
  op c-rec1 : Sys -> Bool
  eq c-rec1(S) = not empty?(cell2(S)) .
  eq  cell1(rec1(S)) = cell1(S) .
  ceq cell2(rec1(S)) = empty if c-rec1(S) .
  ceq bit1(rec1(S))
      = (if bit1(S) = get(cell2(S))
        then bit1(S) else get(cell2(S)) fi)
     if c-rec1(S) .
  eq  bit2(rec1(S))  = bit2(S) .
  ceq index(rec1(S))
      = (if bit1(S) = get(cell2(S))
        then index(S) else s(index(S)) fi)
      if c-rec1(S) .
  eq  list(rec1(S))  = list(S) .
  ceq rec1(S)        = S if not c-rec1(S) .
```

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.

```
  eq cell1(send2(S)) = cell1(S) .
  eq cell2(send2(S)) = c(bit2(S)) .
  eq bit1(send2(S))  = bit1(S) .
  eq bit2(send2(S))  = bit2(S) .
  eq index(send2(S)) = index(S) .
  eq list(send2(S))  = list(S) .
```

遷移関数 rec2 の定義： 遷移関数 rec2 を定義する等式は以下のとおりである.

```
  op c-rec2 : Sys -> Bool
  eq c-rec2(S) = not empty?(cell1(S)) .
  ceq cell1(rec2(S)) = empty if c-rec2(S) .
  eq  cell2(rec2(S)) = cell2(S) .
  eq  bit1(rec2(S))  = bit1(S) .
  ceq bit2(rec2(S))
      = (if bit2(S) = fst(get(cell1(S)))
        then not(bit2(S)) else bit2(S) fi)
     if c-rec2(S) .
  eq  index(rec2(S)) = index(S) .
  ceq list(rec2(S))
      = (if bit2(S) = fst(get(cell1(S)))
        then (snd(get(cell1(S))) list(S))
        else list(S) fi) if c-rec2(S) .
  ceq rec2(S)        = S if not c-rec2(S) .
```

遷移関数 drop1 の定義： 遷移関数 drop1 を定義する等式は以下のとおりである.

```
  op c-drop1 : Sys -> Bool
  eq c-drop1(S) = not empty?(cell1(S)) .
  ceq cell1(drop1(S))
      = empty if c-drop1(S) .
  eq  cell2(drop1(S)) = cell2(S) .
  eq  bit1(drop1(S))  = bit1(S) .
  eq  bit2(drop1(S))  = bit2(S) .
  eq  index(drop1(S)) = index(S) .
  eq  list(drop1(S))  = list(S) .
  ceq drop1(S)
      = S if not c-drop1(S) .
```

遷移関数 drop2 の定義： 遷移関数 drop2 を定義する等式は以下のとおりである.

