Modeling Concurrent and Probabilistic Systems

Lecture 11: Extensions of the Alternating Bit Protocol

Joost-Pieter Katoen   Thomas Noll

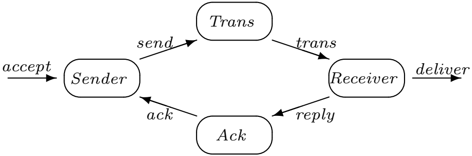
Software Modeling and Verification Group
RWTH Aachen University
noll@cs.rwth-aachen.de

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

Winter Semester 2007/08

Outline

Repetition: The Alternating Bit Protocol

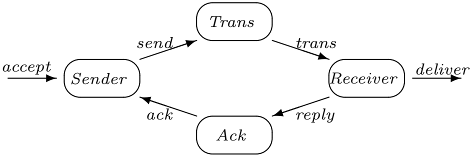


The overall system is given by

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

where

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



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 $d \in D$:

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

Repetition: Implementation of Receiver

Receiver gets frames of the form db or $\bot$. 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 $d \in D$:

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

Repetition: Correctness of ABP I

Theorem

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

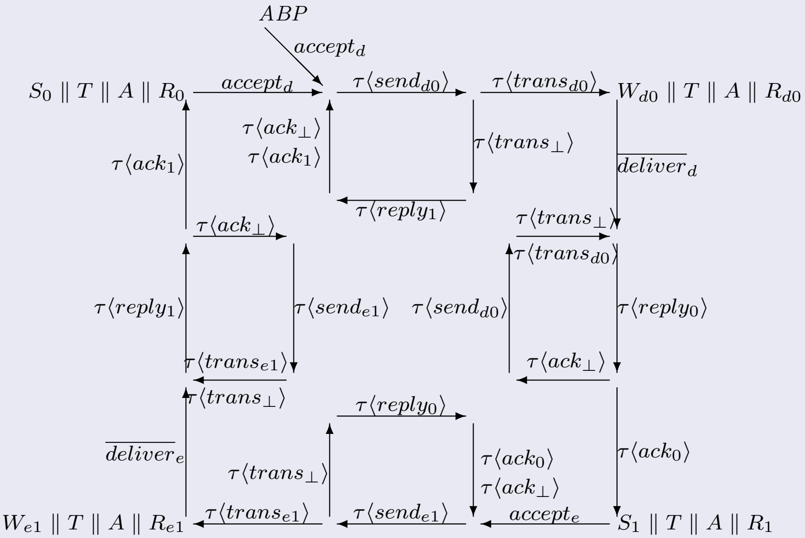
Remark: because of internal τ-steps in ABP, $ABP \sim Buffer$ cannot hold.

Proof.

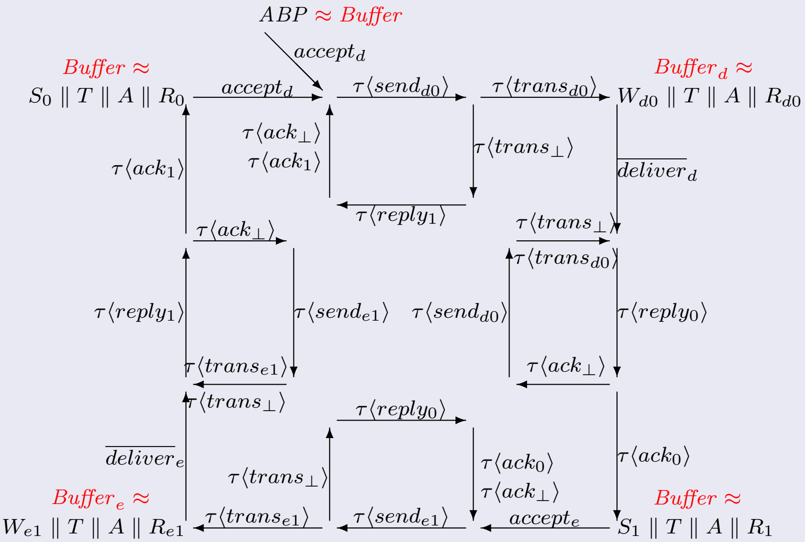
1. Construct transition system of $ABP(\overrightarrow{accept}, \overrightarrow{deliver})$ (next slide; $S = Sender$, $W = Wait$, $T = Trans$, $A = Ack$, $R = Receiver/Reply$, $d, e \in D$; without restrictions)
2. Show that $ABP(\overrightarrow{accept}, \overrightarrow{deliver}) \approx Buffer(\overrightarrow{accept},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

□

Proof (continued).



Proof (continued).



