I636: Specification and Verification of Distributed Systems

11. Alternating Bit Protocol (1)

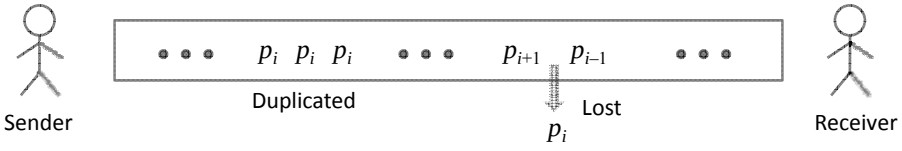
Kazuhiro Ogata & Kokichi Futatsugi

Outline

- The *Alternating Bit Protocol* (*ABP*) is a communication protocol, which aims to provide a reliable communication channel over an unreliable channel.
- ABP is a simplified version of *Transmission Control Protocol* (*TCP*) used in the Internet.
- An OTS of the protocol is made to model check some properties for the protocol.
- This lecture describes the protocol and the OTS modeling the protocol.

Unreliable Channels

- We suppose that a communication channel is *unreliable* in that packets may be *lost* and *duplicated*.
- So, when a sender sends a sequence $p_1,\ldots, p_n$ of packets to a receiver over an unreliable channel, the receiver receives a sequence $p'_1,\ldots, p'_m$ such that $p_1,\ldots, p_n$ may not equal $p'_1,\ldots, p'_m$ (which may be empty), some p_i may not appear in $p'_1,\ldots, p'_m$, and some p_j may appear in $p'_1,\ldots, p'_m$ more than once

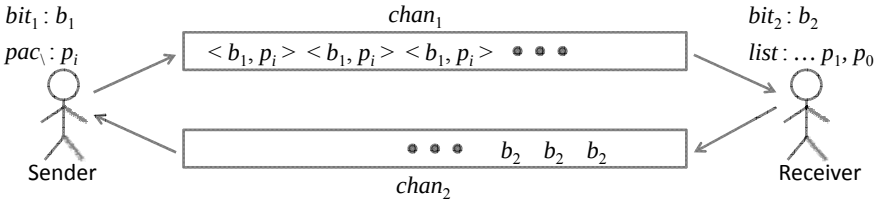


Reliable Communication over Unreliable Channels

- The purpose of ABP is to provide reliable communication over unreliable channels in that
  - When a sender sends a sequence $p_0,\ldots, p_n$ of packets to a receiver, the receiver will eventually receive exactly the same sequence $p_0,\ldots, p_n$ of packets.

Alternating Bit Protocol (1)

- *Alternating Bit Protocol* (*ABP*) uses two unreliable channels $chan_1$, $chan_2$ and two Boolean flags bit_1, bit_2.
- The sender sends pairs $< \text{Bool}, \text{Packet} >$ of Boolean values and packets to the receiver over $chan_1$.
- The receiver sends Boolean values to the sender over $chan_2$, acknowledging that it has received some packet.



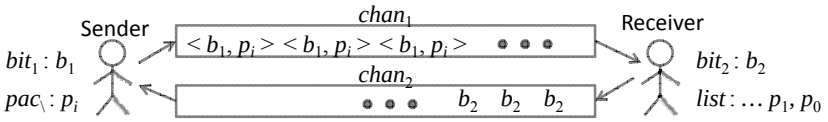
- Two more variables *pac* and *list* are used to model ABP.

  pac contains the packet to be sent.

  list contains the packets the receiver has received.

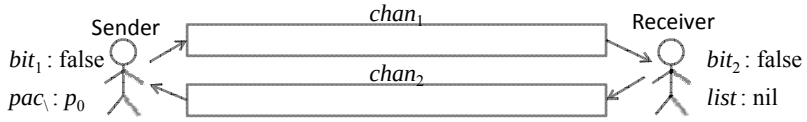
Alternating Bit Protocol (2)

- What the sender does is as follows:
  - It puts the pair $< bit_1, pac >$ of bit_1 and *pac* into $chan_1$ repeatedly.
  - When $chan_2$ is not empty, it gets the top element b from $chan_2$; if $b \neq bit_1$, then it sets *pac* to the next packet p_{i+1} and complements (negates) bit_1.
- What the receiver does is as follows:
  - It puts bit_2 into $chan_2$ repeatedly.
  - When $chan_1$ is not empty, it gets the top element $< b, p >$ from $chan_1$; if $b = bit_2$, then it puts p into *list* and complements (negates) bit_2.

