Analysis of Simple Communication Protocol (2)

- Specification & Verification -

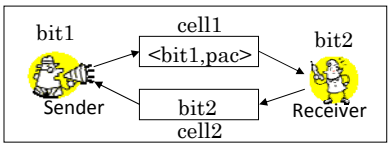
CafeOBJ Team of JAIST

Roadmap

- Review of SCP & Modeling
- Specification in CafeOBJ
- Property to Verify
- Housekeeping for Proof Scores
- Proof Score Writing

Review of SCP & Modeling

Simple Communicationu Protocol



- Although ABP uses unreliable queues, SCP uses unreliable cells. Data in the cells can be lost.

  Initially, both cells are empty & both bits are the same.
- Sender & Receiver do the following:
  - Sender puts <$\mathrm{bit1}$,pac> into $\mathrm{cell1}$ repeatedly.
  - Receiver puts $\mathrm{bit2}$ into $\mathrm{cell2}$ repeatedly.
  - When Sender gets a bit b from $\mathrm{cell2}$, if b does not equal $\mathrm{bit1}$, Sender selects the next packet and alternates $\mathrm{bit1}$.
  - When Receiver gets <b,p> from $\mathrm{cell1}$, if b equals $\mathrm{bit2}$, Receiver receives p and alternates $\mathrm{bit2}$.

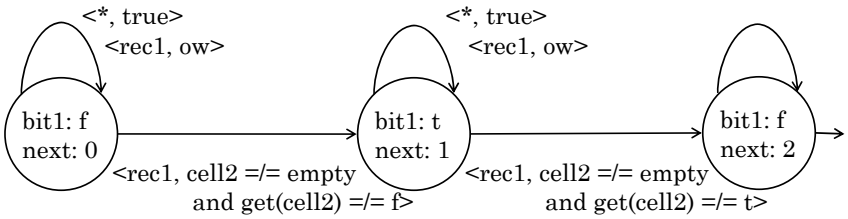
Observations

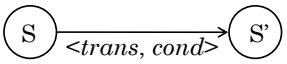
- Sender-to-Receiver channel
  `bop cell1 : Sys -> PCell`
- Receiver-to-Sender channel
  `bop cell2 : Sys -> BCell`
- Sender's bit
  `bop bit1  : Sys -> Bool`
- Receiver's bit
  `bop bit2  : Sys -> Bool`
- The ordinal of the packet sent next by Sender
  `bop next  : Sys -> Nat`
- The packets received by Receiver
  `bop list  : Sys -> List`

Transitions

- Sender's sending pairs of bits & packets
  `bop send1 : Sys -> Sys`
- Sender's receiving bits
  `bop rec1  : Sys -> Sys`
- Receiver's sending bits
  `bop send2 : Sys -> Sys`
- Receiver's receiving pairs of bits & packets
  `bop rec2  : Sys -> Sys`
- Dropping the content of $\mathrm{cell1}$
  `bop drop1 : Sys -> Sys`
- Dropping the content of $\mathrm{cell2}$
  `bop drop2 : Sys -> Sys`

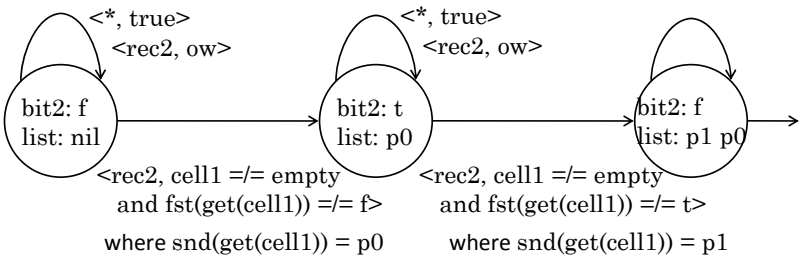
Transition Diagram of Sender





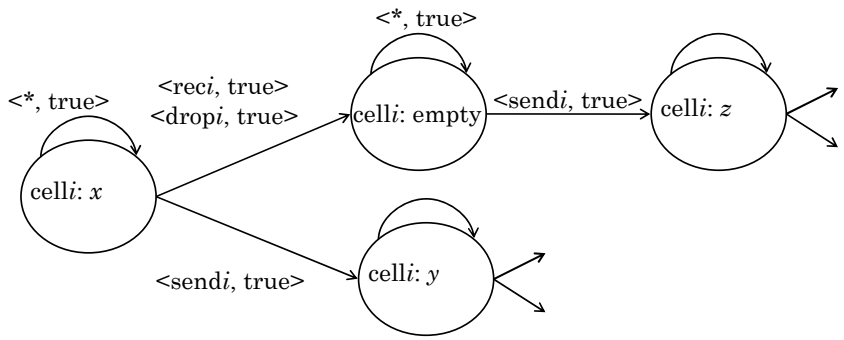
- If the condition *cond* holds in the state S, then the transition trans can change S to S'.
- <*trans*,ow> means that if any other conditions for *trans* do not hold, *trans* can change S to S'.
- * represents any transition except those explicitly stated.

