Modeling Concurrent and Probabilistic Systems

Lecture 10: The Alternating Bit Protocol

Joost-Pieter Katoen   Thomas Noll

Software Modeling and Verification Group
RWTH Aachen University
noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/mcps07/

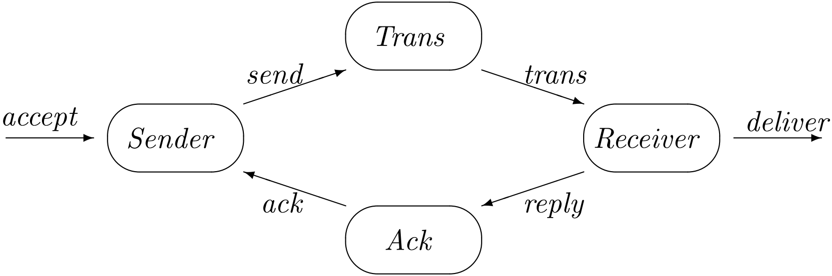
Winter Semester 2007/08

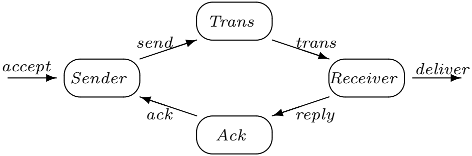
Outline

Repetition: The Setting

Goal: design of a communication protocol which guarantees reliable data transfer over unreliable channels

Overview of system "architecture":

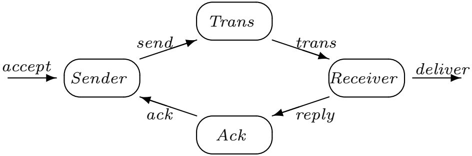




- *Sender* transfers data (from a given finite set D) to *Receiver* using channel *Trans*
- *Receiver* confirms reception via *Ack*
- Properties of channels:
  - unidirectional data transfer
  - capacity: one message
    ($\Longrightarrow$ sequential, i.e., respects order of messages)
  - detection of transmission errors
    (loss/duplication/corruption of messages)
  - errors reported to target process

Idea: use redundancy (additional control bit) to ensure safeness of data transfer

Repetition: Modeling of Channels



- *Trans* transmits frames of the following form:

$$F := \{db \mid d \in D, b \in \{0,1\}\} \qquad \text{(finite)}$$

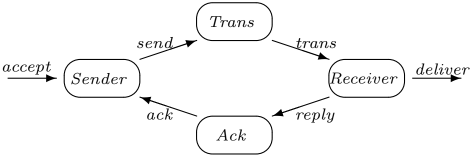
  It detects transmission errors and reports it to *Receiver*:

$$\mathit{Trans} = \sum_{f \in F} \mathit{send}_f.(\underbrace{\overline{\mathit{trans}_f}.\mathit{Trans}}_{\text{successful}} + \underbrace{\overline{\mathit{trans}_\perp}.\mathit{Trans}}_{\text{error}})$$

- *Ack* behaves like *Trans* but transmits only control bits:

$$\mathit{Ack} = \sum_{b \in \{0,1\}} \mathit{reply}_b.(\underbrace{\overline{\mathit{ack}_b}.\mathit{Ack}}_{\text{successful}} + \underbrace{\overline{\mathit{ack}_\perp}.\mathit{Ack}}_{\text{error}})$$

Repetition: Design Goal



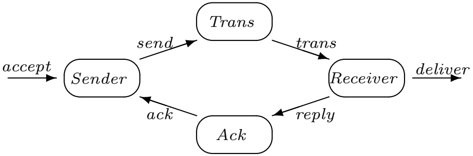
Under the above side conditions, give CCS implementations of *Sender* and *Receiver* such that the overall system works correctly, i.e., behaves like a one-element buffer:

$$
\begin{aligned}
Buffer(\overrightarrow{accept}, \overrightarrow{deliver}) &= \sum_{d \in D} accept_d . Buffer_d(\overrightarrow{accept}, \overrightarrow{deliver}) \\
Buffer_d(\overrightarrow{accept}, \overrightarrow{deliver}) &= \overline{deliver_d} . Buffer(\overrightarrow{accept}, \overrightarrow{deliver})
\end{aligned}
$$

where

$$
\begin{aligned}
\overrightarrow{accept} &:= (accept_{d_1}, \ldots, accept_{d_n}) \\
\text{and } \overrightarrow{deliver} &:= (deliver_{d_1}, \ldots, deliver_{d_n}) \\
\text{for } D &= \{d_1, \ldots, d_n\}
\end{aligned}
$$

Outline

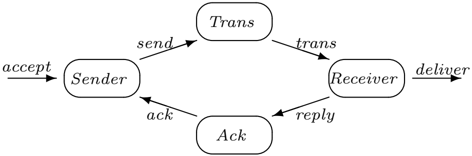


Sender accepts $d \in D$ via $accept_d$ and repeatedly sends frames of the form $d0$ over *Trans* until it receives the acknowledgment 0 over *Ack*. For the next data item, control bit 1 is used and so on ($\Longrightarrow$ "Alternating Bit Protocol").