```
  op c-drop2 : Sys -> Bool
  eq c-drop2(S) = not empty?(cell2(S)) .
  eq  cell1(drop2(S)) = cell1(S) .
  ceq cell2(drop2(S))
      = empty if c-drop2(S) .
  eq  bit1(drop2(S))  = bit1(S) .
  eq  bit2(drop2(S))  = bit2(S) .
  eq  index(drop2(S)) = index(S) .
  eq  list(drop2(S))  = list(S) .
  ceq drop2(S)
      = S if not c-drop2(S) .
```

4 SCP の通信信頼性に関する検証

通信信頼性を定式化し，SCP が通信信頼性を満たしていることを検証する.

4.1 通信信頼性の定式化

受信者が受け取ったパケットは受信者のリスト list に蓄えられる．このため，このリストを用いて通信信頼性を定式化できる．状態述語 rc を以下のとおりに定義する．

$$\begin{aligned}\mathrm{rc}(s) \triangleq\ & (\mathrm{bit1}(s) = \mathrm{bit2}(s)\ \Rightarrow \\ & (\mathrm{index}(s)\ \mathrm{list}(s)) = \mathrm{mk}(\mathrm{index}(s))) \wedge \\ & (\mathrm{bit1}(s) \neq \mathrm{bit2}(s) \Rightarrow \mathrm{list}(s) = \mathrm{mk}(\mathrm{index}(s)))\end{aligned}$$

すると，通信信頼性は，$\forall s : \mathcal{R}_{\mathcal{S}_{\mathrm{SCP}}}.\mathrm{rc}(s)$ として定式化できる．ここで，$\mathcal{R}_{\mathcal{S}_{\mathrm{SCP}}}$ は，$\mathcal{S}_{\mathrm{SCP}}$ の到達可能状態すべての集合である．つまり，通信信頼性とは，状態述語 rc が $\mathcal{S}_{\mathrm{SCP}}$ の不変性である，ということである．このことは，状態 s までに，受信者が受け取ったパケット $\mathrm{list}(s)$ は，送信者が 0 番目から $\mathrm{index}(s)$ あるいは $\mathrm{index}(s) - 1$ 番目までに送ったパケットであり，受け取った順番と送った順番は一致する，ということを表す．つまり，受信者がパケットを受け取るときは，過不足無く，順序も保存している，ということである．

SCP が通信信頼性を満たすことの検証は，$\forall s : \mathcal{R}_{\mathcal{S}_{\mathrm{SCP}}}.\mathrm{rc}(s)$ を証明することである．以降で，CafeOBJ による $\forall s : \mathcal{R}_{\mathcal{S}_{\mathrm{SCP}}}.\mathrm{rc}(s)$ の証明について記述する．

4.2 通信信頼性の検証

まず以下のモジュールを宣言する．

```
mod INV { pr(SCP)
  op inv1 : Sys -> Bool
  var S : Sys
  eq inv1(S)
     = (bit1(S) = bit2(S) implies
        (index(S) list(S)) = mk(index(S)))
       and
       (not(bit1(S) = bit2(S)) implies
        list(S) = mk(index(S))) .
}
```

演算 `inv1` は状態述語 rc に対応する．

$\forall s : \mathcal{R}_{\mathcal{S}_{\mathrm{SCP}}}.\mathrm{rc}(s)$ の証明は，$\mathcal{R}_{\mathcal{S}_{\mathrm{SCP}}}$ が初期状態を表す定数および遷移関数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 2 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，$\forall s : \mathcal{R}_{\mathcal{S}_{\mathrm{SCP}}}.\mathrm{rc}(s)$ の証明を行う．

```
{[1-init],
[1-send1]*, [1-rec1]*, [1-send2]*,
[1-rec2]*, [1-drop1]*, [1-drop2]*}
implies [inv1]*
```

図 2 帰納スキーマ 1

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 : -> Bool
  eq istep1 = inv1(s) implies inv1(s') .
}
```

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')` は各帰納段階で証明すべき論理式である．項 `inv1(s)` 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．

