Analysis of Alternating Bit Protocol (2)

- Verification -

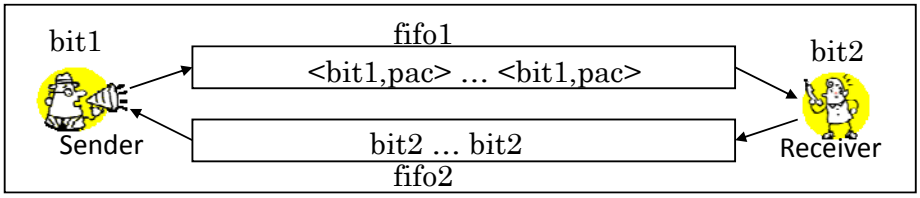
CafeOBJ Team of JAIST

Roadmap

- Review of ABP, Modeling & Specification
- Property to Verify
- Housekeeping for Proof Scores
- Proof Score Writing

Review of ABP, Modeling & Specification

Alternating Bit Protocol



- Sender puts a pair <$\mathrm{bit1}$,pac> of the bit & a packet into $\mathrm{fifo1}$ repeatedly.
- Receiver puts the bit $\mathrm{bit2}$ into $\mathrm{fifo2}$ repeatedly.
- When Sender gets a bit b from $\mathrm{fifo2}$, if b does not equal $\mathrm{bit1}$, Sender selects the next packet and alternates $\mathrm{bit1}$.
- When Receiver gets a pair <b,p> from $\mathrm{fifo1}$, if b equals $\mathrm{bit2}$, Receiver receives p and alternates $\mathrm{bit2}$.
- Data in the channels can be lost and/or duplicated.

Observations

- Sender-to-Receiver channel
  `bop fifo1 : Sys -> PFifo`
- Receiver-to-Sender channel
  `bop fifo2 : Sys -> BFifo`
- 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 1st of $fifoi$ $(i = 1,2)$
  `bop drop`i `: Sys -> Sys`
- Duplicating the 1st of $fifoi$ $(i = 1,2)$
  `bop dup`i `: Sys -> Sys`

Specification of ABP

- Let us take a brief look at the file "`abp.mod`".

Property to Verify

Reliable Com. Property (1)

- When Receiver receives the nth packet,
  - Receiver has received the n+1 packets $\mathrm{p}0$, …, $\mathrm{p}n$ 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_{ABP} is inductively defined as
  - `init` is in R_{ABP}.
  - If s is in R_{ABP}, then so are `send1(`s`)`, `rec1(`s`)`, `send2(`s`)`, `rec2(`s`)`, `drop1(`s`)`, `dup1(`s`)`, `drop2(`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_{ABP}.

Housekeeping for Proof Scores

Module INV

- The module INV is declared as follows:

```
mod INV {  pr(ABP)
  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

- First write the *proof score* (*PS*):

```
-->(I) Base case
open INV
  red inv1(init) .
close
```

```
-->(I) Induction case
open ISTEP
  red istep1 .
close
```

- Next split the ind. case into 8 sub-cases b/c there are the 8 transitions. See the file "template1.mod".
- Then split each of the 6 fragments called *proof passages* (*PPs*) into two sub-cases b/c the corrsponding transition has a non-trivial condition. See the file "template2.mod".
- Finally rewrite some equations that characterize cases. See the file "template.mod".

Proof Score Writing

What to Do

- *Case splitting*

  Split a proof passage into multiple proof passages based on 1. Boolean terms or 2. constructors

  1. (1) p holds, and (2) p does not.
  2. (1) $t = c_1(x_1,\ldots)$, …, (n) $t = c_n(x_n,\ldots)$

- *Lemma discovery*

  When CafeOBJ returns false for a proof passage or some contradiction is found in a proof passage, some lemmas should be conjectured and used.

Appropriate Equations (1)

- Each case (each PP) is characterized by a set of equations.
- It would be better to make a set of equations used confluent & terminating.
- No perfect methods, but some heuristics.
- The 3 equations are equivalent.
  1. `c-rec1(s) = true`
  2. `(fifo2(s) = empty) = false`
  3. `fifo2(s) = b,bs`

     where `b` is an arb. Boolean values & `bs` is an arb. queue of Boolean values.
- 2. (3.) is more appropriate than 1. (2.) in that 1. (2.) can be deduced with reduction when 2. (3.) is used but 2. (3.) may not when 1.(2.) is used.

Case Splitting (1)

- Let us consider the proof passage:

```
open ISTEP
-- arbitrary values
  op b : -> Bool .
  op bs : -> BFifo .
-- assumptions
  eq fifo2(s) = b,bs .
-- 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.

