Two Reduction Systems in Proof Scores Writing

Daniel GAINA
(joint work with Dorel LUCANU, Kazuhiro OGATA and Kokichi FUTATSUGI)

Japan Advanced Institute of Science and Technology

April 10, 2012

Overview I

We present our methodology

• modeling • specification • verification

through an example:

Alternating bit protocol

1. we our logical framework is sufficiently expressive to model dropping of elements in arbitrary positions of the communication channels
2. the semantics of $\_ = \_$ and $\_ \Rightarrow \_$ is that of equality; the reduction system consisting of rewriting rules is regarded as a second reduction system on top of the equational one; it has the advantage of preserving the termination property during the verification process

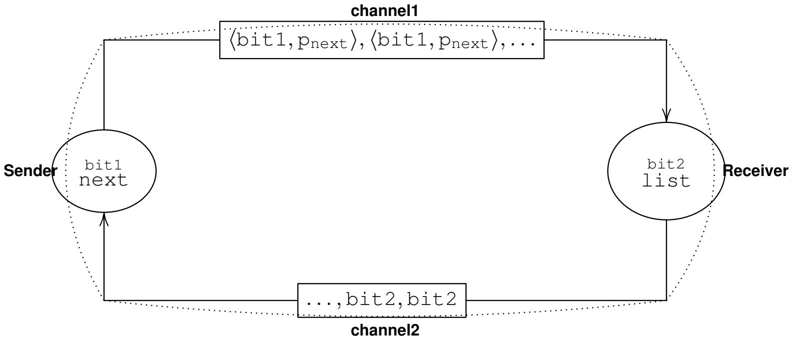
Overview II

- we use conditional equations with conditions executable by matching, which increase the specification operational expressivity; this allows to handle nondeterminism successfully at the operational level

3. an order of application of the proof rules is established (this represents the first step towards automation)
4. a proof rule has the following general form $\frac{SP_1 \vdash Prop_1 \ldots SP_n \vdash Prop_n}{SP \vdash Prop}$;
   In the verification process we identify clearly inconsistent specifications SP_i of the subgoals $SP_i \vdash Prop_i$ obtained by applying a proof rule to a specification SP

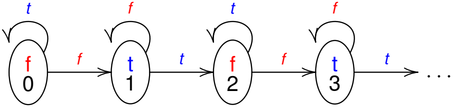
Alternating Bit Protocol

- Two agents, **Sender** and **Receiver** that do not share a common memory use two channels, `channel1` and `channel2` to communicate
  - Sender sends repeatedly pairs of **packets** and **bits**, $\langle \texttt{bit1}, \texttt{p}_\texttt{n} \rangle$, to Receiver over channel1
  - Receiver sends repeatedly `bit2` to Sender over `channel2`



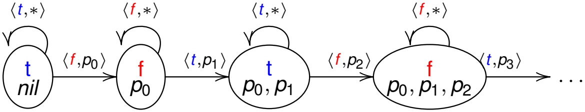
Sender's diagram

- When **Sender** gets `bit1` from the **Receiver** over `channel2`, it is a confirmation from the **Receiver** that the packet sent was received. In this case, **Sender** alternates `bit1` and selects the next packet for sending.
- Initially both channels are empty and the **Sender**'s bit is different from the **Receiver**'s bit

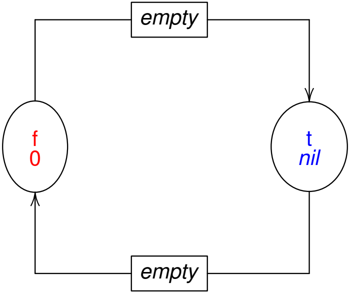


Receiver's diagram

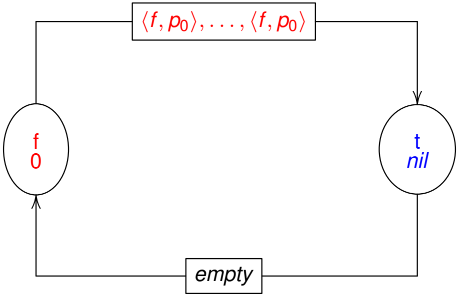
- When **Receiver** gets a pair `< b,p >` such that `b` is different from `bit2` it receives `p` and alternates `bit2`.