Formally, for $b \in \{0,1\}$ and $d \in D$:

$$
\begin{aligned}
Sender &= Sender_0 \\
Sender_b &= \sum_{d \in D} accept_d . Send_{db} \\
Send_{db} &= \overline{send_{db}} . Wait_{db} \\
Wait_{db} &= \underbrace{ack_b . Sender_{1-b}}_{\text{successful}} + \underbrace{ack_{1-b} . Send_{db} + ack_{\perp} . Send_{db}}_{\text{error}}
\end{aligned}
$$

Implementation of the Sender

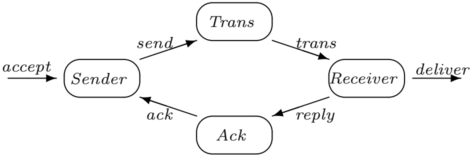


Sender accepts $d \in D$ via $accept_d$ and repeatedly sends frames of the form $d0$ over *Trans* until it receives the acknowledgment 0 over *Ack*. For the next data item, control bit 1 is used and so on ($\implies$ “Alternating Bit Protocol”).

Formally, for $b \in \{0, 1\}$ and $d \in D$:

$$
\begin{aligned}
Sender &= Sender_0 \\
Sender_b &= \sum_{d \in D} accept_d.Send_{db} \\
Send_{db} &= \overline{send_{db}}.Wait_{db} \\
Wait_{db} &= \underbrace{ack_b.Sender_{1-b}}_{\text{successful}} + \underbrace{ack_{1-b}.Send_{db} + ack_\perp.Send_{db}}_{\text{error}}
\end{aligned}
$$

Implementation of the Sender

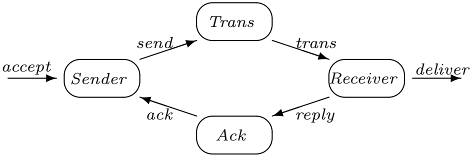


Sender accepts $d \in D$ via $accept_d$ and repeatedly sends frames of the form $d0$ over *Trans* until it receives the acknowledgment 0 over *Ack*. For the next data item, control bit 1 is used and so on ($\implies$ "Alternating Bit Protocol").

Formally, for $b \in \{0,1\}$ and $d \in D$:

$$
\begin{aligned}
Sender &= Sender_0 \\
Sender_b &= \sum_{d \in D} accept_d.Send_{db} \\
Send_{db} &= \overline{send_{db}}.Wait_{db} \\
Wait_{db} &= \underbrace{ack_b.Sender_{1-b}}_{\text{successful}} + \underbrace{ack_{1-b}.Send_{db} + ack_{\perp}.Send_{db}}_{\text{error}}
\end{aligned}
$$

Implementation of the Sender

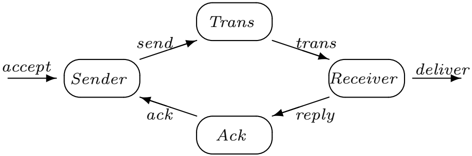


Sender accepts $d \in D$ via $accept_d$ and repeatedly sends frames of the form $d0$ over *Trans* until it receives the acknowledgment 0 over *Ack*. For the next data item, control bit 1 is used and so on ($\Longrightarrow$ "Alternating Bit Protocol").

Formally, for $b \in \{0, 1\}$ and $d \in D$:

$$\begin{aligned}
Sender &= Sender_0 \\
Sender_b &= \sum_{d \in D} accept_d.Send_{db} \\
Send_{db} &= \overline{send_{db}}.Wait_{db} \\
Wait_{db} &= \underbrace{ack_b.Sender_{1-b}}_{\text{successful}} + \underbrace{ack_{1-b}.Send_{db} + ack_\perp.Send_{db}}_{\text{error}}
\end{aligned}$$

Implementation of the Sender



Sender accepts $d \in D$ via $accept_d$ and repeatedly sends frames of the form $d0$ over *Trans* until it receives the acknowledgment 0 over *Ack*. For the next data item, control bit 1 is used and so on ($\implies$ "Alternating Bit Protocol").

