

Modeling Concurrent and Probabilistic Systems

Lecture 11: Extensions of the Alternating Bit Protocol

Joost-Pieter Katoen Thomas Noll

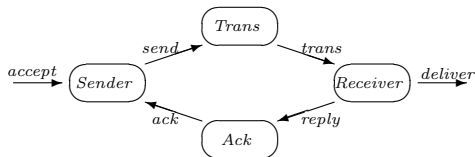
Software Modeling and Verification Group
RWTH Aachen University
`noll@cs.rwth-aachen.de`

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

Winter Semester 2007/08

- 1 Repetition: The Alternating Bit Protocol
- 2 Handling Duplication of Messages
- 3 Concluding Remarks
- 4 Modeling Mobile Concurrent Systems

Repetition: The Alternating Bit Protocol



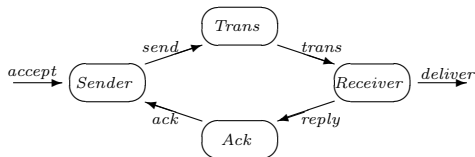
The **overall system** is given by

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

where

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

Repetition: Implementation of 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 (\Rightarrow “Alternating Bit Protocol”).

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

$$Sender = Sender_0$$

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

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

$$Wait_{db} = \underbrace{ack_b. Sender_{1-b}}_{\text{successful}} + \underbrace{ack_{1-b}. Send_{db} + ack_{\perp}. Send_{db}}_{\text{error}}$$

Repetition: Implementation of 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 $d \in D$:

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

Repetition: Correctness of ABP I

Theorem

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

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; $S = \text{Sender}$, $W = \text{Wait}$, $T = \text{Trans}$, $A = \text{Ack}$,
 $R = \text{Receiver/Reply}$, $d, e \in D$; without restrictions)
- 2 Show that $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \approx Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$
- 3 $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \not\stackrel{\tau}{\rightarrow}$ and $Buffer(\overrightarrow{accept}, \overrightarrow{deliver}) \not\stackrel{\tau}{\rightarrow}$
 $\implies ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \simeq Buffer(\overrightarrow{accept}, \overrightarrow{deliver})$



Repetition: Duplication of Messages

Duplication of messages can be modelled as follows:

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

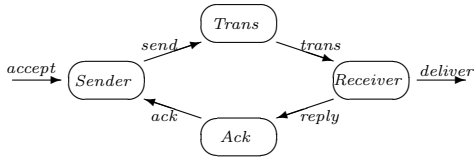
Observation: with the original definition of *Sender* and *Receiver*, **deadlocks** are possible

- 1 Repetition: The Alternating Bit Protocol
- 2 Handling Duplication of Messages
- 3 Concluding Remarks
- 4 Modeling Mobile Concurrent Systems

Handling Duplication of Messages

- **Idea:** allow *Sender* and *Receiver* to **transmit \perp frames**:
 - $Receiver \xrightarrow{reply} \perp$: message not received
 - $Sender \xrightarrow{send} \perp$: acknowledgment not received
- Allows to **distinguish corrupted and duplicated frames**

Modified Implementation of Sender



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

$$Sender = Sender_0$$

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

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

$$Wait_{db} = \underbrace{ack_b . Sender_{1-b}}_{\text{successful}} + \underbrace{ack_{\perp} . Send_{db}}_{\text{error, restart}} + \underbrace{ack_{1-b} . Wait_{db}}_{\text{duplication, ignore}}$$

Modified Implementation of Receiver

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

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

The Overall System

$$\begin{aligned} ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \\ = \text{new } L \text{ (} Sender \parallel Trans \parallel Ack \parallel Receiver \text{)} \end{aligned}$$

$$Sender = Sender_0$$

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

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

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

$$Receiver = Receiver_0$$

$$\begin{aligned} Receiver_b = & \sum_{d \in D} \overrightarrow{trans_{db}} . Reply_{db} \\ & + \overrightarrow{trans_{\perp}} . \overrightarrow{reply_{\perp}} . Receiver_b \\ & + \sum_{d \in D} \overrightarrow{trans_{d(1-b)}} . Receiver_b \end{aligned}$$

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

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

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

$$\begin{aligned} \text{where } L := & \{ \overrightarrow{send_{db}}, \overrightarrow{trans_{db}}, \overrightarrow{reply_b}, \overrightarrow{ack_b} \mid db \in F \} \\ & \cup \{ \overrightarrow{send_{\perp}}, \overrightarrow{trans_{\perp}}, \overrightarrow{reply_{\perp}}, \overrightarrow{ack_{\perp}} \} \end{aligned}$$

Again:

Theorem 11.1

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

Proof.

on the board

($S = \text{Sender/Send}$, $W = \text{Wait}$, $T = \text{Trans}$, $A = \text{Ack}$,
 $R = \text{Receiver/Reply}$, $d, e \in D$; without restrictions)



- 1 Repetition: The Alternating Bit Protocol
- 2 Handling Duplication of Messages
- 3 Concluding Remarks
- 4 Modeling Mobile Concurrent Systems

Concluding Remarks

- Handling **loss of messages**: by introducing **timeouts** (see 7th exercise sheet)
- **Validity of correctness proof** (τ -cycles in *ABP*, but not in *Buffer*)?

Simplest case:

$$A(a) = \tau.A + a.\text{nil} \quad \simeq \quad B(a) = \tau.a.\text{nil}$$

Even more: every LTS containing τ -cycles is observationally congruent to one without τ -cycles

- There are notions of equivalence which distinguish **divergent** (τ -cycles) and **convergent** (no τ -cycles) processes
- **But:**
 - they are more complicated than standard bisimulation
 - (weak) bisimulation allows the proportion between the speeds of processes to vary unboundedly – why not infinite?
 - if convergence is essential, it can be assured separately

- 1 Repetition: The Alternating Bit Protocol
- 2 Handling Duplication of Messages
- 3 Concluding Remarks
- 4 Modeling Mobile Concurrent Systems

Observation: CCS imposes a **static communication structure**: if $P, Q \in \text{Proc}$ want to communicate, then both must syntactically refer to the same action name

\Rightarrow every potential communication partner known beforehand,
no dynamic passing of communication links

\Rightarrow **no mobility**

Goal: develop calculus in the spirit of CCS which supports mobility

\Rightarrow **π -calculus**

Example 11.2 (Dynamic access to resources)

- Server S controls access to printer P
- Client C wishes to use P
- In **CCS**: P and C must share some action name a
 $\implies C$ could access P without being granted it by S
- In **π -calculus** :
 - initially only S has access to P (using link a)
 - using another link b , C can request access to P
- Formally:

$$\begin{array}{l}
 \underbrace{\bar{b}\langle a \rangle . S'}_S \parallel \underbrace{b(c) . \bar{c}\langle d \rangle . C'}_C \parallel \underbrace{a(e) . P}_P \\
 \xrightarrow{\tau} S' \parallel \bar{a}\langle d \rangle . C' \parallel a(e) . P' \\
 \xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]
 \end{array}$$

- a : link to P
- b : link between S and C
- c : “placeholder” for a
- d : data to be printed
- e : “placeholder” for d

Example 11.2 (Dynamic access to resources; continued)

- Different rôles of action name a :
 - in interaction between S and C :
object transferred from S to C
 - in interaction between C and P :
name of communication link
- Intuitively, names represent access rights:
 - a : for P
 - b : for S
 - d : for data to be printed
- If a is only way to access P
 $\implies P$ “moves” from S to C