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) .
close
```

この証明節に対し，CafeOBJ は `true` を返す．

遷移関数 `rec1` に関する帰納段階： 遷移関数 `rec1` に関する帰納段階 ([1-rec1]*) 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 2 つの項に基づいて，前者の証明節を以下のとおり 3 つに分割する．

- `bit1(s) = get(cell2(s))`

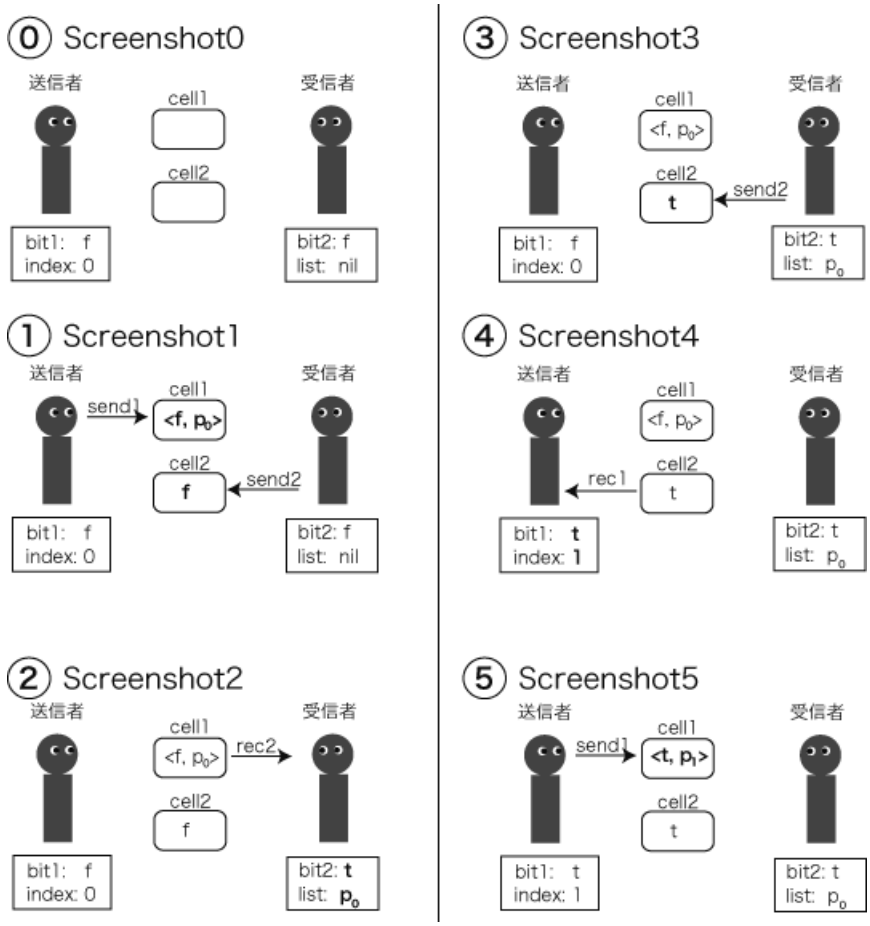

図 3　SCP の模擬実験

- `bit2(s) = get(cell2(s))`

分割された 3 つの場合は以下のとおりである．

1. `bit1(s) = get(cell2(s))`
2. `not(bit1(s) = get(cell2(s)))`, `bit2(s) = get(cell2(s))`
3. `not(bit1(s) = get(cell2(s)))`, `not(bit2(s) = get(cell2(s)))`

1 番目と 2 番目のいずれの証明節に対しても，CafeOBJ は `true` を返すが，3 番目の証明節に対しては，`true` でも `false` でもない `Bool` の項を返す．

場合分けの基準の選択理由： 2 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．

- `bit1(s) = get(cell2(s))`：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 `true` と等しいとき (1 番目の証明節に対し)，CafeOBJ が `true` を返すからである．
- `bit2(s) = get(cell2(s))`：この項が簡約結果に現れることと，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．

模擬実験よる SCP の理解と補題発見 (inv2)： 場合分けにより，3 番目の証明節を分割することも可能であるが，ここでは，この証明節に対し CafeOBJ が `true` を返すような補題を，補題発見法 1(本チュートリアルの第 4 回を参照) により類推する．このため，簡単な模擬実験により SCP への理解を深めることを試みる．

図 3 に SCP の模擬実験を示す．セルに何かが描か

れている場合，セルにはそれが存在しているか，あるいは空であるかのいずれかであることを意味する．Snapshot0 は初期状態である．送信者と受信者が，それぞれ，組 $\langle \mathrm{f}, \mathrm{p}_0 \rangle$ とブール値 f を cell1 と cell2 に入れることで (何度入れても構わない)，Snapshot1 の状態になる．受信者が cell1 から組 $\langle \mathrm{f}, \mathrm{p}_0 \rangle$ を取り出すことができれば，その最初の要素 f と bit2 は等しいため，Snapshot2 の状態になる．この間，送信者が cell2 からブール値 f を取り出しても，その値は bit1 と等しいため，本質的な変化は起こらない．つづいて，送信者と受信者が，それぞれ，組 $\langle \mathrm{f}, \mathrm{p}_0 \rangle$ とブール値 t を cell1 と cell2 に入れることで (何度入れても構わない)，Snapshot3 の状態になる．ここで，送信者が cell2 からブール値 t を取り出すことができれば，その値は bit1 と異なるため，Snapshot4 の状態になる．この間，受信者が cell1 から $\langle \mathrm{f}, \mathrm{p}_0 \rangle$ を取り出しても，その最初の要素 f は bit1 と異なるため，本質的な変化は起こらない．それから，送信者と受信者が，それぞれ，組 $\langle \mathrm{t}, \mathrm{p}_1 \rangle$ とブール値 t を cell1 と cell2 に入れることで (何度入れても構わない)，Snapshot5 の状態になる．Snapshot1 と Snapshot5 は，bit1，bit2，cell1 の組の最初の要素，それに cell2 のブール値がいずれも等しいという側面において，同じである．また，Snapshot0 は，Snapshot1 の特殊形である．このため，bit1，bit2，cell1 の組の最初の要素，それに cell2 のブール値が等しいか否かという側面においては，Snapshot1 から Snapshot5 までの 5 つを考えればよいことがわかる．

この模擬実験から，cell2 が空でなければ，そこにあるブール値は，bit1 と bit2 のいずれかとは必ず等しいことがわかる．このことは 3 番目の仮定と矛盾している．つまり，3 番目の仮定は起こり得ない状態 (到達不可能状態) を表している．この観察により，以下の補題を推測できる．

