Analysis of Alternating Bit Protocol (1)

- Modeling and Specification -

CafeOBJ Team of JAIST

Roadmap

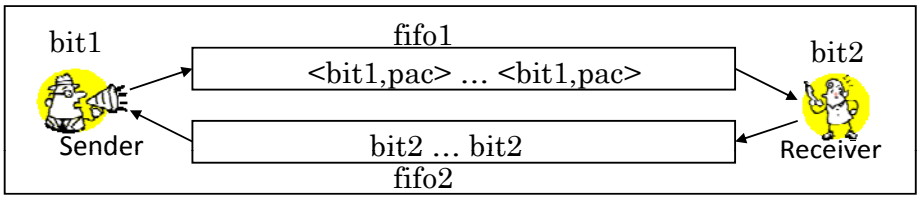
- Alternating Bit Protocol (ABP)
- Modeling ABP
- Specification in CafeOBJ
- Experiments based on Specification

Alternating Bit Protocol (ABP)

Communication Protocols

- Two processes that do not have any memories in common but share com. channels.
- Com. channels may be unreliable.
  - Data in channels may be lost and/or duplicated.
- For one process (a sender) to send packets to the other (a receiver) reliably over unreliable channels, mechanisms should be devised: *communication protocols*.

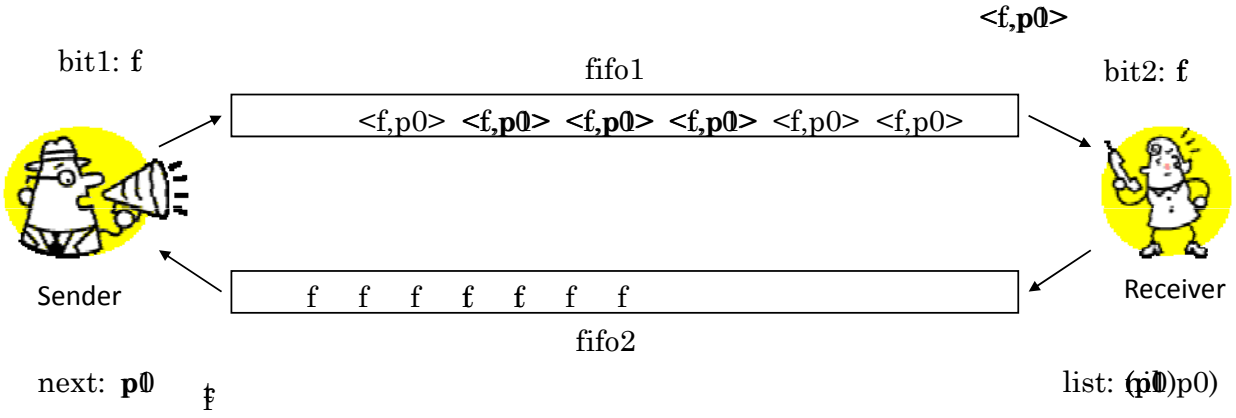
Alternating Bit Protocol



- Data in the channels may be lost and/or duplicated, but neither exchanged nor damaged.

  Initially, channels are empty & both bits are the same.
- Sender & Receiver do the following:
  - Sender puts a pair <bit1,pac> of the bit & a packet into fifo1 repeatedly.
  - Receiver puts the bit bit2 into fifo2 repeatedly.
  - When Sender gets a bit b from fifo2, if b does not equal bit1, Sender selects the next packet and alternates bit1.
  - When Receiver gets a pair <b,p> from fifo1, if b equals bit2, Receiver receives p and alternates bit2.

Animation



One Desirable Property

- 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.
- The property is called
  the *reliable communication property*
  in this talk.

Modeling ABP

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 (1)

- 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
  ```

Transitions (2)

- Any data in a channel can be lost and/or duplicated.
- Since only the top data in a channel is extracted, however, it suffices that the effects of losing and duplicating data can be seen when the data becomes top in the channel.

Transitions (3)

- Dropping the 1st of fifo1