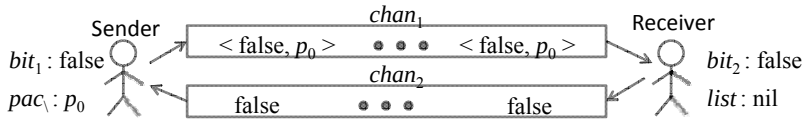


Alternating Bit Protocol (3)

0. Initially, both channels are empty, both Boolean flags are false, *pac* is p_0, and *list* is nil.

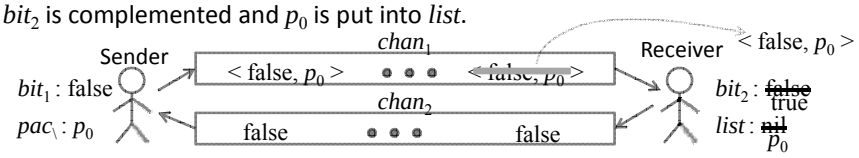


1. The sender puts < false, p_0 > into $chan_1$ repeatedly, and the receiver puts false into $chan_2$ repeatedly.

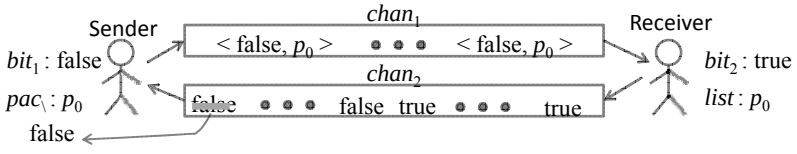


Alternating Bit Protocol (4)

2. If packets in $chan_1$ are not lost so often, some packet can arrive at the receiver. When < false, p_0 > arrives at the receiver, since bit_2 equals false, bit_2 is complemented and p_0 is put into *list*.

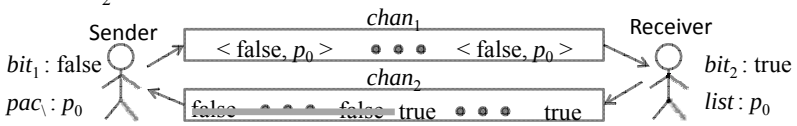


3. If packets in $chan_2$ are not lost so often, some packet can arrive at the sender. When false arrives at the sender, since bit_1 equals false, the sender does nothing.

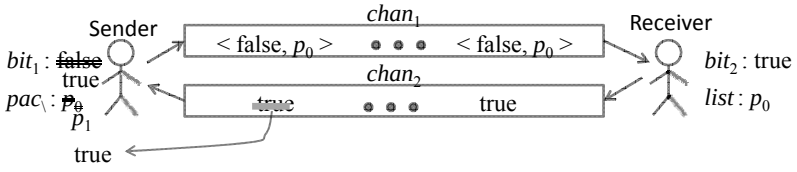


Alternating Bit Protocol (5)

4. If packets in $chan_2$ are not duplicated so often, every false disappears from $chan_2$.

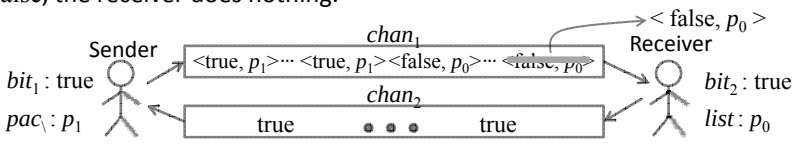


5. When true arrives at the sender, since bit_1 does not equal true, bit_1 is complemented and pac is set to the next packet p_1.

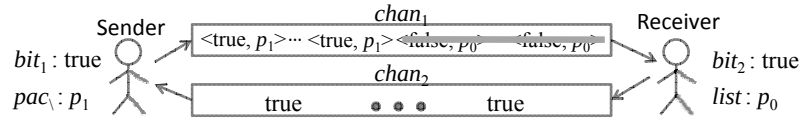


Alternating Bit Protocol (6)