```
op inv2 : Sys -> Bool
eq inv2(S) = (empty?(cell2(S)) or
              bit1(S) = get(cell2(S)) or
              bit2(S) = get(cell2(S))) .
```

これらをモジュール INV に追加する．

この補題を用いることで，3 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 3 番目の証明節は以下のとおりである．

```
open ISTEP
  eq empty?(cell2(s)) = false .
  eq (bit1(s) = get(cell2(s))) = false .
  eq (bit2(s) = get(cell2(s))) = false .
  eq s' = rec1(s) .
  red inv2(s) implies istep1 .
close
```

遷移関数 rec2 に関する帰納段階： 次に，遷移関数 rec2 に関する帰納段階 ([1-rec2]∗) 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 3 つの項に基づいて，前者の証明節を 4 つに分割する．

- bit2(s) = fst(get(cell1(s)))
- bit1(s) = fst(get(cell1(s)))
- index(s) = snd(get(cell1(s)))

分割された 4 つの場合は以下のとおりである．

1. bit2(s) = fst(get(cell1(s))),
   bit1(s) = fst(get(cell1(s))),
   index(s) = snd(get(cell1(s)))
2. bit2(s) = fst(get(cell1(s))),
   bit1(s) = fst(get(cell1(s))),
   not(index(s) = snd(get(cell1(s))))
3. bit2(s) = fst(get(cell1(s))),
   not(bit1(s) = fst(get(cell1(s))))
4. not(bit2(s) = fst(get(cell1(s))))

4 番目の証明節に対しては，CafeOBJ は true を返すが，残りのいずれの証明節に対しては，true でも false でもない Bool の項を返す．

場合分けの基準の選択理由： 3 つの項を場合分けの基準に選ぶ理由は以下のとおりである．

- bit2(s) = fst(get(cell1(s)))：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 と等しいとき (4 番目の証明節に対し)，CafeOBJ が true を返すから

である．

- `bit1(s) = fst(get(cell1(s)))`：理由の 1 つはこの項が簡約結果に現れることである．ただし，この項をもとに場合分けして得られるいずれの証明節に対しても，CafeOBJ は `true` でも `false` でもない `Bool` の項を返す．このため，この項は場合分けの基準として適切であるとはいえない．しかし，後述するように，図 3 に示す模擬実験より，この項が `false` と等しいときの証明節 (3 番目の証明節) における仮定は，起こり得ない状態 (到達不能状態) を表していることがわかる．この観察より補題を推測でき，3 番目の証明節に対し，CafeOBJ が `true` を返すようにすることができる．このことがこの項を場合分けの基準として選ぶ大きな理由である．
- `index(s) = snd(get(cell1(s)))`：理由の 1 つは，項 `index(s)` と項 `snd(get(cell1(s)))` は等しくなる可能性もそうでない可能性をもあるからである．この項も，1 つ前の項と同じ理由で，場合分けの基準として適切であるとはいえない．しかし，1 つ前の項と同じように，この項が `false` に等しいとき，対応する証明節 (2 番目の証明節) に対し，CafeOBJ が `true` を返すことのできる補題を推測することができる．さらに，`true` に等しいとき，対応する証明節 (1 番目の証明節) に対し，CafeOBJ が `true` を返すことのできるブール値に関する補題を推測することができる．このことがこの項を場合分けの基準として選ぶ大きな理由である．

ブール値の補題発見 (`eqbool-lemma1`)： 1 番目の証明節に対して CafeOBJ が返す項は，以下を部分項として含む．