```
bop drop1 : Sys -> Sys
```

- Duplicating the 1st of fifo1

```
bop dup1  : Sys -> Sys
```

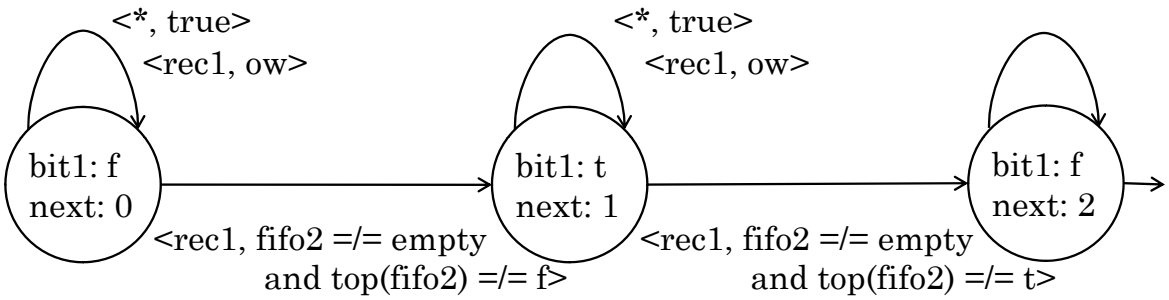
- Dropping the 1st of fifo2

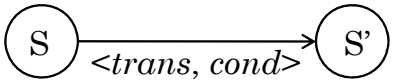
```
bop drop2 : Sys -> Sys
```

- Duplicating the 1st of fifo2

```
bop dup2  : Sys -> Sys
```

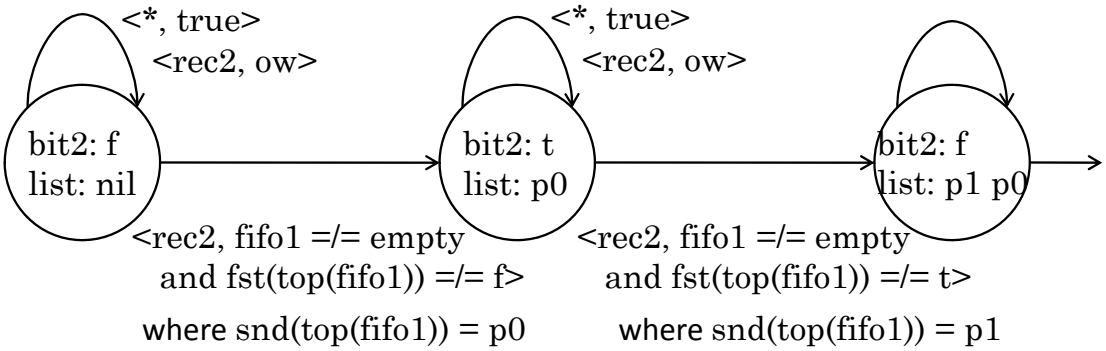
Transition Diagram of Sender





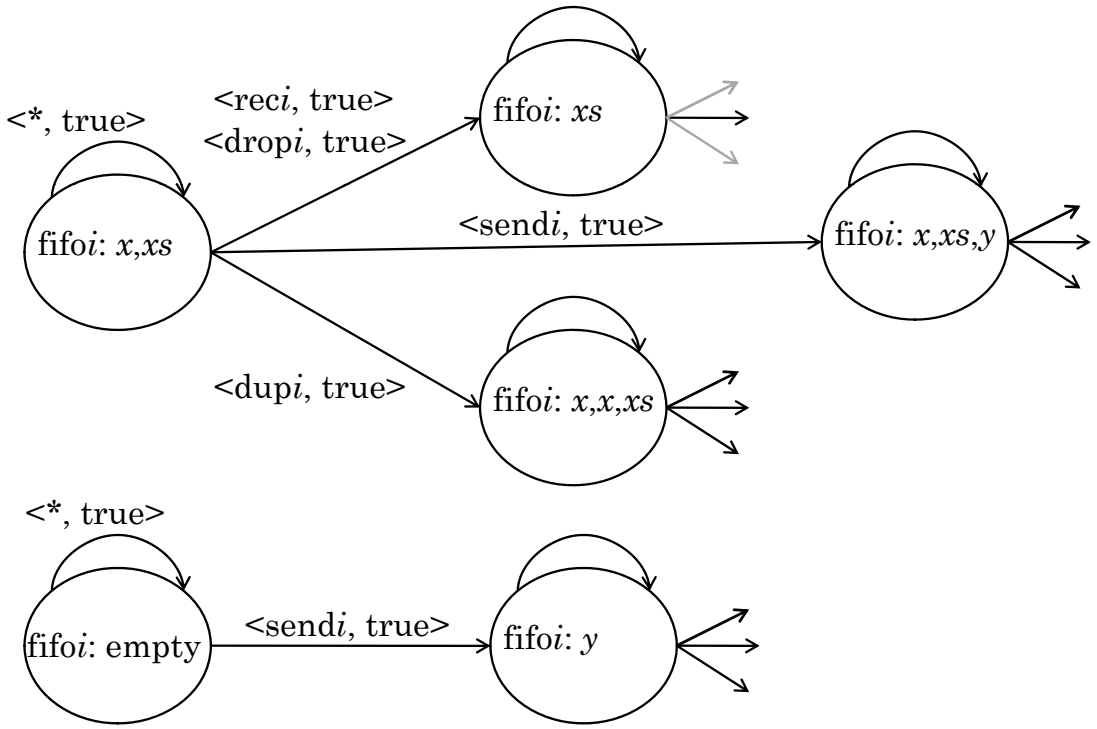
- If the condition *cond* holds in the state S, then the transition trans can change S to S'.
- <*trans*,ow> means that if any other conditions for *trans* do not hold, *trans* can change S to S'.
- * represents any transition except those explicitly stated.

Transition Diagram of Receiver



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

Transition Diagram of Channels



Specification in CafeOBJ

Overview of Specification

- The specification consists of two parts:
  - Multiple modules in which data used are specified.
  - One module in which the model of ABP is specified.

Data Used

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

Data Modules

- Modules

  EQBOOL, PNAT, PACKET, PAIR, QUEUE, LIST, PACKET-LIST, BOOL-PACKET-PAIR, BOOL-QUEUE, BOOL-PACKET-PAIR-QUEUE, EQTRIV
- Views

  EQTRIV2PACKET, EQTRIV2EQBOOL, EQTRIV2BOOL-PACKET-PAIR

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

Equations Defining Transitions

- For each `t : H` D_{t1} … D_{tm} `-> H`,
  - For each `o : H` D_{o1} … D_{on} `->` D_o,