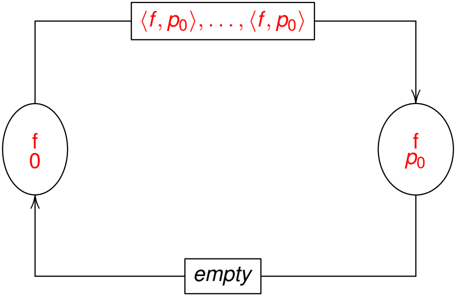
Snapshot I



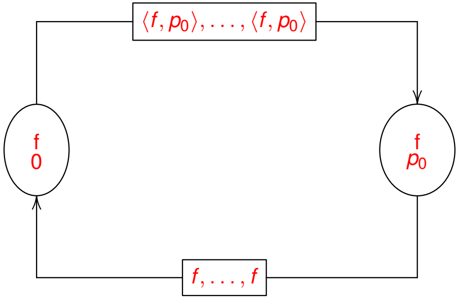
Snapshot II



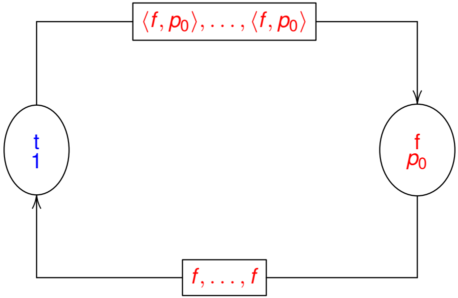
Snapshot III



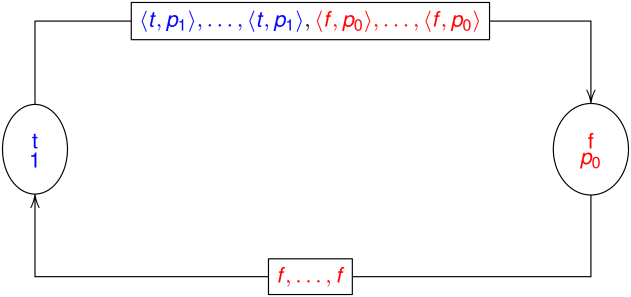
Snapshot IV



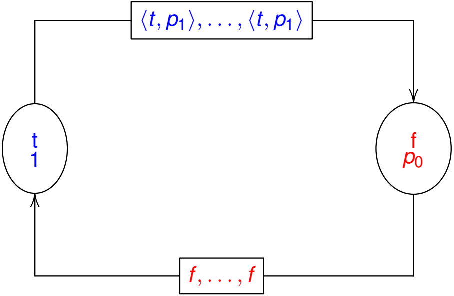
Snapshot V



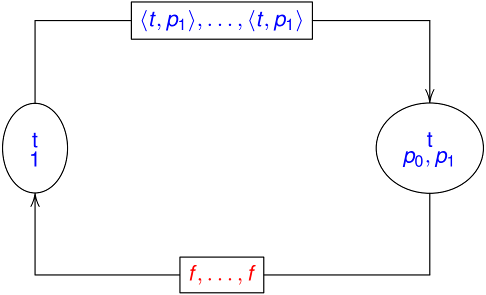
Snapshot VI


$\langle t, p_1\rangle, \ldots, \langle t, p_1\rangle, \langle f, p_0\rangle, \ldots, \langle f, p_0\rangle$
t
1
f
p_0
$f, \ldots, f$