Transition Diagram of Receiver



fst(< *e1*,*e2*>) = *e1*
snd(<*e1*,*e2*>) = *e2*

Transition Diagram of Channels



Specification in CafeOBJ

Data Used

- Boolean values for bits
- Natural numbers for ordinals of packets
- Packets
- Pairs of Boolean values & packets
- Cells (for channels) of pairs of BVs & pacs
- Cells (for channels) of Boolean values
- List of packets

Data Modules

- Modules
  `EQBOOL`, `PNAT`, `PACKET`, `PAIR`, `CELL`, `LIST`, `PACKET-LIST`, `BOOL-PACKET-PAIR`, `BOOL-CELL`, `BOOL-PACKET-PAIR-CELL`, `EQTRIV`
- Views
  `EQTRIV2PACKET`, `EQTRIV2EQBOOL`, `EQTRIV2BOOL-PACKET-PAIR`

Let us take a look at the file "`scp.mod`".

System Modules

- Modules
  SCP

Let us take a look at the file "`scp.mod`".

Property to Verify

Reliable Com. Property (1)

- When Receiver receives the nth packet,
  - Receiver has received the n+1 packets $\mathrm{p}0, \ldots, \mathrm{p}n$ in this order,
  - each $\mathrm{p}i$ for $i = 0,\ldots,n$ has been received only once, and
  - no other packets have been received.

Reliable Com. Property (2)

- The reachable state space R_{SCP} is inductively defined as
  - `init` is in R_{SCP}.
  - If s is in R_{SCP}, then so are `send1(`s`)`, `rec1(`s`)`, `send2(`s`)`, `rec2(`s`)`, `dup1(`s`)` and `dup2(`s`)`.
- Let `rc(`s`)` be the state predicate:

```
(bit1(s) = bit2(s)
  implies list(s) = pac(next(s)-1) … pac(0)) and
(bit1(s) =/= bit2(s)
  implies list(s) = pac(next(s)) … pac(0))
```

- All we have to do is to prove `rc(`s`)` for all s in R_{SCP}.

Housekeeping for Proof Scores

Module INV

- The module INV is declared as follows:

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

where mk(n) = pac(n) … pac(0)

- Constant S denotes an arbitrary state.

Module ISTEP

- The module ISTEP is declared as follows:

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

- Constant s' denotes an arbitrary successor state of s.
- inv1(s) is the induction hypothesis.
- inv1(s') is the formula to prove in the induction case.

Proof Score Templates

- Let us take a look at "template.mod".

Proof Score Writing

Proof Score of `inv1` (1)

- Let us consider the proof passage.

```
open ISTEP
-- arbitrary values
  op b : -> Bool .
-- assumptions
  eq cell2(s) = c(b) .
-- successor state
  eq s' = rec1(s) .
-- check
  red istep1 .
close
```

- CafeOBJ returns `mk(if (bit1(s) = b) then …) ….`
- "`bit1(s) = b`" is a candidate used to split the proof passage.

Proof Score of `inv1` (2)

- The PP is split into the two PPs:

```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = c(b) .
  eq bit1(s) = b .
  eq s' = rec1(s) .
  red istep1 .
close
```

```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = c(b) .
  eq (bit1(s) = b) = false .
  eq s' = rec1(s) .
  red istep1 .
close
```

- CafeOBJ returns `true` for the left, but … `b = bit2(s)` … for the right.
- "`b = bit2(s)`" is a candidate used to split the right.

Proof Score of `inv1` (3)

- The PP is split into the two PPs:

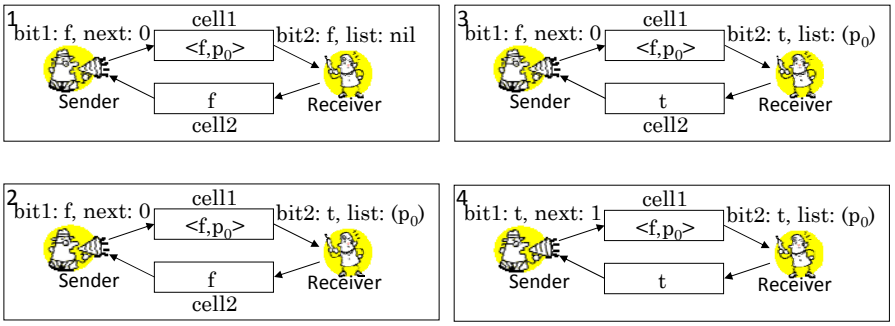
```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = c(b) .
  eq (bit1(s) = b) = false .
  eq bit2(s) = b .
  eq s' = rec1(s) .
  red istep1 .
close
```

```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = c(b) .
  eq (bit1(s) = b) = false .
  eq (bit2(s) = b) = false .
  eq s' = rec1(s) .
  red istep1 .
close
```

- CafeOBJ returns `true` for the left, but does not for the right.
- We can split the right, but try to find a lemma, which can discharge it b/c the two equations seem to contradict.