```
ceq o(t(S,X1,…,Xm),Y1,…,Yn)
      = NewValue
      if c-t(S,X1,…,Xm) .
```

  - One more equation:

```
bceq t(S,X1,…,Xm) = S
       if not c-t(S,X1,…,Xm) .
```

System Modules

- Modules
  ABP

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

Experiments based on Specification

Naïve Way to Experiment

- Some experiments

```
eq s1 = rec2(dup1(drop1(send1(send2(send1(init)))))) .
red fifo1(s1) .
eq s2 = rec1(rec1(send2(s1))) .
red fifo1(s2) .
eq s3 = rec2(rec2(rec2(dup1(send1(s2))))) .
red fifo1(s3) .
```

- How much time does it take?
  Let t be $o(t_n(t_{n-1}(\ldots t_1(s)\ldots)))$.
  Suppose that c-t_i uses k (> 1) observations.
  The order of reducing t is k^n.

Explicit States

- States of ABP are expressed as collections of values returned by observers.

```
[Observation < State]
op _ _ : State State -> State {assoc comm}
op fifo1:_ : PFifo -> Observation
…
```

- The hidden state `init` is expressed as the explicit state:

```
fifo1: empty fifo2: empty bit1: false  bit2: false
next: 0      list: nil
```

Conversion between Hidden & Explict States

- Conversion of hidden states into explicit ones

```
op hs2es : Sys -> State
eq hs2es(S) = (fifo1: fifo1(S)) (fifo2: fifo2(S))
              (bit1: bit1(S)) (bit2: bit2(S))
              (next: next(S)) (list: list(S)) .
```

- Conversion of explicit states into hidden ones

```
op es2hs : State -> Sys
op send1 : State -> State {strat: (1 0)}
…
eq fifo1(es2hs((fifo1: PF) SS)) = PF .
…
```

- Relations b/w transitions of hidden & explicit states

```
eq send1(SS) = hs2es(send1(es2hs(SS))) .
…
```


Experiments

```
eq s0 = hs2es(init) .
red s0 .
eq s1 = rec2(dup1(drop1(send1(send2(send1(s0)))))) .
red s1 .
eq s2 = rec1(rec1(send2(s1))) .
red s2 .
eq s3 = rec2(rec2(rec2(dup1(send1(s2))))) .
red s3 .
```

- Let us take a look at the file "`experiment1.mod`".