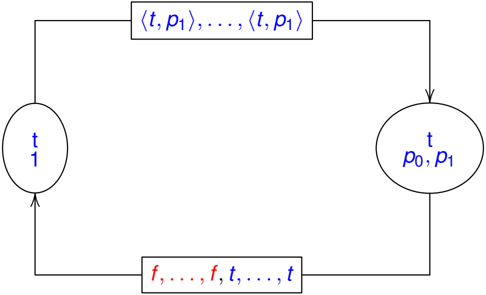
Snapshot VII



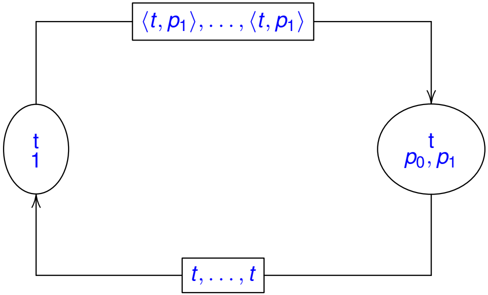
Snapshot VIII



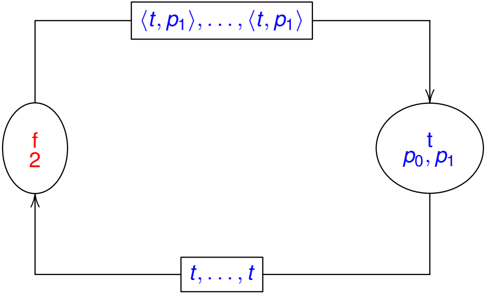
Snapshot IX


$\langle t, p_1 \rangle, \ldots, \langle t, p_1 \rangle$
t
1
t
p_0, p_1
$f, \ldots, f, t, \ldots, t$

Snapshot X



Snapshot XI



Safety Property

We assume that the communication channels are unreliable:
- data in the channels may be lost, but not changed or damaged.

Safety Property

- If Receiver receives the *n*th packet then
  - Receiver has received the n+1 packets $p_0, ..., p_n$ in this order,
  - each p_i for $i = \overline{0, n}$ has been received only once, and
  - no other packets have been received
- In this case study we check the above property

Data Types used

- the **Packets** are indexed by natural numbers: $\mathtt{pac}(0), \mathtt{pac}(\mathtt{s}0), \ldots, \mathtt{pac}(\mathtt{s}^n 0)$

  `op pac : Nat -> Packet [ctor]`

- the bits sent by **Sender** and **Receiver** have two values

  `op t : -> Bit [ctor]` and `op f : -> Bit[ctor]`

  The function `op not_: Bit -> Bit` alternates the bits

- The communication channels and the packets received by **Receiver** are modeled by sequences.
  1. **Channel1** consists of sequences of pairs of bits and packets

     $$\langle b_1, p_1 \rangle, \ldots, \langle b_n, p_n \rangle$$

  2. **Channel2** consists of sequences of bits

     $$b_1, \ldots, b_n$$

  3. **List** of packets received by **Receiver** consists of sequences of packets

     $$p_1, \ldots, p_n$$

ABP I

```
fth ABP is inc CHANNEL1 . inc CHANNEL2 . inc PACKET-LIST .
sort Sys .
— — — constructors — — —
  op init : -> Sys [ctor] .            — — — initial state
  op rec1 : Sys -> Sys [ctor] .        — — — Sender receives bits
  op rec2 : Sys -> Sys [ctor] .        — — — Receiver receives pairs of bits & packets
  op send1 : Sys -> Sys [ctor] .       — — — Sender sends pairs of bits & packets
  op send2 : Sys -> Sys [ctor] .       — — — Receiver sends bits
  op drop1 : Sys -> Sys [ctor] .       — — — dropping one element of channel1
  op drop2 : Sys -> Sys [ctor] .       — — — dropping one element of channel2
— — — observers — — —
  op channel1 : Sys -> Channel1 .      — — — Sender-to-Receiver channel
  op channel2 : Sys -> Channel2 .      — — — Receiver-to-Sender channel
  op bit1 : Sys -> Bit .               — — — Sender's bit
  op bit2 : Sys -> Bit .               — — — Receiver's bit
  op next : Sys -> Nat .               — — — number of packet sent next by Sender
  op list : Sys -> List .              — — — lists of packets received by Receiver
— — — underspecified functions — — —
  ops x1 y1 : Sys -> Channel1 .
  ops x2 y2 : Sys -> Channel2 .
```