Formally, for $b \in \{0, 1\}$ and $d \in D$:

$$\begin{aligned}
Sender &= Sender_0 \\
Sender_b &= \sum_{d \in D} accept_d.Send_{db} \\
Send_{db} &= \overline{send_{db}}.Wait_{db} \\
Wait_{db} &= \underbrace{ack_b.Sender_{1-b}}_{\text{successful}} + \underbrace{ack_{1-b}.Send_{db} + ack_\perp.Send_{db}}_{\text{error}}
\end{aligned}$$

Implementation of the Receiver

Receiver gets frames of the form db or $\perp$. In the first case, if b has the expected value, d is forwarded via $\mathit{deliver}_d$, and b is returned via Ack. Otherwise the transmission is re-initiated by returning the "wrong" control bit $1-b$ to Sender.

Formally, for $b \in \{0,1\}$ and $d \in D$:

$$
\begin{array}{rcl}
\mathit{Receiver} & = & \mathit{Receiver}_0 \\
\mathit{Receiver}_b & = & \displaystyle\sum_{d\in D} \mathit{trans}_{db}.\mathit{Reply}_{db} \\
 & + & \displaystyle\sum_{d\in D} \mathit{trans}_{d(1-b)}.\overline{\mathit{reply}_{1-b}}.\mathit{Receiver}_b \\
 & + & \mathit{trans}_{\perp}.\overline{\mathit{reply}_{1-b}}.\mathit{Receiver}_b \\
\mathit{Reply}_{db} & = & \overline{\mathit{deliver}_d}.\overline{\mathit{reply}_b}.\mathit{Receiver}_{1-b}
\end{array}
$$

Implementation of the Receiver

Receiver gets frames of the form db or $\perp$. In the first case, if b has the expected value, d is forwarded via $\mathit{deliver}_d$, and b is returned via Ack. Otherwise the transmission is re-initiated by returning the "wrong" control bit $1 - b$ to Sender.

Formally, for $b \in \{0, 1\}$ and $d \in D$:

$$
\begin{aligned}
\mathit{Receiver} &= \mathit{Receiver}_0 \\
\mathit{Receiver}_b &= \sum_{d \in D} \mathit{trans}_{db}.\mathit{Reply}_{db} \\
&\quad + \sum_{d \in D} \mathit{trans}_{d(1-b)}.\overline{\mathit{reply}_{1-b}}.\mathit{Receiver}_b \\
&\quad + \mathit{trans}_{\perp}.\overline{\mathit{reply}_{1-b}}\mathit{Receiver}_b \\
\mathit{Reply}_{db} &= \overline{\mathit{deliver}_d}.\overline{\mathit{reply}_b}.\mathit{Receiver}_{1-b}
\end{aligned}
$$

Implementation of the Receiver

Receiver gets frames of the form db or $\perp$. In the first case, if b has the expected value, d is forwarded via $deliver_d$, and b is returned via Ack. Otherwise the transmission is re-initiated by returning the "wrong" control bit $1-b$ to $Sender$.

Formally, for $b \in \{0,1\}$ and $d \in D$:

$$\begin{array}{rcl}
Receiver & = & Receiver_0 \\
Receiver_b & = & \sum_{d\in D} trans_{db}.Reply_{db} \\
& + & \sum_{d\in D} trans_{d(1-b)}.\overline{reply_{1-b}}.Receiver_b \\
& + & trans_{\perp}.\overline{reply_{1-b}}Receiver_b \\
Reply_{db} & = & \overline{deliver_d}.\overline{reply_b}.Receiver_{1-b}
\end{array}$$

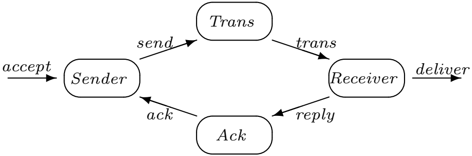
Implementation of the Receiver

Receiver gets frames of the form db or $\perp$. In the first case, if b has the expected value, d is forwarded via $deliver_d$, and b is returned via Ack. Otherwise the transmission is re-initiated by returning the "wrong" control bit $1 - b$ to $Sender$.