Snapshots of SCP

- 4 patterns of snapshots.



Proof Score of `inv1` (4)

- From the 4 snapshots, the following lemma can be conjectured.

```
eq inv2(S) = not(cell2(S) = empty)
       implies (bit1(S) = get(cell2(S))
                or bit2(S) = get(cell2(S))) .
```

- `inv2` can be used to discharge the PP concerned.

```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = b,bs .
  eq (bit1(s) = b) = false .
  eq (bit2(s) = b) = false .
  eq s' = rec1(s) .
  red inv2(s) implies istep1 .
close
```

Proof Score of `inv1` (5)

- Let us consider the proof passage.

```
open ISTEP
-- arbitrary values
  op p : -> BPPair .
-- assumptions
  eq cell1(s) = c(p) .
-- successor state
  eq s' = rec2(s) .
-- check
  red istep1 .
close
```

- First split the PP based on "`bit2(s) = fst(p)`" b/c it appears in the result returned by CafeOBJ.

Proof Score of `inv1` (6)

- The PP is split into the two PPs:

```
open ISTEP
  op p : -> BPPair .
  eq cell1(s) = c(p) .
  eq s' = rec2(s) .
  eq bit2(s) = fst(p) .
  red istep1 .
close
```

```
open ISTEP
  op p : -> BPPair .
  eq cell1(s) = c(p) .
  eq s' = rec2(s) .
  eq (bit2(s) = fst(p)) = false .
  red istep1 .
close
```

- CafeOBJ returns `true` for the right, but … `bit1(s) = fst(p)` … for the left.
- "`bit1(s) = fst(p)`" is a candidate used to split the right.

Proof Score of inv1 (7)

- The PP is split into the two PPs:

```
open ISTEP                    open ISTEP
  op p : -> BPPair .            op p : -> BPPair .
  eq cell1(s) = c(p) .          eq cell1(s) = c(p) .
  eq s' = rec2(s) .             eq s' = rec2(s) .
  eq bit2(s) = fst(p) .         eq bit2(s) = fst(p) .
  eq bit1(s) = fst(p) .         eq (bit1(s) = fst(p)) = false .
  red istep1 .                  red istep1 .
close                         close
```

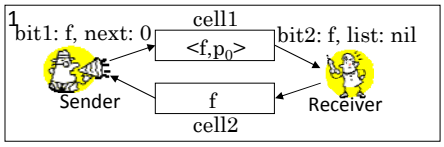
- CafeOBJ does not returns `true` for both PPs.

Proof Score of inv1 (8)

- From snapshot 1, the lemma can be conjectured:

```
eq inv4(S)
  = not(cell1(S) = empty) and bit2(S) = fst(get(cell1(S)))
    implies bit1(S) = fst(get(cell1(S))) .
```

- inv4 can discharge the right PP.




```
open ISTEP
  op p : -> BPPair .
  eq cell1(s) = c(p) .
  eq s' = rec2(s) .
  eq bit2(s) = fst(p) .
  eq (bit1(s) = fst(p)) = false .
  red inv4(s) implies istep1 .
close
```

Proof Score of `inv1` (9)

- The left PP is split based on "`pac(next(s)) = snd(p)`".

```
open ISTEP
  op p : -> BPPair .
  eq cell1(s) = c(p) .
  eq s' = rec2(s) .
  eq bit2(s) = fst(p) .
  eq bit1(s) = fst(p) .
  eq pac(next(s)) = snd(p) .
  red istep1 .
close
```

```
open ISTEP
  op p : -> BPPair .
  eq cell1(s) = c(p) .
  eq s' = rec2(s) .
  eq bit2(s) = fst(p) .
  eq bit1(s) = fst(p) .
  eq (pac(next(s)) = snd(p))
     = false .
  red istep1 .
close
```

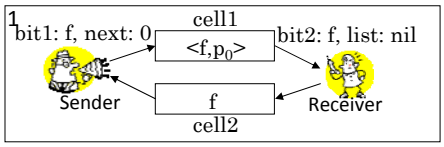
- CafeOBJ does not returns `true` for both PPs.