Case Splitting (2)

- The PP is split into the two PPs:

```
open ISTEP
  op b : -> Bool .
  op bs : -> BFifo .
  eq fifo2(s) = b,bs .
  eq bit1(s) = b .
  eq s' = rec1(s) .
  red istep1 .
close
```

```
open ISTEP
  op b : -> Bool .
  op bs : -> BFifo .
  eq fifo2(s) = b,bs .
  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.

Case Splitting (3)

- The PP is split into the two PPs:

```
open ISTEP
  op b : -> Bool .
  op bs : -> BFifo .
  eq fifo2(s) = b,bs .
  eq (bit1(s) = b) = false .
  eq bit2(s) = b .
  eq s' = rec1(s) .
  red istep1 .
close
```

```
open ISTEP
  op b : -> Bool .
  op bs : -> BFifo .
  eq fifo2(s) = b,bs .
  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.
- Note that `bit2(s) = b` is used instead of `b = bit2(s)`.

Appropriate Equations (2)

- The 2 sets of equations are equivalent.
  1. `(bit1(s) = b) = false`
     `b = bit2(s)`
  2. `(bit1(s) = b) = false`
     `bit2(s) = b`
- 2. is more appropriate than 1. in that `not(bit1(s) = bit2(s))` can be deduced with reduction when 2. is used but may not when 1. is used.

Appropriate Equations (3)

- The 2 sets of equations are equivalent.
  1. `b,b,bs = bit1,bit2,bfifo`
  2. `bit1 = b`, `bit2 = b`, and `bfifo2 = bs`

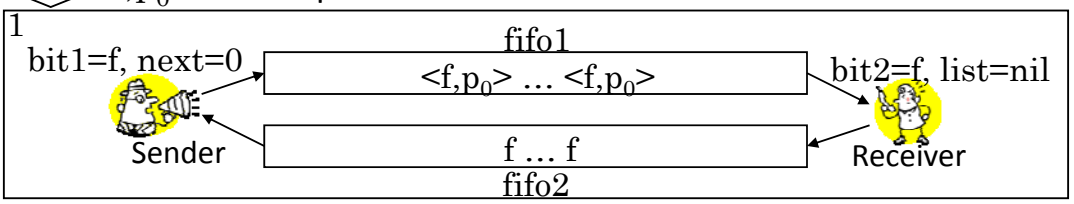
  2. is more appropriate than 1.
- The 2 sets of equations are equivalent.
  1. `(fifo2(s) = (bfifo1 @ (bit1 , empty))`
     `and bit2(s) = bit2) = true`
  2. `fifo2(s) = (bfifo1 @ (bit1 , empty))`
     `bit2(s) = bit2`

  2. is more appropriate than 1.

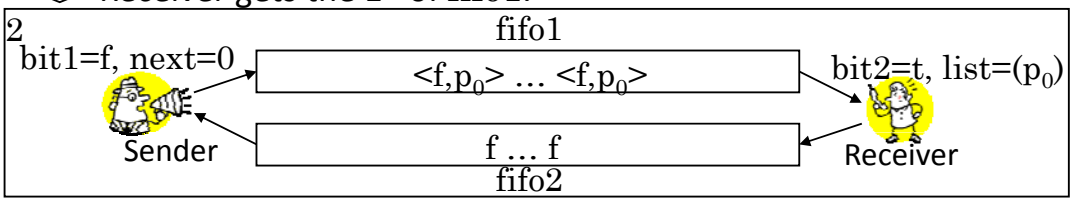
Snapshots of ABP (1)

- Let us observe snapshots to conjecture lemmas.