```
fst(get(cell1(s)))
  = (fst(get(cell1(s))) xor true)
```

この項は `false` と等価であることが類推でき，もしそうであれば，1 番目の証明節に対し，CafeOBJ は `true` を返す．そこで，ブール値に関する以下の補題を推測する．

```
op eqbool-lemma1 : Bool -> Bool .
eq eqbool-lemma1(B) = not((not B) = B) .
```

これらをモジュール `EQBOOL` に追加する．

この補題を用いることで，1 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 1 番目の証明節は以下のとおりである．

```
open ISTEP
  eq empty?(cell1(s)) = false .
  eq bit2(s) = fst(get(cell1(s))) .
  eq bit1(s) = fst(get(cell1(s))) .
  eq index(s) = snd(get(cell1(s))) .
  eq s' = rec2(s) .
  red eqbool-lemma1(fst(get(cell1(s))))
      implies istep1 .
close
```

補題発見 (`inv3`)： SCP の模擬実験 (図 3 参照) から，cell1 が空でない場合，bit2 と cell1 の組の最初の要素が等しいのは，Snapshot1 だけである．このとき，cell1 の組の 2 番目の要素のパケットの識別番号は，送信者が送ろうとしているパケットのものと同じである．これは，2 番目の仮定と矛盾している．つまり，2 番目の仮定は起こり得ない状態 (到達不可能状態) を表している．この観察により，以下の補題を推測できる．

```
op inv3 : Sys -> Bool
eq inv3(S) = (not empty?(cell1(S)) and
       bit2(S) = fst(get(cell1(S)))
       implies
       index(S) = snd(get(cell1(S)))) .
```

これらをモジュール `INV` に追加する．

この補題を用いることで，2 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 2 番目の証明節は以下のとおりである．

```
open ISTEP
  eq empty?(cell1(s)) = false .
  eq bit2(s) = fst(get(cell1(s))) .
  eq bit1(s) = fst(get(cell1(s))) .
  eq (index(s) = snd(get(cell1(s))))
     = false .
  eq s' = rec2(s) .
  red inv3(s) implies istep1 .
close
```

補題発見 (inv4)：SCP の模擬実験の観察から，cell1 が空でない場合，bit2 と cell1 の組の最初の要素が等しいのは，Snapshot1 だけであることがわかっている．このとき，bit1 も cell1 の組の最初の要素と等しいことがわかる．これは，3 番目の仮定と矛盾している．つまり，3 番目の仮定は起こり得ない状態 (到達不可能状態) を表している．この観察により，以下の補題を推測できる．

```
op inv4 : Sys -> Bool
eq inv4(S) = (not empty?(cell1(S)) and
       bit2(S) = fst(get(cell1(S)))
       implies
       bit1(S) = fst(get(cell1(S)))) .
```

これらをモジュール INV に追加する．

この補題を用いることで，3 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 3 番目の証明節は以下のとおりである．

```
open ISTEP
  eq empty?(cell1(s)) = false .
  eq bit2(s) = fst(get(cell1(s))) .
  eq (bit1(s) = fst(get(cell1(s))))
     = false .
  eq s' = rec2(s) .
  red inv4(s) implies istep1 .
close
```

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： 3 つの補題 inv2，inv3 と inv4 を導入したため，帰納スキーマ IS1 の代わりに，図 4 のように記述される連立帰納スキーマ (SIS4 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，inv3 の証明譜は，さらにつづく 7 つの言明に，そして inv4 の証明譜は，最後の 7 つの言明に対応している．これら 4 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 $\mathcal{S}_{\mathrm{SCP}}$ にとって不変であるのみならず，inv2，inv3 それに inv4 もそうであることが証明されたことになる．