Proof Score of `inv1` (10)

- From snapshot 1, the lemma can be conjectured:

```
eq inv3(S)
  = not(cell1(S) = empty) and bit2(S) = fst(get(cell1(S)))
    implies pac(next(S)) = snd(get(cell1(S))) .
```

- inv3 can discharge the right PP.




```
open ISTEP
  op p : -> BPPair .
  eq cell1(s) = c(p) .
  eq s' = rec2(s) .
  eq bit2(s) = fst(p) .
  eq bit1(s) = fst(p) .
  eq (pac(next(s)) = snd(p))
     = false .
  red inv3(s) implies istep1 .
close
```

Proof Score of inv1 (11)

- The left PP can be discharged by the lemma on Boolean values:

```
eq eqbool-lemma1(B) = not((not B) = B) .
```

```
open ISTEP
  op p : -> BPPair .
  eq cell1(s) = c(p) .
  eq s' = rec2(s) .
  eq bit2(s) = fst(p) .
  eq bit1(s) = fst(p) .
  eq pac(next(s)) = snd(p) .
  red eqbool-lemma1(fst(p))
      implies istep1 .
close
```


Proof Score of inv3 (1)

- Let us consider the proof passage.

```
open ISTEP
-- arbitrary values
  op b : -> Bool .
-- assumptions
  eq cell2(s) = c(b) .
-- successor state
  eq s' = rec1(s) .
-- check
  red istep3 .
close
```

- CafeOBJ returns `pac(if (bit1(s) = b) then …) ….`
- "`bit1(s) = b`" is a candidate used to split the proof passage.

Proof Score of `inv3` (2)

- The PP is split into the two PPs:

```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = c(b) .
  eq bit1(s) = b .
  eq s' = rec1(s) .
  red istep3 .
close
```

```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = c(b) .
  eq (bit1(s) = b) = false .
  eq s' = rec1(s) .
  red istep3 .
close
```

- CafeOBJ returns `true` for the left, but … `bit2(s) = fst(get(cell1(s)))` … for the right.
- "`bit2(s) = fst(get(cell1(s)))`" is a candidate used to split the right.

Proof Score of `inv3` (3)

- The PP is split into the two PPs:

```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = c(b) .
  eq (bit1(s) = b) = false .
  eq bit2(s)
     = fst(get(cell1(s))) .
  eq s' = rec1(s) .
  red istep3 .
close
```

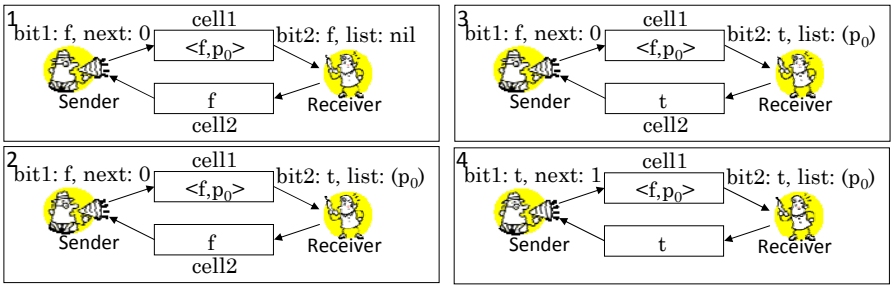
```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = c(b) .
  eq (bit1(s) = b) = false .
  eq (bit2(s)
      = fst(get(cell1(s))))
     = false .
  eq s' = rec1(s) .
  red istep3 .
close
```

- CafeOBJ returns `true` for the right, but does not for the left.

Proof Score of inv3 (4)

- From the 4 snapshots, the lemma can be conjectured:

```
eq inv5(S)
   = not(cell1(S) = empty) and not(cell2(S) = empty)
     implies (bit1(S) = get(cell2(S)) or
              not(bit2(S) = fst(get(cell1(S))))) .
```

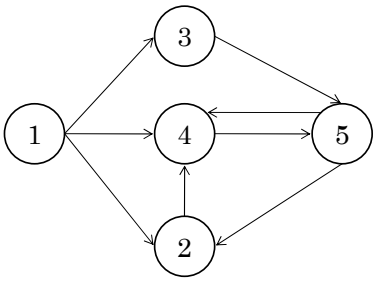


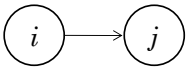
Proof Score of inv3 (5)

- inv5 can discharge the left PP.

```
open ISTEP
  op b : -> Bool .
  eq cell2(s) = c(b) .
  eq (bit1(s) = b) = false .
  eq bit2(s) = fst(get(cell1(s))) .
  eq s' = rec1(s) .
  red inv5(s) implies istep3 .
close
```

Dependence in Verification





The proof of $\texttt{inv}i$ uses $\texttt{inv}j$ as an induction hypothesis.