ABP II

```
— variables — — —
var S : Sys . vars C1 C1' : Channel1 . vars C2 C2' : Channel2 .
var B : Bit . var P : Packet . var N : Nat .
— — — receive1 — — —
  eq channel1(rec1(S)) = channel1(S) .
  ceq [ch2-a] : channel2(rec1(S)) = channel2(S)    if empty = channel2(S) .
  ceq [ch2-b] : channel2(rec1(S)) = C2             if B,C2 := channel2(S) .
  ceq [bit1-a] : bit1(rec1(S)) = bit1(S)           if empty = channel2(S) .
  ceq [bit1-b] : bit1(rec1(S)) = bit1(S)           if B,C2 := channel2(S) ⋀ B = not bit1(S) .
  ceq [bit1-c] : bit1(rec1(S)) = not bit1(S)       if B,C2 := channel2(S) ⋀ B = bit1(S) .
  eq bit2(rec1(S)) = bit2(S) .
  ceq [next-a] : next(rec1(S)) = next(S)           if empty = channel2(S) .
  ceq [next-b] : next(rec1(S)) = next(S)           if B,C2 := channel2(S) ⋀ B = not bit1(S) .
  ceq [next-c] : next(rec1(S)) = s next(S)         if B,C2 := channel2(S) ⋀ B = bit1(S) .
  eq list(rec1(S)) = list(S) .
```

ABP III

```
— — — drop1 — — —
ceq [d1-a] : channel1(drop1(S)) = x1(S),y1(S) if x1(S),< B,P >,y1(S) := channel1(S) .
eq [d1-b] : channel1(drop1(S)) = channel1(S) [owise] .
eq channel2(drop1(S)) = channel2(S) .
eq bit1(drop1(S)) = bit1(S) .
eq bit2(drop1(S)) = bit2(S) .
eq next(drop1(S)) = next(S) .
eq list(drop1(S)) = list(S) .
```

Some specification aspects

1. ABP module is declared with loose semantics. Since the sort Sys has seven constructors, the carrier sets of the ABP models for the sort Sys consist of interpretations of constructor terms of the form $\sigma_n(\ldots\sigma_1(\mathtt{init}))$, where $\sigma_i \in \{\mathtt{rec1}, \ldots, \mathtt{drop2}\}$. This semantical aspect has a direct implication to the verification methodology since it allows the use of induction on constructors for proving properties of ABP.
2. The non-constructor functions $\mathtt{x1}, \mathtt{y1}, \mathtt{x2}, \mathtt{y2}$ are underspecified because there are no equations to define them, meaning that each model has its own interpretation of $\mathtt{x1}, \mathtt{y1}, \mathtt{x2}, \mathtt{y2}$. This specification technique in connection with the associativity of sequences is sufficiently expressive to model the nondeterminism, in this case, dropping elements in arbitrary positions of the communication channels. In every model the arguments of these functions are elements consisting of interpretations of the constructor terms $\sigma_n(\ldots\sigma_1(\mathtt{init}))$, where $\sigma_i \in \{\mathtt{rec1}, \ldots, \mathtt{drop2}\}$, and the values returned are sequences because the modules corresponding to the communication channels are imported with protecting. The matching equations used in conditions make the equational rules executable.