```
{[1-init],
 [1-4-send1]*, [1-4-rec1]*, [1-4-send2]*,
 [1-4-rec2]*, [1-4-drop1]*, [1-4-drop2]*,
 [2-init],
 [2-4-send1]*, [2-4-rec1]*, [2-4-send2]*,
 [2-4-rec2]*, [2-4-drop1]*, [2-4-drop2]*,
 [3-init],
 [3-4-send1]*, [3-4-rec1]*, [3-4-send2]*,
 [3-4-rec2]*, [3-4-drop1]*, [3-4-drop2]*,
 [4-init],
 [4-4-send1]*, [4-4-rec1]*, [4-4-send2]*,
 [4-4-rec2]*, [4-4-drop1]*, [4-4-drop2]*}
implies {[inv1]*, [inv2]*, [inv3]*, [inv4]*}
```

図 4 連立帰納スキーマ 4

IS1 から SIS4 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 SIS4 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．SIS4 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，SIS4 でもできることになる．

inv2，inv3 それに inv4 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 : -> Bool
op istep3 : -> Bool
op istep4 : -> Bool
eq istep2 = inv2(s) implies inv2(s') .
eq istep3 = inv3(s) implies inv3(s') .
eq istep4 = inv4(s) implies inv4(s') .
```

場合分けと補題発見のどちらを試みるかについて： 帰納段階の最初のうちは，場合分けを先に試みたほうがよいことのほうが多い．実際に，著者らが帰納段階の証明譜 (節) を作成するときは，対応する遷移関数が効力条件を持てば (常に成り立つわけでなければ)，まず，効力条件に基づき場合分けを行っている．さらに，効力条件が成り立たない場合に CafeOBJ が true でも false でもない Bool の項を返せば，さら

に場合分けを行っている．しかし，ある証明節に対し，CafeOBJ が true でも false でもない Bool の項を返すとき，場合分けと補題発見のどちらを先に試みるべきかについては，対象であるシステムやプロトコルの理解度に依存する．たとえば，inv1 の遷移関数 rec1 に関する帰納段階の 3 番目の証明節に対しては，SCP のことをある程度理解していれば，その証明節に対応する仮定は起こり得ない状態 (到達不能状態) を表していることがわかる．このことから補題を推測できる．しかし，仮定が起こり得ない状態を表していることがわからなければ，適切な補題を発見することができないかもしれない．このときは，場合分けを試みた後で (それにより SCP への理解も深まるので) 補題発見を試みたほうがよいかもしれない．

ある証明節に対し，CafeOBJ が true でも false でもない Bool の項を返すとき，場合分けと補題発見のどちらを先に試みるべきかに関するおおまかな指針は以下のとおりである．

場合分けと補題発見の指針 ある証明節に対し，CafeOBJ は true でも false でもない Bool の項を返すとする．この証明節に宣言されている場合を特徴づける等式の集合 (仮定) が起こり得ない状態 (到達不能状態) を表していることが類推できれば，その等式の集合をもとに補題を推測する．類推できなければ，場合分けを行う．

CafeOBJ が false を返すとき，証明が成功するとすれば，証明節に宣言されている場合を特徴づける等式の集合 (仮定) は起こり得ない状態を表している．このため，補題発見法 1 により補題を類推することができる．同じく，上記指針の補題も，補題発見法 1 により類推できる．また，補題発見法 1 と補題発見法 2 の組み合わせ (前回参照) も用いることができる．

場合を特徴づける等式の集合 (仮定) が起こり得ない状態だけを表していないときにも，補題を類推することができることがある．本チュートリアルの第 1 回と第 4 回で解説した，相互排除プロトコル Qlock が相互排除性を満たすことの検証における，補題 inv2 の発見はその一例である．その補題を発見したときの場合を特徴づける等式の集合は以下のとおりであった．

```
{pc(s,k) = wt, top(queue(s)) = k,
 i = k, (j = k) = false}
```