Repetition: Duplication of Messages

Duplication of messages can be modelled as follows:

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

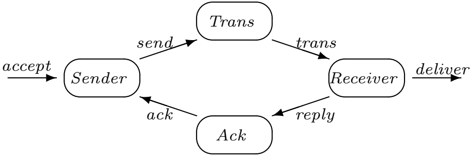
Observation: with the original definition of *Sender* and *Receiver*, deadlocks are possible

Outline

Handling Duplication of Messages

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

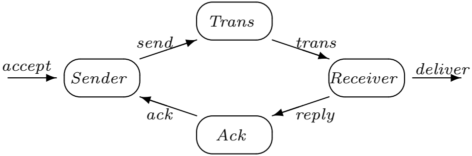
Modified Implementation of Sender



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

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

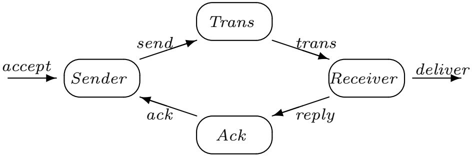
Modified Implementation of Sender



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

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

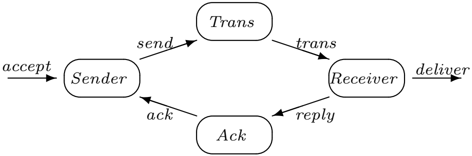
Modified Implementation of Sender



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

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

Modified Implementation of Sender



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

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

Modified Implementation of Receiver

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

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

Modified Implementation of Receiver

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

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

Modified Implementation of Receiver

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

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

The Overall System

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

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

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

$$\begin{aligned}
Trans &= \textstyle\sum_{f \in F} send_f . (\overline{trans_f} . Trans + \overline{trans_\perp} . Trans + \\
&\qquad \overline{trans_f} . \overline{trans_f} . Trans) \\
Ack &= \textstyle\sum_{b \in \{0,1\}} reply_b . (\overline{ack_b} . Ack + \overline{ack_\perp} . Ack + \overline{ack_b} . \overline{ack_b} . Ack)
\end{aligned}$$

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

Correctness of the Modification

Again:

Theorem 11.1

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

Proof.

on the board
($S = Sender/Send$, $W = Wait$, $T = Trans$, $A = Ack$,
$R = Receiver/Reply$, $d, e \in D$; without restrictions) □

Correctness of the Modification

Again:

Theorem 11.1

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

Proof.

on the board
$(S = Sender/Send,\ W = Wait,\ T = Trans,\ A = Ack,$
$R = Receiver/Reply,\ d, e \in D$; without restrictions) $\square$

Outline

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:

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

  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

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:

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

  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

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:

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

  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

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:

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

  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

Outline

Mobility in Concurrent Systems I

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

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

$\Longrightarrow$ no mobility

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

$\Longrightarrow$ π-calculus

Mobility in Concurrent Systems I

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

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

$\Longrightarrow$ no mobility

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

$\Longrightarrow$ π-calculus

Mobility in Concurrent Systems I

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

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

$\Longrightarrow$ no mobility

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

$\Longrightarrow$ π-calculus

Mobility in Concurrent Systems I

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

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

$\Longrightarrow$ no mobility

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

$\Longrightarrow$ 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
  $\Longrightarrow$ 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:

$$\underbrace{\overline{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P}$$
$$\xrightarrow{\tau} S' \parallel \overline{a}\langle d\rangle.C' \parallel a(e).P'$$
$$\xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]$$

- 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)

- Server S controls access to printer P
- Client C wishes to use P
- In CCS: P and C must share some action name a
  $\Longrightarrow$ 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:

$$\underbrace{\bar{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\bar{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P}$$
$$\xrightarrow{\tau} S' \parallel \bar{a}\langle d\rangle.C' \parallel a(e).P'$$
$$\xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]$$

- 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)

- Server S controls access to printer P
- Client C wishes to use P
- In CCS: P and C must share some action name a
  $\Longrightarrow$ 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:

$$\underbrace{\overline{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P}$$
$$\xrightarrow{\tau} S' \parallel \overline{a}\langle d\rangle.C' \parallel a(e).P'$$
$$\xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]$$

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

Mobility in Concurrent Systems II

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
  $\Longrightarrow$ 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:

$$\underbrace{\overline{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P}$$

$$\xrightarrow{\tau} S' \parallel \overline{a}\langle d\rangle.C' \parallel a(e).P'$$

$$\xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]$$

- 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)

- 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:

$$\underbrace{\overline{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P}$$
$$\xrightarrow{\tau} S' \parallel \overline{a}\langle d\rangle.C' \parallel a(e).P'$$
$$\xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]$$

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

Mobility in Concurrent Systems II

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
  $\Longrightarrow$ 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:

$$\underbrace{\overline{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P}$$
$$\xrightarrow{\tau} S' \parallel \overline{a}\langle d\rangle.C' \parallel a(e).P'$$
$$\xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]$$

- 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
  $\Longrightarrow$ P "moves" from S to C

Mobility in Concurrent Systems III

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

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
  $\Longrightarrow$ P "moves" from S to C