⇩ <f,p_0> & f are put in fifo1 & fifo2.

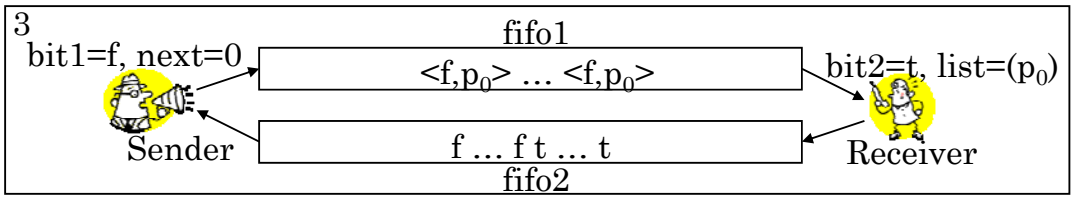


⇩ Receiver gets the 1st of fifo1.

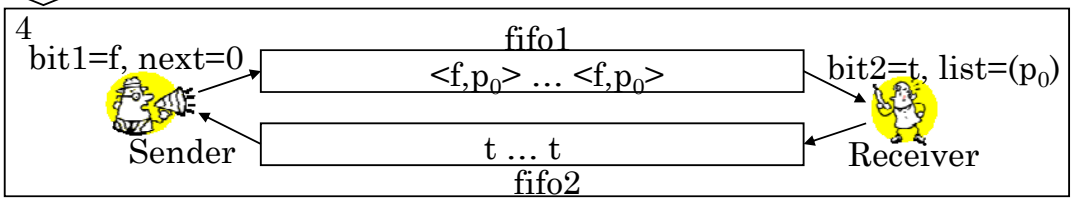


Snapshots of ABP (2)

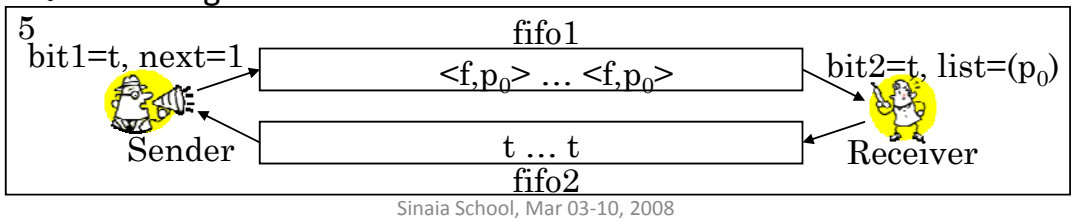
⇩ <f,p_0> & t are put in fifo1 & fifo2.