Formally, for $b \in \{0, 1\}$ and $d \in D$:

$$
\begin{aligned}
Receiver &= Receiver_0 \\
Receiver_b &= \sum_{d \in D} trans_{db}.Reply_{db} \\
&+ \sum_{d \in D} trans_{d(1-b)}.\overline{reply_{1-b}}.Receiver_b \\
&+ trans_{\perp}.\overline{reply_{1-b}}Receiver_b \\
Reply_{db} &= \overline{deliver_d}.\overline{reply_b}.Receiver_{1-b}
\end{aligned}
$$

The Overall System



The overall system is given by

$$ABP(\overrightarrow{accept}, \overrightarrow{deliver}) = \mathsf{new}\, L\, (Sender \parallel Trans \parallel Ack \parallel Receiver)$$

where

$$L := \{send_{db}, trans_{db}, reply_b, ack_b \mid db \in F\} \cup \{trans_\perp, ack_\perp\}$$

Outline

Correctness of the ABP I

Theorem 10.1

$$ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$$

Remark: because of internal τ-steps in ABP, $ABP \sim Buffer$ cannot hold.

Proof.

1. Construct transition system of $ABP(\overrightarrow{accept}, \overrightarrow{deliver})$ (next slide; $S = Sender$, $W = Wait$, $T = Trans$, $A = Ack$, $R = Receiver/Reply$, $d, e \in D$; without restrictions)
2. Show that $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \approx Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$
3. $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \overset{\tau}{\not\longrightarrow}$ and $Buffer(\overrightarrow{accept}, \overrightarrow{deliver}) \overset{\tau}{\not\longrightarrow}$ $\implies ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$

□

Theorem 10.1

$$ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$$

Remark: because of internal τ-steps in ABP, $ABP \sim Buffer$ cannot hold.

Proof.

1. Construct transition system of $ABP(\overrightarrow{accept}, \overrightarrow{deliver})$ (next slide; $S = Sender$, $W = Wait$, $T = Trans$, $A = Ack$, $R = Receiver/Reply$, $d, e \in D$; without restrictions)
2. Show that $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \approx Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$
3. $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \not\xrightarrow{\tau}$ and $Buffer(\overrightarrow{accept}, \overrightarrow{deliver}) \not\xrightarrow{\tau}$ $\implies ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$

□

Correctness of the ABP I

Theorem 10.1

$ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$

Remark: because of internal τ-steps in ABP, $ABP \sim Buffer$ cannot hold.

Proof.

1. Construct transition system of $ABP(\overrightarrow{accept}, \overrightarrow{deliver})$ (next slide; $S = Sender$, $W = Wait$, $T = Trans$, $A = Ack$, $R = Receiver/Reply$, $d, e \in D$; without restrictions)
2. Show that $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \approx Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$
3. $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \not\xrightarrow{\tau}$ and $Buffer(\overrightarrow{accept}, \overrightarrow{deliver}) \not\xrightarrow{\tau}$
   $\implies ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$

□

Correctness of the ABP I

Theorem 10.1

$ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$

Remark: because of internal τ-steps in ABP, $ABP \sim Buffer$ cannot hold.

Proof.

1. Construct transition system of $ABP(\overrightarrow{accept}, \overrightarrow{deliver})$ (next slide; $S = Sender$, $W = Wait$, $T = Trans$, $A = Ack$, $R = Receiver/Reply$, $d, e \in D$; without restrictions)
2. Show that $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \approx Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$
3. $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \stackrel{\tau}{\not\longrightarrow}$ and $Buffer(\overrightarrow{accept}, \overrightarrow{deliver}) \stackrel{\tau}{\not\longrightarrow}$
   $\Longrightarrow ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$

□

Correctness of the ABP I

Theorem 10.1

$ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq \mathit{Buffer}(\overrightarrow{accept}, \overrightarrow{deliver})$

Remark: because of internal τ-steps in ABP, $ABP \sim \mathit{Buffer}$ cannot hold.

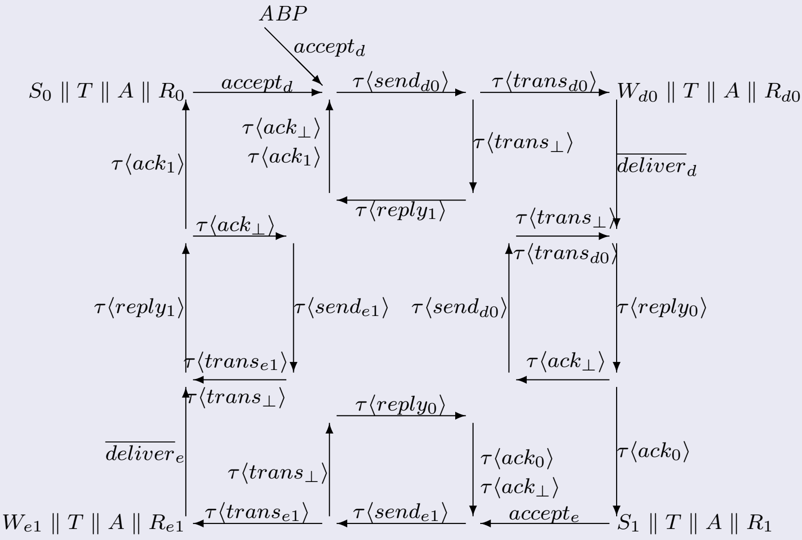
Proof.