Verification approach

- Proofs cannot be automated entirely.
- The best approach is the combination between interactive proof verification and automation.
- Push the boundaries of automation.
- We have two reduction systems, one given by equations ($\_ = \_$) and the other given by rewrite rules ($\_ \Rightarrow \_$).
  How do they work together?

$$t \to^* nf(t) \Rightarrow t' \to^* nf(t') \Rightarrow t'' \ldots$$

Goal

1. Goal:
   $\boxed{goal_1}$ $\mathrm{mk(next(S)) = list(S)}$ if $\mathrm{bit1(S) = bit2(S)}$
   $\boxed{goal_2}$ $\mathrm{mk(next(S)) = pac(next(S)), list(S)}$ if $\mathrm{bit1(S) = not\ bit2(S)}$
   where $\mathrm{mk(s^n 0) = pac(s^n 0), pac(s^{n-1}), \ldots, pac(0)}$
2. Invariants:
   $\boxed{inv_5}$ $\mathrm{B \Rightarrow bit2(S)}$ if $\mathrm{channel2(S) := Ch2, B, Ch2' \bigwedge bit2(S) = not\ bit1(s)}$
   $\boxed{inv_6}$ $\mathrm{pac(next(S)) = P}$ if $\mathrm{Ch1, < B, P >, Ch1' := channel1(S) \bigwedge B1 = bit1(S)}$
3. Basic Invariants:
   $\boxed{inv_1}$ $\mathrm{B_1' \Rightarrow bit1(S)}$ if $\mathrm{Ch_1, \langle B_1, P_1 \rangle, Sq_1, \langle B_1', P_1' \rangle, Ch_1' := channel1(S) \bigwedge B_1 = bit1(S)}$
   $\boxed{inv_2}$ $\mathrm{B1 \Rightarrow bit1(s)}$ if $\mathrm{Ch1, \langle B1, P1 \rangle, Ch1 := channel1(s) \bigwedge bit1(S) = bit2(S)}$
   $\boxed{inv_3}$ $\mathrm{bit1(s) = bit2(s)}$ if $\mathrm{Ch2, B2, Ch2' := channel2(S) \bigwedge B2 = bit1(S)}$
   $\boxed{inv_4}$ $\mathrm{B2' \Rightarrow bit1(s)}$ if $\mathrm{Ch2, B2, Sq2, B2', Ch2' := channel2(S) \bigwedge B2 = bit1(S)}$

Invariants

1. In Maude $\texttt{inv}_1$, $\texttt{inv}_2$ and $\texttt{inv}_4$ are written as rewriting rules. As equational rules the above invariants would cause non-termination: when the simultaneous induction is applied to the variable `S`, $\texttt{inv}_\texttt{i}$ are added as hypotheses to the specification `ABP`; then an application of inv_1, for example, to reduce a term implies the evaluation of the condition $\texttt{Ch1}, \langle \texttt{B1}, \texttt{P1} \rangle, \texttt{Sq1}, \langle \texttt{B1}', \texttt{P1}' \rangle, \texttt{Ch1}' := \texttt{channel1}(\texttt{S})$ that requires another application of inv_1, which produces a non-termination process. Since these hypotheses are needed in the verification process, i.e., they must be executable, we choose to formalize them as rewrite rules.
2. The use of matching equations over `Sequences` as conditions of equations allows to assume an arbitrary structure of the channels of a certain type without any cost to operational semantics (the underlying conditional equations are executable).

Order of Application of Proof Rules

The application order of the proof rules is as follows:

- *Simultaneous Induction (SI)*
- *Case Analysis(CA)*
- *Sequence Case Analysis (CA(X,Y))*
- *Theorem of Constants (TC)*
- *Reduction (red)*

Simultaneous Induction I

$$\boxed{\mathrm{ABP} \vdash \{\mathrm{inv}_1, \mathrm{inv}_2, \mathrm{inv}_3, \mathrm{inv}_4\}}$$