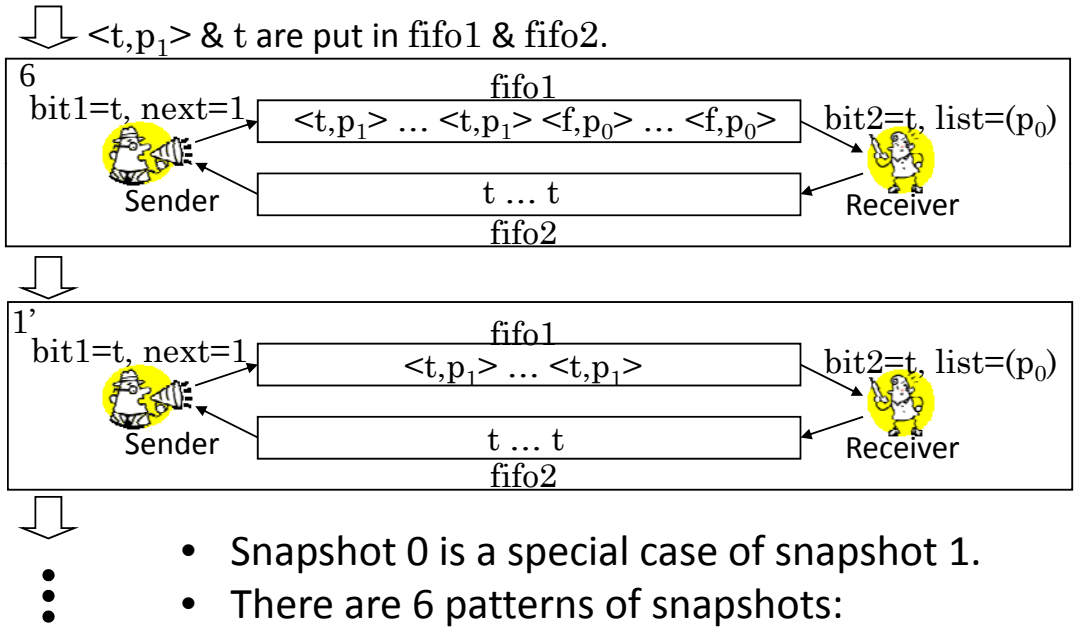
⇩



⇩ Sender gets the 1st of fifo2.



Snapshots of ABP (3)



- Snapshot 0 is a special case of snapshot 1.
- There are 6 patterns of snapshots: snapshot 1, …, snapshot 6.

Lemma Discovery (1)

- From the 6 snapshots, the following lemma can be conjectured.

```
eq inv2(S) = not(fifo2(S) = empty)
     implies ((bit1(S) = top(fifo2(S)))
              or (bit2(S) = top(fifo2(S)))) .
```

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

```
open ISTEP
  op b : -> Bool .
  op bs : -> BFifo .
  eq fifo2(s) = b,bs .
  eq (bit1(s) = b) = false .
  eq (bit2(s) = b) = false .
  eq s' = rec1(s) .
  red inv2(s) implies istep1 .
close
```

Lemma Discovery (2)

- Let us consider the proof passage:

```
open ISTEP
  op p : -> BPPair .  op ps : -> PFifo .
  eq fifo1(s) = p,ps .
  eq bit2(s) = fst(p) .
  eq (pac(next(s)) = snd(p)) = false .
  eq s' = rec2(s) .
  red istep1 .
close
```

- From snapshot 1, in which `bit2(s) = fst(p)`, the following lemma can be conjectured:

```
eq inv3(S)
   = (not(fifo1(S) = empty) and bit2(S) = fst(top(fifo1(S))))
     implies
     (bit1(S) = fst(top(fifo1(S)))
      and pac(next(S)) = snd(top(fifo1(S)))) .
```


Lemma Discovery (3)

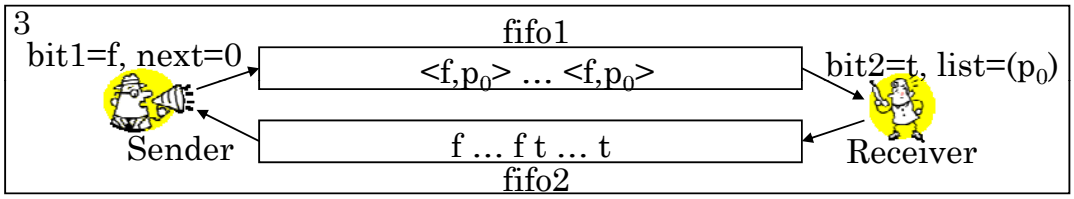
- Let us consider the proof passage:

```
open ISTEP
  op p : -> BPPair .  op ps : -> PFifo .
  eq fifo1(s) = p,ps .
  eq bit2(s) = fst(p) .
  eq pac(next(s)) = snd(p) .
  eq bit1(s) = fst(p) .
  eq (fst(p) = (fst(p) xor true)) = true .
  eq s' = rec2(s) .
  red implies istep1 .
close
```

- The following lemma on Boolean values can discharge the PP:

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

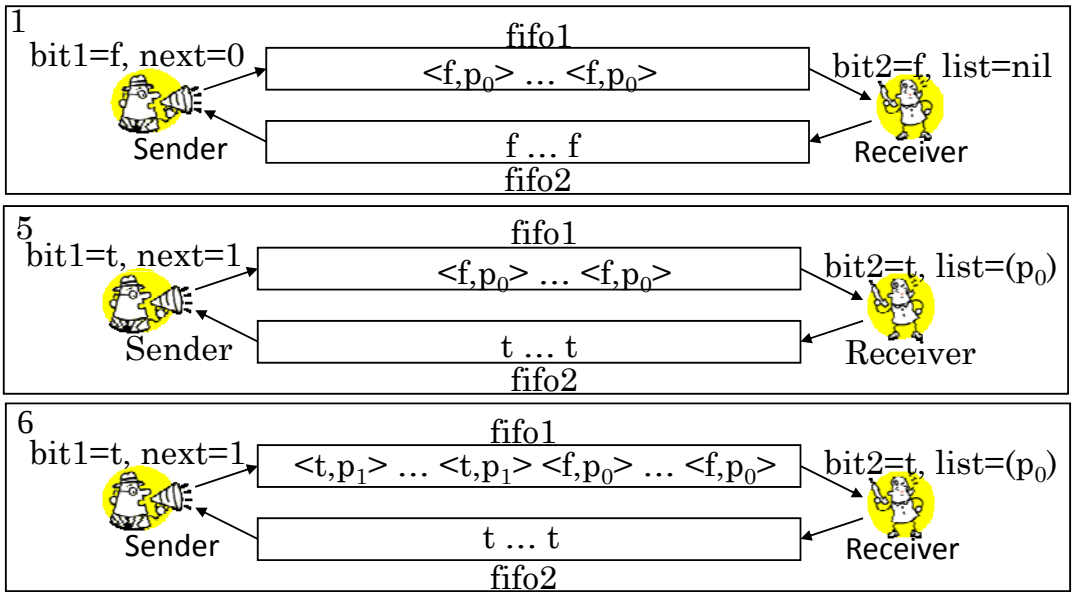
Lemma Discovery (4)



- This is the only snapshot where fifo2 contains both t & f.
- From this, the lemma can be conjectured.

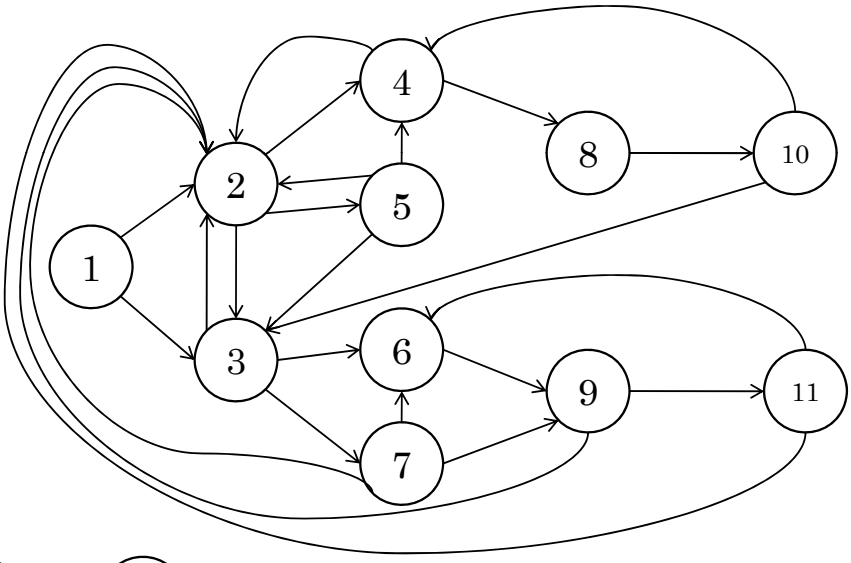
```
eq inv8(S,BIT1,BIT2,BIT3,BFIFO1,BFIFO2)
     = ((fifo2(S) = BFIFO1 @ (BIT1,BIT2,BFIFO2)
         and not(BIT1 = BIT2))
        implies ((BIT3 ¥in BFIFO2 implies BIT2 = BIT3)
                  and BIT2 = bit2(S))) .
```

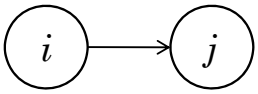

Lemma Discovery (5)




```
eq inv10(S,BIT) = ((bit1(S) = bit2(S))
       implies (BIT ¥in fifo2(S) implies BIT = bit2(S))) .
```

Dependence in Verification





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