1. Construct transition system of $ABP(\overrightarrow{accept}, \overrightarrow{deliver})$ (next slide; $S = \mathit{Sender}$, $W = \mathit{Wait}$, $T = \mathit{Trans}$, $A = \mathit{Ack}$, $R = \mathit{Receiver}/\mathit{Reply}$, $d, e \in D$; without restrictions)
2. Show that $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \approx \mathit{Buffer}(\overrightarrow{accept}, \overrightarrow{deliver})$
3. $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \not\xrightarrow{\tau}$ and $\mathit{Buffer}(\overrightarrow{accept}, \overrightarrow{deliver}) \not\xrightarrow{\tau}$
   $\implies ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq \mathit{Buffer}(\overrightarrow{accept}, \overrightarrow{deliver})$

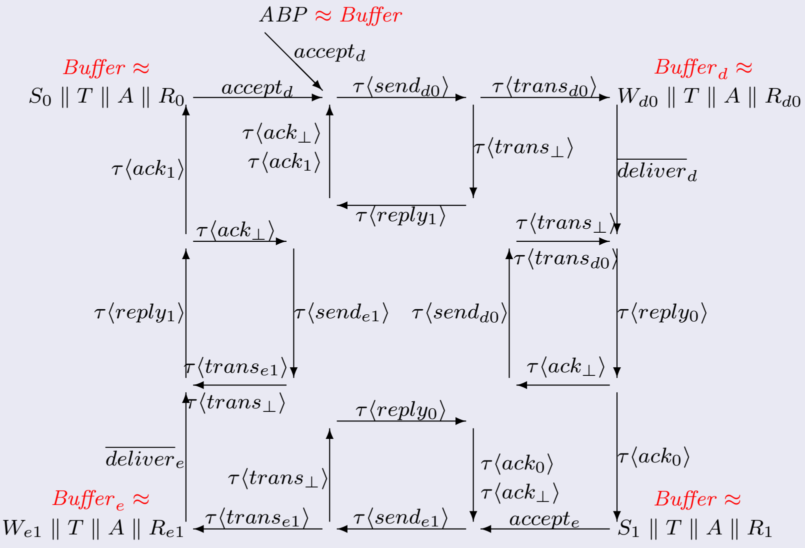
□

Correctness of the ABP II

Proof (continued).



Proof (continued).



Outline

Duplication of Messages I

Duplication of messages can be modelled as follows:

$$Trans = \sum_{f\in F} send_f.(\underbrace{\overline{trans_f}.Trans}_{\text{successful}} + \underbrace{\overline{trans_\perp}.Trans}_{\text{error}} + \underbrace{\overline{trans_f}.\overline{trans_f}.Trans}_{\text{duplication}})$$

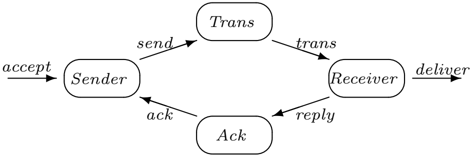
$$Ack = \sum_{b\in\{0,1\}} reply_b.(\underbrace{\overline{ack_b}.Ack}_{\text{successful}} + \underbrace{\overline{ack_\perp}.Ack}_{\text{error}} + \underbrace{\overline{ack_b}.\overline{ack_b}.Ack}_{\text{duplication}})$$

Duplication of Messages I

Duplication of messages can be modelled as follows:

$$\begin{aligned}
Trans &= \sum_{f \in F} send_f.(\underbrace{\overline{trans_f}.Trans}_{\text{successful}} + \underbrace{\overline{trans_\perp}.Trans}_{\text{error}} + \underbrace{\overline{trans_f}.\overline{trans_f}.Trans}_{\text{duplication}}) \\
Ack &= \sum_{b \in \{0,1\}} reply_b.(\underbrace{\overline{ack_b}.Ack}_{\text{successful}} + \underbrace{\overline{ack_\perp}.Ack}_{\text{error}} + \underbrace{\overline{ack_b}.\overline{ack_b}.Ack}_{\text{duplication}})
\end{aligned}$$