By applying simultaneous induction we obtain

$$\boxed{\mathrm{ABP} \vdash \mathit{inv}_i[S \leftarrow \mathit{init}]}$$

$$\boxed{\mathrm{ABP} * \iota_{\mathrm{s}} \cup \{\mathrm{inv}_1, \mathrm{inv}_2, \mathrm{inv}_3, \mathrm{inv}_4\} \vdash \mathrm{inv}_{\mathrm{i}}[\mathrm{S} \leftarrow \mathrm{act}(\mathrm{s})]}$$

- $\iota_{\mathrm{s}} : \mathbb{S}\mathrm{ig}(\mathrm{ABP}) \hookrightarrow \mathbb{S}\mathrm{ig}(\mathrm{ABP})[\mathrm{s} :\rightarrow \mathrm{Sys}]$
- $\mathrm{act} \in \{\mathrm{init}, \mathrm{rec1}, \mathrm{rec2}, \mathrm{send1}, \mathrm{send2}, \mathrm{drop1}, \mathrm{drop2}\}$

Simultaneous Induction II

Maude code for $\mathrm{ABP} * \iota_s \cup \{\mathrm{inv}_1, \mathrm{inv}_2, \mathrm{inv}_3, \mathrm{inv}_4\}$

```
th INV is inc ABP .
 vars B1 B1' B2 B2' : Bit .
 vars P P1 P1' : Packet .
 vars Ch1 Sq1 Ch1' : Channel1 .
 vars Ch2 Sq2 Ch2' : Channel2 .
 op s : -> Sys .
 crl [inv1]: B1' => bit1(s) if Ch1,< B1,P1 >,Sq1,< B1',P1' >,Ch1' := channel1(s) ⋀ B1 = bit1(s).
 crl [inv2]: B1 => bit1(s) if Ch1',< B1,P1 >,Ch1 := channel1(s) ⋀ bit1(s) = bit2(s) .
 ceq [inv3]: bit2(s) = bit1(s) if Ch2,B2,Ch2' := channel2(s) ⋀ B2 = bit1(s) .
 crl [inv4]: B2' => bit1(s) if Ch2,B2,Sq2,B2',Ch2' := channel2(s) ⋀ B2 = bit1(s) .
endth
```

We present the proof trees for

- $\boxed{\mathrm{INV} \vdash \mathrm{inv}_1[S \leftarrow \mathrm{rec1}(s)]}$
- $\boxed{\mathrm{INV} \vdash \mathrm{inv}_1[S \leftarrow \mathrm{drop1}(s)]}$

Diagram - inv1

$$\mathrm{INV} \vdash \mathrm{inv}_1[\mathrm{S} \leftarrow \mathrm{act}(\mathrm{s})]\ ;\ \mathit{ABP} \vdash \mathit{inv}_1[s \leftarrow \mathit{init}]$$

I1-drop2 I1-drop1 I1-send2 I1-send1 I1-rec2 I1-rec1 I1-init

[S/]

$$\mathrm{ABP} \vdash \{\mathrm{inv}_1, \mathrm{inv}_2, \mathrm{inv}_3, \mathrm{inv}_4\}$$

Diagram - inv1[S←rec1(s)]

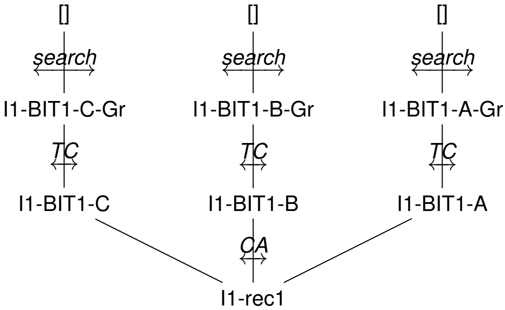
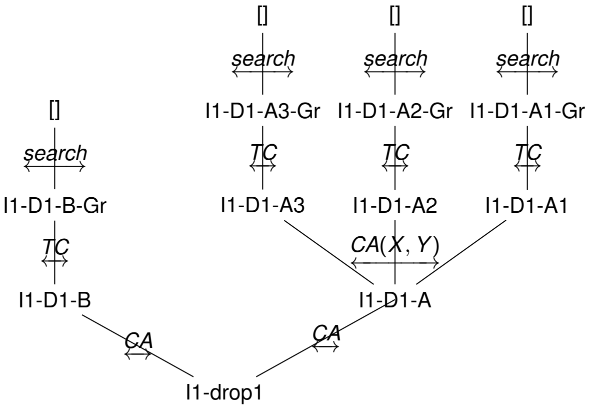