6. When < false, p_0 > arrives at the receiver, since bit_2 does not equal false, the receiver does nothing.

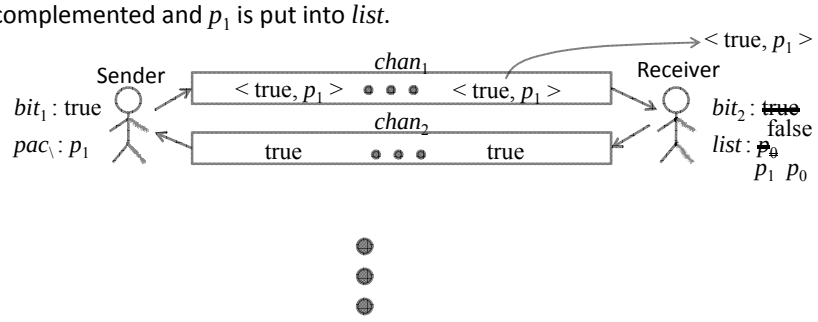


7. If packets in $chan_1$ are not duplicated so often, every < false, p_0 > disappears from $chan_1$.



Alternating Bit Protocol (7)

8. When < true, p_1 > arrives at the receiver, since bit_2 equals true, bit_2 is complemented and p_1 is put into *list*.



Modeling Packets & Pairs

- The i^{th} packet to be sent is formalized as the term: pac(i).
- The following function is defined:
  - next(pac(i)) returns the next packet pac(i+1).
  - isNth(pac(i),n) checks pac(i) is the n^{th} packet to be sent.
- The pair of a Boolean value b and a packet p is formalized as the term: < b,p >.
- The following functions are defined:
  - b(< b,p >) returns b.
  - p(< b,p >) returns p.

Modeling Channels

- Channels are formalized as queues.
- The empty queue is denoted as empty, and a non-empty queue is denoted as $e_1\ e_2\ \ldots\ e_n\ \mathrm{empty}$, where e_1 is the top element and e_n is the bottom element.
- The following functions are defined:
  - $\mathrm{put}(q,e)$ puts e into q at the bottom.
  - $\mathrm{get}(q)$ deletes the top element from q.
  - $\mathrm{top}(q)$ returns the top element of q.
  - $\mathrm{len}(q)$ returns the number of elements in q.
  - e \in q checks if q contains e.
- Queues of pairs (of Boolean values & packets) and queues of Boolean values are used.

Lists of Packets

- The empty list is denoted as nil, and a non-empty lists is denoted as $p_1\ p_2\ \ldots\ p_n\ \mathrm{nil}$, where p_1 is the top packet and p_n is the bottom packet.
- The following functions are defined:
  - $\mathrm{mk}(\mathrm{pac}(n))$ returns $\mathrm{pac}(n)\ \mathrm{pac}(n–1)\ \ldots\ \mathrm{pac}(0)\ \mathrm{nil}$.
  - p \in l checks if l contains p.

Modeling the Protocol (1)

- The observers used:
  - chan1 : $\boldsymbol{U} \rightarrow$ PairChan
  - chan2 : $\boldsymbol{U} \rightarrow$ BoolChan
  - bit1 : $\boldsymbol{U} \rightarrow$ Bool
  - bit2 : $\boldsymbol{U} \rightarrow$ Bool
  - pac : $\boldsymbol{U} \rightarrow$ Packet
  - list : $\boldsymbol{U} \rightarrow$ List

  where PairChan is the type of queues of pairs of Boolean values & packets, BoolChan is the type of queues of Boolean values, Packet is the type of packets, and List is the type of lists of packets.

Modeling the Protocol (2)

- For a state u,
  - chan1(u) is the sender-to-receiver channel $chan_1$ in state u.
  - chan2(u) is the receiver-to-sender channel $chan_2$ in state u.
  - bit1(u) is the sender's Boolean flag bit_1 in state u.
  - bit2(u) is the receiver's Boolean flag bit_2 in state u.
  - pac(u) is the packet contained in the sender's *pac* in state u.
  - list(u) is the receiver's *list* in state u.

Modeling the Protocol (3)

- For any initial state u_0,
  - $\text{chan1}(u_0)$ is the empty queue denoted by empty.
  - $\text{chan2}(u_0)$ is the empty queue denoted by empty.
  - $\text{bit1}(u_0)$ is false.
  - $\text{bit2}(u_0)$ is false.
  - $\text{pac}(u_0)$ is the first packet (denoted by $\text{pac}(0)$) to be sent.
  - $\text{list}(u_0)$ is the empty list denoted by nil.