この等式の集合は起こり得ない状態だけを表していない．ただし，プロセス j の位置が cs であれば，つまり pc(s,j) = cs であれば，この等式の集合は起こり得ない状態を表すことになる．Qlock の検証における補題 inv2 の発見はこのことに基づいている．つまり，場合を特徴づける等式の集合が起こり得ない状態だけを表していなくても，さらなる仮定をすることで，起こり得ない状態を表すことを類推できれば，その等式の集合とさらなる仮定 (別の等式の集合) をもとに補題を推測することができる．これにより，場合分けと補題発見に関する指針を一般化できる．

場合分けと補題発見の指針の一般化 ある証明節に対し，CafeOBJ は true でも false でもない Bool の項を返すとする．この証明節に宣言されている場合を特徴づける等式の集合 A(仮定) が，別の等式の集合 B(さらなる仮定) を加えることで，起こり得ない状態 (到達不能状態) を表していることが類推できれば，等式の集合 $A \cup B$ をもとに補題を推測する．類推できなければ，場合分けを行う．

別の等式の集合 B が空のとき，この一般化は，先に記載した場合分けと補題発見に関する指針と一致する．

4.2.2 inv3 の証明譜

次に，inv3 の証明譜 (の一部) について記述する．

```
op istep3 : -> Bool
eq istep3 = inv3(s) implies inv3(s') .
```

遷移関数 rec1 に関する帰納段階： 遷移関数 rec1 に関する帰納段階 ([3-4-rec1]∗) 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 2 つの項に基づいて，前者の証明節を以下のとおり 3 つに分割する．

- bit1(s) = get(cell2(s))
- bit2(s) = fst(get(cell1(s)))

分割された 3 つの場合は以下のとおりである．

```
1. bit1(s) = get(cell2(s))
2. not(bit1(s) = get(cell2(s))),
   bit2(s) = fst(get(cell1(s)))
3. not(bit1(s) = get(cell2(s))),
   not(bit2(s) = fst(get(cell1(s))))
```

1 番目と 3 番目のいずれの証明節に対しても，CafeOBJ は `true` を返すが，2 番目の証明節に対しては，`true` でも `false` でもない `Bool` の項を返す．2 つの項を場合分けの基準に選ぶ理由は，`inv1` の証明譜の遷移関数 `rec1` に関する帰納段階のときと同じである．

補題発見 (`inv5`)：SCP の模擬実験の観察から，cell1 が空でない場合，bit2 と cell1 の組の最初の要素が等しいのは，Snapshot1 だけであることがわかっている．このとき，cell2 が空でなければ，bit1 と cell2 のブール値は等しいことがわかる．もし，cell2 が空でなければ，このことは 2 番目の仮定と矛盾しており，2 番目の仮定は起こり得ない状態 (到達不能状態) を表している．この観察より，以下の補題を推測できる．

```
op inv5 : Sys -> Bool
eq inv5(S) = (empty?(cell1(S)) or
     empty?(cell2(S)) or
     bit1(S) = get(cell2(S)) or
     not(bit2(S) = fst(get(cell1(S))))) .
```

これらをモジュール `INV` に追加する．

この補題を用いることで，2 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 2 番目の証明節は以下のとおりである．

```
open ISTEP
  eq empty?(cell2(s)) = false .
  eq (bit1(s) = get(cell2(s))) = false .
  eq bit2(s) = fst(get(cell1(s))) .
  eq s' = rec1(s) .
  red inv5(s) implies istep3 .
close
```

inv3 の証明譜の残り： 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．残りの 5 つの帰納段階の証明では，補題を必要としない．

```
{[1-init],
 [1-5-send1]*, [1-5-rec1]*, [1-send2-5]*,
 [1-5-rec2]*, [1-5-drop1]*, [1-5-drop2]*,
 [2-init],
 [2-5-send1]*, [2-5-rec1]*, [2-5-send2]*,
 [2-5-rec2]*, [2-5-drop1]*, [2-5-drop2]*,
 [3-init],
 [3-5-send1]*, [3-5-rec1]*, [3-5-send2]*,
 [3-5-rec2]*, [3-5-drop1]*, [3-5-drop2]*,
 [4-init],
 [4-5-send1]*, [4-5-rec1]*, [4-5-send2]*,
 [4-5-rec2]*, [4-5-drop1]*, [4-5-drop2]*,
 [5-init],
 [5-5-send1]*, [5-5-rec1]*, [5-5-send2]*,
 [5-5-rec2]*, [5-5-drop1]*, [5-5-drop2]*}
implies {[inv1]*, [inv2]*, [inv3]*, [inv4]*,
                                  [inv5]*}
```