Diagram - inv1[S←drop1(s)]



Init

$\boxed{\mathrm{ABP} \vdash \mathrm{B1}' \Rightarrow \mathrm{bit1}(\mathrm{init}) \text{ if } \mathrm{Ch1}, \langle \mathrm{B1}, \mathrm{P1} \rangle, \mathrm{Sq1}, \langle \mathrm{B1}', \mathrm{P1}' \rangle, \mathrm{Ch1}' := \mathrm{channel1}(\mathrm{init}) \bigwedge \mathrm{B1} = \mathrm{bit1}(\mathrm{init})}$

red in ABP : bit1(init) .
— — — result Bit: f
red in ABP : channel1(init) .
— — — result Channel1: (empty).Channel1

$\boxed{\mathrm{ABP} \vdash \mathrm{B1}' \Rightarrow \mathrm{f} \text{ if } \mathrm{Ch1}, \langle \mathrm{B1}, \mathrm{P1} \rangle, \mathrm{Sq1}, \langle \mathrm{B1}', \mathrm{P1}' \rangle, \mathrm{Ch1}' := \mathrm{channel1}(\mathrm{init}) \bigwedge \mathrm{B1} = \mathrm{f}}$

```
th I1-INIT is inc ABP .
ops ch1 ch1' sq1 : -> Channel1 .
ops b1 b1' : -> Bit .
ops p1 p1' : -> Packet .
eq empty = ch1,< b1,p1 >,sq1,< b1',p1' >,ch1' .
eq b1 = f .
endth
```

$\boxed{\mathrm{I1-INIT} \vdash b1' \Rightarrow f}$

search true =>* false .

Receive1

INV $\models$ B1' => bit1(rec1(s)) if	Ch1,$\langle$B1,P1$\rangle$,Sq1,$\langle$B1',P1'$\rangle$,Ch1' := channel1(rec1(s)) $\bigwedge$ B1 = bit1(rec1(s))

red in INV : channel1(rec1(s)) .
— — — result Channel1: channel1(s)

INV $\vdash$ B1' => bit1(rec1(s)) if	Ch1,$\langle$B1,P1$\rangle$,Sq1,$\langle$B1',P1'$\rangle$,Ch1' := channel1(s) $\bigwedge$ B1 = bit1(rec1(s))

Receive2-A

th I1-BIT1-A is inc INV .
 eq channel2(s) = empty .
endth

I1-BIT1-A $\vdash$ B1' => bit1(rec1(s)) if	Ch1,$\langle$B1,P1$\rangle$,Sq1,$\langle$B1',P1'$\rangle$,Ch1' := channel1(s) $\bigwedge$ B1 = bit1(rec1(s))

red bit1(rec1(s)) .

— — — result Bit: bit1(s)

$\boxed{\text{I1-BIT1-A} \vdash \text{B1'} \Rightarrow \text{bit1(s) if Ch1,}\langle\text{B1,P1}\rangle\text{,Sq1,}\langle\text{B1',P1'}\rangle\text{,Ch1' := channel1(s)} \bigwedge \text{B1 = bit1(s)}}$

th I1-BIT1-A-Gr is inc I1-BIT1-A .
 ops ch1 ch1' sq1 : -> Channel1 . ops b1 b1' : -> Bit . ops p1 p1' : -> Packet .
 eq channel1(s) = ch1,< b1,p1 >,sq1,< b1',p1' >,ch1' . eq b1 = bit1(s) .
endth