Modeling the Protocol (4)

- The transitions used:
- The transitions modeling the behaviors of the sender & receiver.
  - $\text{send1} : \boldsymbol{U} \rightarrow \boldsymbol{U}$
  - $\text{rec1} : \boldsymbol{U} \rightarrow \boldsymbol{U}$
  - $\text{send2} : \boldsymbol{U} \rightarrow \boldsymbol{U}$
  - $\text{rec2} : \boldsymbol{U} \rightarrow \boldsymbol{U}$
- The transitions modeling the behaviors of the two unreliable channels.
  - $\text{drop1} : \boldsymbol{U} \rightarrow \boldsymbol{U}$
  - $\text{dup1} : \boldsymbol{U} \rightarrow \boldsymbol{U}$
  - $\text{drop2} : \boldsymbol{U} \rightarrow \boldsymbol{U}$
  - $\text{dup2}: \boldsymbol{U} \rightarrow \boldsymbol{U}$

Modeling the Protocol (4)

- For a state u,
  - $\mathrm{send1}(u)$ denotes the state after the sender puts $< bit_1, pac >$ into $chan_1$ in state u.
  - $\mathrm{rec1}(u)$ denotes the state after when $chan_2$ is not empty, the sender gets the top element b of $chan_2$ in state u; if b is different from bit_1, then bit_1 is complemented and pac is set to the next packet to be sent.
  - $\mathrm{send2}(u)$ denotes the state after the receiver puts bit_2 into $chan_2$ in state u.
  - $\mathrm{rec2}(u)$ denotes the state after when $chan_1$ is not empty, the receiver gets the top element $< b, p >$ of $chan_1$ in state u; if b equals bit_2, then bit_2 is complemented and p is put into $list$ at the bottom.

Modeling the Protocol (5)

- Any elements in an unreliable channel may be lost and duplicated.
- But, we restrict the behaviors of the two unreliable channels such that only the top element of the two unreliable channels may be lost and duplicated.
- The effects of loss and/or duplication in a channel can be seen only when you get the top element from the channel.
- So, if a channel is unbounded, the restriction does not affect the behaviors of the OTS S_{ABP} modeling ABP.

Modeling the Protocol (6)

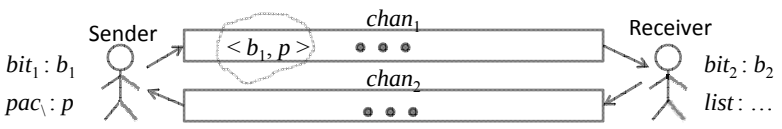
- For a state u,
  - $\mathrm{drop1}(u)$ denotes the state after when $chan_1$ is not empty, the top element is deleted from $chan_1$ in state u.
  - $\mathrm{dup1}(u)$ denotes the state after when $chan_1$ is not empty, the top element is duplicated in state u.
  - $\mathrm{drop2}(u)$ denotes the state after when $chan_2$ is not empty, the top element is deleted from $chan_2$ in state u.
  - $\mathrm{dup2}(u)$ denotes the state after when $chan_2$ is not empty, the top element is duplicated in state u.

Modeling the Protocol (7)

- Definition of the transitions:

$$\mathrm{chan1}(\mathrm{send1}(u)) = \mathrm{put}(\mathrm{chan1}(u), < \mathrm{bit1}(u), \mathrm{pac}(u) >)$$
$$\mathrm{chan2}(\mathrm{send1}(u)) = \mathrm{chan2}(u)$$
$$\mathrm{bit1}(\mathrm{send1}(u)) = \mathrm{bit1}(u)$$
$$\mathrm{bit2}(\mathrm{send1}(u)) = \mathrm{bit2}(u)$$
$$\mathrm{pac}(\mathrm{send1}(u)) = \mathrm{pac}(u)$$
$$\mathrm{list}(\mathrm{send1}(u)) = \mathrm{list}(u)$$

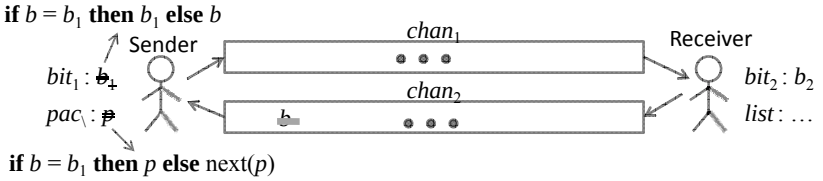
The effective condition of send1 is always true, and then it is omitted.