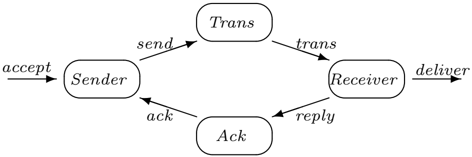
Duplication of Messages II



Now the ABP behaves as follows (without restriction):

$$Sender_0 \parallel Trans \parallel Ack \parallel Receiver_0$$

Duplication of Messages II

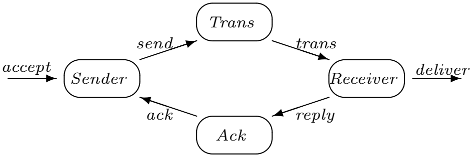


Now the ABP behaves as follows (without restriction):

$$Sender_b = \sum_{d \in D} \color{red}{accept_d}\color{black}{.Send_{db}}$$

$$Sender_0 \parallel Trans \parallel Ack \parallel Receiver_0$$
$$\downarrow accept_d$$
$$Send_{d0} \parallel Trans \parallel Ack \parallel Receiver_0$$

Duplication of Messages II



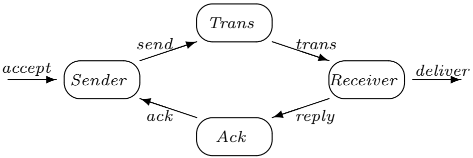
Now the ABP behaves as follows (without restriction):

$$Send_{db} = \overline{send_{db}}.Wait_{db}$$

$$Trans = \sum_{f \in F} send_f.(\overline{trans_f}.Trans + \overline{trans_\perp}.Trans + \overline{trans_f}.\overline{trans_f}.Trans)$$

$$Send_{d0} \parallel Trans \parallel Ack \parallel Receiver_0$$
$$\downarrow \tau\langle send_{d0}\rangle$$
$$Wait_{d0} \parallel (\ldots + \overline{trans_{d0}}.\overline{trans_{d0}}.Trans) \parallel Ack \parallel Receiver_0$$

Duplication of Messages II



Now the ABP behaves as follows (without restriction):

$$Receiver_b = \sum_{d \in D} trans_{db}.Reply_{db} + \ldots$$

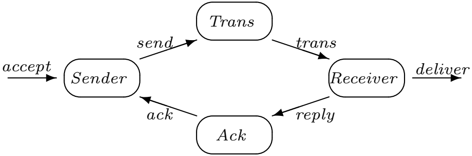
$$Trans = \sum_{f \in F} send_f.(\overline{trans_f}.Trans + \overline{trans_\perp}.Trans + \overline{trans_f}.\overline{trans_f}.Trans)$$

$$Wait_{d0} \parallel (\ldots + \overline{trans_{d0}}.\overline{trans_{d0}}.Trans) \parallel Ack \parallel Receiver_0$$

$$\downarrow \tau\langle trans_{d0}\rangle$$

$$Wait_{d0} \parallel \overline{trans_{d0}}.Trans \parallel Ack \parallel Reply_{d0}$$

Duplication of Messages II

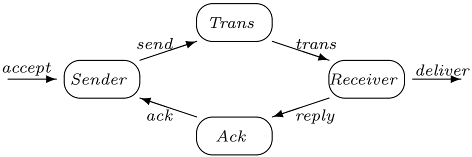


Now the ABP behaves as follows (without restriction):

$$Reply_{db} = \overline{deliver_d}.\overline{reply_b}.Receiver_{1-b}$$

$$Wait_{d0} \parallel \overline{trans_{d0}}.Trans \parallel Ack \parallel Reply_{d0}$$
$$\downarrow \overline{deliver_d}$$
$$Wait_{d0} \parallel \overline{trans_{d0}}.Trans \parallel Ack \parallel \overline{reply_0}.Receiver_1$$

Duplication of Messages II



Now the ABP behaves as follows (without restriction):

$$Reply_{db} = \overline{deliver_d}.\textcolor{red}{\overline{reply_b}.}Receiver_{1-b}$$

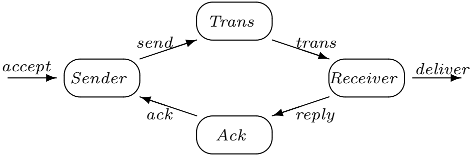
$$Ack = \sum_{b \in \{0,1\}} \textcolor{red}{reply_b}.(\overline{ack_b}.Ack + \overline{ack_\perp}.Ack + \overline{ack_b}.\overline{ack_b}.Ack)$$

$$Wait_{d0} \parallel \overline{trans_{d0}}.Trans \parallel Ack \parallel \overline{reply_0}.Receiver_1$$

$$\downarrow \tau\langle reply_0 \rangle$$

$$Wait_{d0} \parallel \overline{trans_{d0}}.Trans \parallel (\ldots + \overline{ack_0}.\overline{ack_0}.Ack) \parallel Receiver_1$$