$\boxed{\text{I1-BIT1-A-Gr} \vdash \text{b1'} \Rightarrow \text{bit1(s)}}$

search b1' =>* bit1(s)

Receive1-B

th I1-BIT1-B is inc INV .
 op b : -> Bit . op c2 : -> Channel2 .
 eq channel2(s) = b,c2 . eq b = not bit1(s) .
endth

$\boxed{\text{I1-BIT1-B} \vdash \text{B1' => bit1(rec1(s)) if } \begin{array}{l}\text{Ch1},\langle \text{B1,P1}\rangle,\text{Sq1},\langle \text{B1',P1'}\rangle,\text{Ch1' := channel1(s)} \\ \bigwedge \text{B1 = bit1(rec1(s))}\end{array}}$

red bit1(rec1(s)) .
— — — result Bit: bit1(s)

$\boxed{\text{I1-BIT1-B} \vdash \text{B1'} \Rightarrow \text{bit1(s) if Ch1},\langle \text{B1,P1}\rangle,\text{Sq1},\langle \text{B1',P1'}\rangle,\text{Ch1' := channel1(s)} \bigwedge \text{B1 = bit1(s)}}$

th I1-BIT1-B-Gr is inc I1-BIT1-B .
 ops ch1 ch1' sq1 : -> Channel1 . ops b1 b1' : -> Bit . ops p1 p1' : -> Packet .
 eq channel1(s) = ch1,< b1,p1 >,sq1,< b1',p1' >,ch1' . eq b1 = bit1(s) .
endth

$\boxed{\text{I1-BIT1-B-Gr} \vdash \text{b1'} \Rightarrow \text{bit1(s)}}$

search b1' =>* bit1(s)

Receive-C

```
th I1-BIT1-C is inc INV .
 op b : -> Bit . op c2 : -> Channel2 .
 eq channel2(s) = b,c2 . eq b = bit1(s) .
endth
```

$\boxed{\text{I1-BIT1-C} \vdash \text{B1' => bit1(rec1(s)) if } \begin{array}{l}\text{Ch1,}\langle\text{B1,P1}\rangle\text{,Sq1,}\langle\text{B1',P1'}\rangle\text{,Ch1' := channel1(s)} \\ \bigwedge \text{ B1 = bit1(rec1(s))}\end{array}}$

red bit1(rec1(s)) .

— — — result Bit: not bit1(s)

$\boxed{\text{I1-BIT1-C}\vdash\text{B1'}\Rightarrow\text{not bit1(s) if Ch1,}\langle\text{B1,P1}\rangle\text{,Sq1,}\langle\text{B1',P1'}\rangle\text{,Ch1':=channel1(s)}\bigwedge\text{B1=not bit1(s)}}$

```
th I1-BIT1-C-Gr is inc I1-BIT1-C .
 ops ch1 ch1' sq1 : -> Channel1 . ops b1 b1' : -> Bit . ops p1 p1' : -> Packet .
 eq channel1(s) = ch1,< b1,p1 >,sq1,< b1',p1' >,ch1' . eq b1 = not bit1(s) .
endth
```

$\boxed{\text{I1-BIT1-B-Gr} \vdash \text{b1'} \Rightarrow \text{not bit1(s)}}$

search not bit1(s) =>* bit1(s).

Conclusions

Specifying the communication channels of the ABP protocol with sequences is natural and expressive; to our knowledge, this approach is novel, at least in algebraic specifications. But this expressiveness comes with a “cost”.

1. at the operational semantics level: we use both equational and rewriting rules with conditions consisting of matching equations.
2. at the denotational semantics level: we define new proof rules to deal with the case analysis on sequences.

`Sequences` have many applications in communication protocols, and we believe that the methodology developed here can be applied successfully to many other important protocols.