図 5 連立帰納スキーマ 5

連立帰納スキーマの変更： 新たに補題 `inv5` を追加したので，連立帰納スキーマ SIS4 の代わりに，図 5 のように記述される連立帰納スキーマを用いる．

通信信頼性検証の残り： `inv2`，`inv4` それに `inv5` の証明譜も同様に作成できる．`inv2` の帰納段階では，`inv4` を，`inv4` の帰納段階では，`inv5` を，それに `inv5` の帰納段階では，`inv2` と `inv4` を補題として用いる．これ以上の補題は必要としない．

4.2.3 `eqbool-lemma1` の証明譜

SCP が通信信頼性を満たすことの検証を完了させるには，`inv1` の証明譜で用いたブール値に関する補題 `eqbool-lemma1` の証明をしなければならない．対応する証明譜は以下のとおりである．

```
open EQBOOL
  op b : -> Bool .
  eq b = true .
  red eqbool-lemma1(b) .
close
open EQBOOL
  op b : -> Bool .
  eq b = false .
```

```
  red eqbool-lemma1(b) .
close
```

いずれの証明節に対しても，CafeOBJ は `true` を返す．

以上で SCP が通信信頼性を満たすことの検証が終了したことになる．

5 おわりに

SCP 通信プロトコルが通信信頼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，以下のことについて解説した．

- 場合分けの基準の選択方法 (理由)
- 模擬実験による SCP プロトコルの理解と補題発見
- 場合分けと補題発見のどちらを試みるかについて
- 補題発見に伴う (連立) 帰納スキーマの変更
- データ型の補題

最初にも書いたが，今回が本チュートリアルの最終回である．これまでの 6 回のチュートリアルで解説したことを簡単に振り返ることで本チュートリアルのまとめとしたい．

1. 形式手法と CafeOBJ： 相互排除プロトコルのモデル化，形式仕様の作成，検証を例として，CafeOBJ システムを用いた形式手法がどのようなものであるかを総括的に解説した．
2. 構文と意味： CafeOBJ の構文と意味について解説し，それに基づき仕様の構造化，仕様の作成支援，仕様の検証などに関する基本事項について説明した．
3. 等式推論と項書換システム： CafeOBJ 仕様に関する最重要の推論機構である等式推論とその実行エンジンである項書換システムについて，CafeOBJ 仕様を作成するときに必要となる知識に焦点を当てて解説した．
4. 証明譜による検証法：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．
5. 認証プロトコルの検証：認証プロトコルの証明譜による検証について解説しつつ，関連する証明譜の作成技法も説明した．
6. 通信プロトコルの検証：通信プロトコルの証明譜による検証について解説しつつ，関連する証明譜の作成技法も説明した．

最後に，既に実践している読者もいると期待するが，是非，CafeOBJ Official Homepage[†1]を訪れ，CafeOBJ システムをダウンロードし，各自のパソコン等にインストールし，チュートリアルで取り上げた例を CafeOBJ システムで実行する，といったことを強く奨める．形式仕様の作成や検証の学習は，プログラミングのそれと基本的に同じである．実体験がすべてではないが，学習効果を飛躍的に向上させることは間違いない．

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] Bartett, K. A., Scantlebury, R. A. and Wilkinson, P. T.: A note on reliable full-duplex transmission over half-duplex links, *Communication of the ACM*, Vol. 12(1969), pp. 260–261.

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

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

[4] 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.

[5] 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.

†1 http://www.ldl.jaist.ac.jp/cafeobj/