Duplication of Messages II



Now the ABP behaves as follows (without restriction):

$$Wait_{db} = ack_b.Sender_{1-b} + ack_{1-b}.Send_{db} + ack_\perp.Send_{db}$$

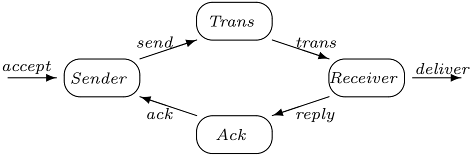
$$Ack = \sum_{b\in\{0,1\}} reply_b.(\overline{ack_b}.Ack + \overline{ack_\perp}.Ack + \overline{ack_b}.\overline{ack_b}.Ack)$$

$$Wait_{d0} \parallel \overline{trans_{d0}}.Trans \parallel (\ldots + \overline{ack_0}.\overline{ack_0}.Ack) \parallel Receiver_1$$

$$\downarrow \tau\langle ack_0\rangle$$

$$Sender_1 \parallel \overline{trans_{d0}}.Trans \parallel \overline{ack_0}.Ack \parallel Receiver_1$$

Duplication of Messages II



Now the ABP behaves as follows (without restriction):

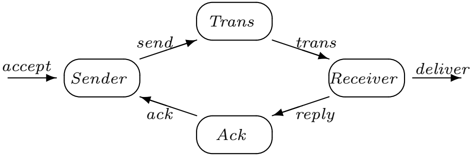
$$Receiver_b = \ldots + \sum_{d \in D} trans_{d(1-b)}.\overline{reply_{1-b}}.Receiver_b$$

$$Trans = \sum_{f \in F} send_f.(\overline{trans_f}.Trans + \overline{trans_\perp}.Trans + \overline{trans_f}.\overline{trans_f}.Trans)$$

$$Sender_1 \parallel \overline{trans_{d0}}.Trans \parallel \overline{ack_0}.Ack \parallel Receiver_1$$

$$\downarrow \tau\langle trans_{d0}\rangle$$

$$Sender_1 \parallel Trans \parallel \overline{ack_0}.Ack \parallel \overline{reply_0}.Receiver_1$$

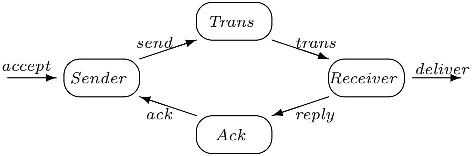


Now the ABP behaves as follows (without restriction):

$$Sender_b = \sum_{d \in D} \textcolor{red}{accept_d}.Send_{db}$$

$$Sender_1 \parallel Trans \parallel \overline{ack_0}.Ack \parallel \overline{reply_0}.Receiver_1$$
$$\downarrow accept_e$$
$$Send_{e1} \parallel Trans \parallel \overline{ack_0}.Ack \parallel \overline{reply_0}.Receiver_1$$

Duplication of Messages II



Now the ABP behaves as follows (without restriction):

$$Send_{db} = \overline{send_{db}}.Wait_{db}$$

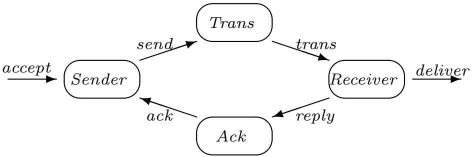
$$Trans = \sum_{f \in F} send_f.(\overline{trans_f}.Trans + \overline{trans_\perp}.Trans + \overline{trans_f}.\overline{trans_f}.Trans)$$

$$Send_{e1} \parallel Trans \parallel \overline{ack_0}.Ack \parallel \overline{reply_0}.Receiver_1$$

$$\downarrow \tau\langle send_{e1}\rangle$$

$$Wait_{e1} \parallel (\ldots + \overline{trans_{e1}}.\overline{trans_{e1}}.Trans) \parallel \overline{ack_0}.Ack \parallel \overline{reply_0}.Receiver_1$$

Duplication of Messages II



Now the ABP behaves as follows (without restriction):