Modeling the Protocol (8)

- Definition of the transitions:

chan1(rec1(u)) = chan1(u) **if** c-rec1(u)
chan2(rec1(u)) = get(chan2(u)) **if** c-rec1(u)
bit1(rec1(u)) = (**if** bit1(u) = top(chan2(u)) **then** bit1(u) **else** ¬bit1(u)) **if** c-rec1(u)
bit2(rec1(u)) = bit2(u) **if** c-rec1(u)
pac(rec1(u)) = (**if** bit1(u) = top(chan2(u)) **then** pac(u) **else** next(pac(u))) **if** c-rec1(u)
list(rec1(u)) = list(u) **if** c-rec1(u)
rec1(u) = $_{S_{ABP}}$ u **if** ¬c-rec1(u)
where c-rec1(u) is chan2(u) ≠ empty.

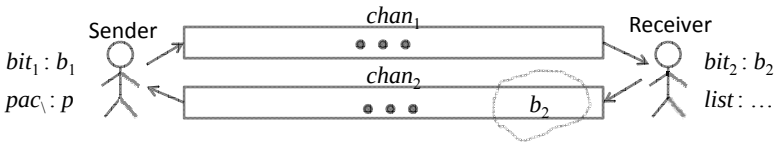


Modeling the Protocol (9)

- Definition of the transitions:

chan1(send2(u)) = chan1(u)
chan2(send2(u)) = put(chan2(u),bit2(u))
bit1(send2(u)) = bit1(u)
bit2(send2(u)) = bit2(u)
pac(send2(u)) = pac(u)
list(send2(u)) = list(u)

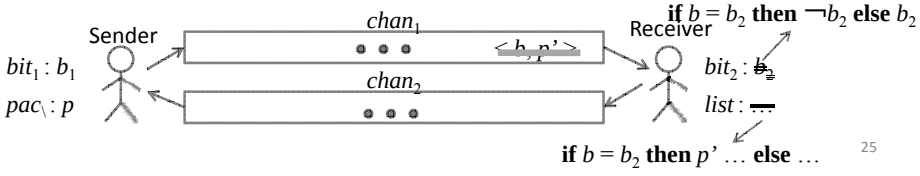
The effective condition of send2 is always true, and then it is omitted.



Modeling the Protocol (10)

- Definition of the transitions:

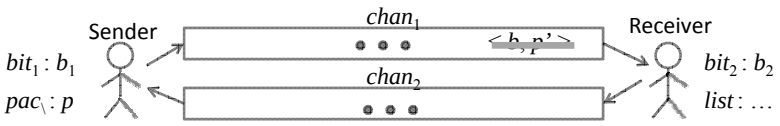
chan1(rec2(u)) = get(chan1(u)) **if** c-rec2(u)
chan2(rec2(u)) = chan2(u) **if** c-rec2(u)
bit1(rec2(u)) = bit1(u) **if** c-rec2(u)
bit2(rec2(u)) = (**if** bit2(u) = b(top(chan1(u))) **then** ¬bit2(u) **else** bit2(u))
if c-rec2(u)
pac(rec2(u)) = pac(u) **if** c-rec2(u)
list(rec2(u)) = (**if** bit2(u) = b(top(chan1(u)))
then p(top(chan1(u))) list(u) **else** list(u)) **if** c-rec2(u)
rec2(u) $=_{S_{ABP}}$ u **if** ¬c-rec2(u)
where c-rec2(u) is chan1(u) ≠ empty.



Modeling the Protocol (11)

- Definition of the transitions:

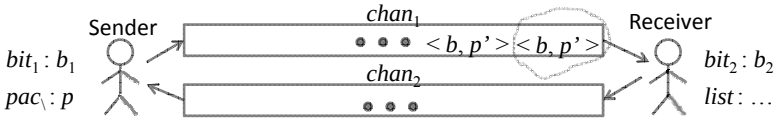
chan1(drop1(u)) = get(chan1(u)) **if** c-drop1(u)
chan2(drop1(u)) = chan2(u) **if** c-drop1(u)
bit1(drop1(u)) = bit1(u) **if** c-drop1(u)
bit2(drop1(u)) = bit2(u) **if** c-drop1(u)
pac(drop1(u)) = pac(u) **if** c-drop1(u)
list(drop1(u)) = list(u)) **if** c-drop1(u)
drop1(u) $=_{S_{ABP}}$ u **if** ¬c-drop1(u)
where c-drop1(u) is chan1(u) ≠ empty.