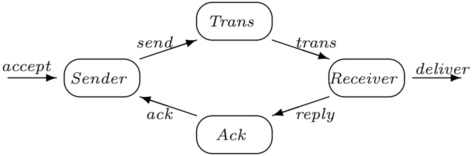
$$Wait_{db} = ack_b.Sender_{1-b} + ack_{1-b}.Send_{db} + ack_\perp.Send_{db}$$

$$Ack = \sum_{b \in \{0,1\}} reply_b.(\overline{ack_b}.Ack + \overline{ack_\perp}.Ack + \overline{ack_b}.\overline{ack_b}.Ack)$$

$$Wait_{e1} \parallel (\ldots + \overline{trans_{e1}}.\overline{trans_{e1}}.Trans) \parallel \overline{ack_0}.Ack \parallel \overline{reply_0}.Receiver_1$$

$$\downarrow \tau \langle ack_0 \rangle$$

$$Send_{e1} \parallel (\ldots + \overline{trans_{e1}}.\overline{trans_{e1}}.Trans) \parallel Ack \parallel \overline{reply_0}.Receiver_1$$



Now the ABP behaves as follows (without restriction):

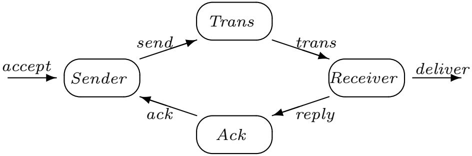
$$Receiver_b = \ldots + \sum_{d \in D} trans_{d(1-b)}.\overline{reply_{1-b}}.Receiver_b$$

$$Ack = \sum_{b \in \{0,1\}} reply_b.(\overline{ack_b}.Ack + \overline{ack_\perp}.Ack + \overline{ack_b}.\overline{ack_b}.Ack)$$

$$Send_{e1} \parallel (\ldots + \overline{trans_{e1}}.\overline{trans_{e1}}.Trans) \parallel Ack \parallel \overline{reply_0}.Receiver_1$$

$$\downarrow \tau\langle reply_0 \rangle$$

$$Send_{e1} \parallel (\ldots + \overline{trans_{e1}}.\overline{trans_{e1}}.Trans) \parallel (\ldots + \overline{ack_0}.\overline{ack_0}.Ack) \parallel Receiver_1$$



Now the ABP behaves as follows (without restriction):

$$Receiver_b = \sum_{d \in D} trans_{db}.Reply_{db} + \ldots$$

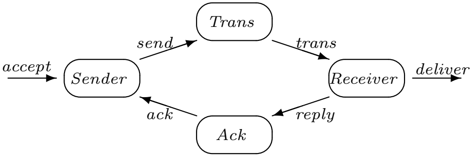
$$Trans = \sum_{f \in F} send_f.(\overline{trans_f}.Trans + \overline{trans_\perp}.Trans + \overline{trans_f}.\overline{trans_f}.Trans)$$

$$Send_{e1} \parallel (\ldots + \overline{trans_{e1}}.\overline{trans_{e1}}.Trans) \parallel (\ldots + \overline{ack_0}.\overline{ack_0}.Ack) \parallel Receiver$$

$$\downarrow \tau\langle trans_{e1} \rangle$$

$$Send_{e1} \parallel \overline{trans_{e1}}.Trans \parallel (\ldots + \overline{ack_0}.\overline{ack_0}.Ack) \parallel Reply_{e1}$$

Duplication of Messages II

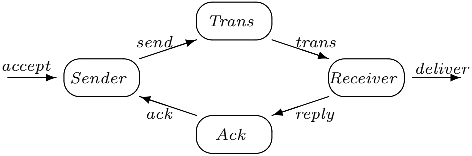


Now the ABP behaves as follows (without restriction):

$$Reply_{db} = \overline{deliver_d}.\overline{reply_b}.Receiver_{1-b}$$

$$Send_{e1} \parallel \overline{trans_{e1}}.Trans \parallel (\ldots + \overline{ack_0}.\overline{ack_0}.Ack) \parallel Reply_{e1}$$
$$\downarrow \overline{deliver_e}$$
$$Send_{e1} \parallel \overline{trans_{e1}}.Trans \parallel (\ldots + \overline{ack_0}.\overline{ack_0}.Ack) \parallel \overline{reply_1}.Receiver_0$$

Duplication of Messages II



Now the ABP behaves as follows (without restriction):

$$Send_{e1} \parallel \overline{trans_{e1}}.Trans \parallel (\ldots + \overline{ack_0}.\overline{ack_0}.Ack) \parallel \overline{reply_1}.Receiver_0$$

$$\downarrow$$

$$?$$

Deadlock $\Longrightarrow$ ABP cannot handle this