Modeling the Protocol (12)

- Definition of the transitions:

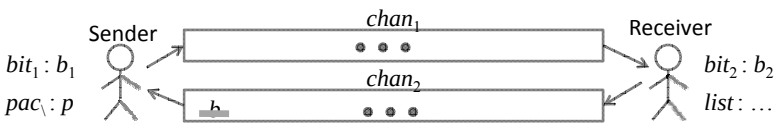
chan1(dup1(u)) = top(chan1(u)) chan1(u) **if** c-dup1(u)
chan2(dup1(u)) = chan2(u) **if** c-dup1(u)
bit1(dup1(u)) = bit1(u) **if** c-dup1(u)
bit2(dup1(u)) = bit2(u) **if** c-dup1(u)
pac(dup1(u)) = pac(u) **if** c-dup1(u)
list(dup1(u)) = list(u)) **if** c-dup1(u)
dup1(u) $=_{S_{ABP}}$ u **if** ¬c-dup1(u)
where c-dup1(u) is chan1(u) ≠ empty.



Modeling the Protocol (13)

- Definition of the transitions:

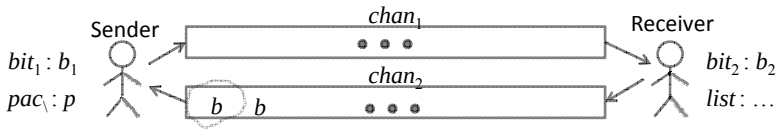
chan1(drop2(u)) = chan1(u) **if** c-drop2(u)
chan2(drop2(u)) = get(chan2(u)) **if** c-drop2(u)
bit1(drop2(u)) = bit1(u) **if** c-drop2(u)
bit2(drop2(u)) = bit2(u) **if** c-drop2(u)
pac(drop2(u)) = pac(u) **if** c-drop2(u)
list(drop2(u)) = list(u)) **if** c-drop2(u)
drop2(u) $=_{S_{ABP}}$ u **if** ¬c-drop2(u)
where c-drop2(u) is chan2(u) ≠ empty.



Modeling the Protocol (14)

- Definition of the transitions:

chan1(dup2(u)) = chan1(u) **if** c-dup2(u)
chan2(dup2(u)) = top(chan2(u)) chan2(u) **if** c-dup2(u)
bit1(dup2(u)) = bit1(u) **if** c-dup2(u)
bit2(dup2(u)) = bit2(u) **if** c-dup2(u)
pac(dup2(u)) = pac(u) **if** c-dup2(u)
list(dup2(u)) = list(u)) **if** c-dup2(u)
dup2(u) $=_{S_{ABP}}$ u **if** ¬c-dup2(u)
where c-dup2(u) is chan2(u) ≠ empty.



Formalizing the Properties (1)

- The property (requirement) that "when the sender sends a sequence pac(0),…,pac(n) of packets to the receiver, the receiver will eventually receive exactly the same sequence pac(0),…,pac(n) of packets" is divided into two properties:
  1. *Reliable Communication Property*: When the receiver receives pac(n), it has received the n+1 packets pac(0),…,pac(n) in this order, each pac(i) has been received only once and no other packets have been received.
  2. *Progress Communication Property*: Each pac(i) will be eventually delivered to the receiver.

Formalizing the Properties (2)

- Since Reliable Communication Property assumes that some packet is received, even a protocol that does not deliver any packets to the receiver does satisfy the property.

  So, only this property is not enough.
- Progress Communication Property discharges the assumption.
- But, only Progress Communication Property is not enough either because the property does not guarantee that each packet is received exactly once and packets are received in the same order in which they are sent.

Formalizing the Properties (2)

- Reliable Communication Property is formalized as $\Box$ p-rcp(u), where p-rcp(u) is as follows:

  $(\text{bit1}(u) = \text{bit2}(u) \rightarrow \text{mk}(\text{pac}(u)) = (\text{pac}(u)\ \text{list}(u))$
  $\wedge\ (\text{bit1}(u) \neq \text{bit2}(u) \rightarrow \text{mk}(\text{pac}(u)) = \text{list}(u))$
- Progress Communication Property is formalized as $\Diamond$ isDelivered(u,i) for each i, where isDelivered(u,n) is as follows:

  pac(i) \in list(u)