Software Modeling and Verification
Lehrstuhl für Informatik 2
RWTH Aachen University

Prof. Dr. Ir. J.-P. Katoen
Priv.-Doz. Dr. T. Noll
T. Han, M. Neuhäußer

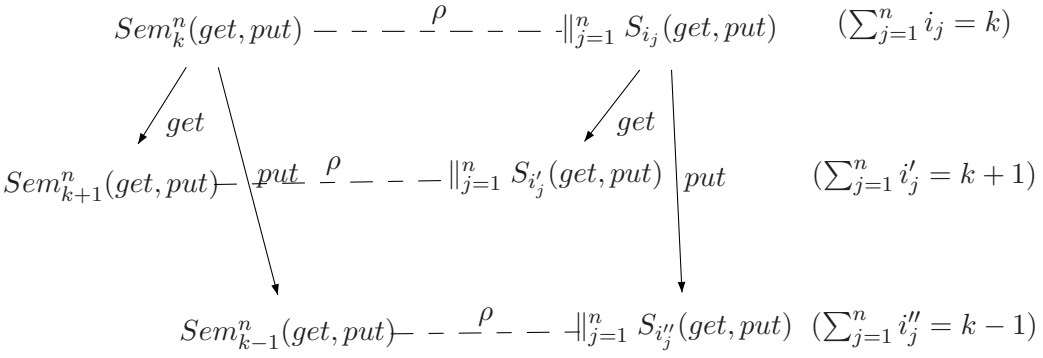
Modeling Concurrent and Probabilistic Systems

Winter Term 07/08

– Solution 3 –

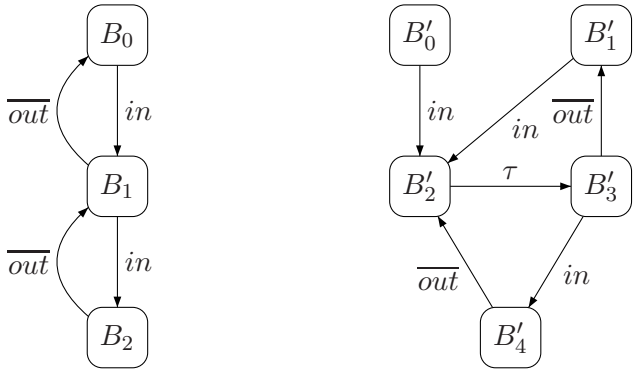
Exercise 1 **(4 + 1 points)**

a) We show that $\rho = (Sem_0^n(get, put), S^n(get, put)) \cup \{(Sem_k^n(get, put), S_{i_1}(get, put) \| \ldots \| S_{i_n}(get, put)) \mid 0 \leq k \leq n, \sum_{j=1}^{n} i_j = k\}$ is a strong bisimulation:



This is the most general case (for $k = 0$, there are no *put*-transitions, and for $k = n$ no *get*-transitions). The states on the right-hand side actually represent all different states for which the summation condition holds – they are equivalent with respect to ρ.

b) See Example 2.3 for the specifications of the sequential and parallel two-place buffer.
The corresponding LTSs are:



Assuming $B_0 \sim B'_0$, it then follows that $B_1 \sim B'_2$. However, B'_2 has an outgoing τ-transition while B_1 does not, therefore $B_1 \not\sim B'_2$. From this it follows that the two LTSs are not strongly bisimilar.

Exercise 2 (4 points)

a) $P + \mathsf{nil} \sim P$:

- $P + \mathsf{nil} \xrightarrow{a} P'$ implies $P \xrightarrow{a} P'$ for all $a \in A$ and clearly $P' \sim P'$.
- $P \xrightarrow{a} P'$ obviously implies $P + \mathsf{nil} \xrightarrow{a} P'$ for all $a \in A$ and $P' \sim P'$.

Hence $P + \mathsf{nil} \sim P$.

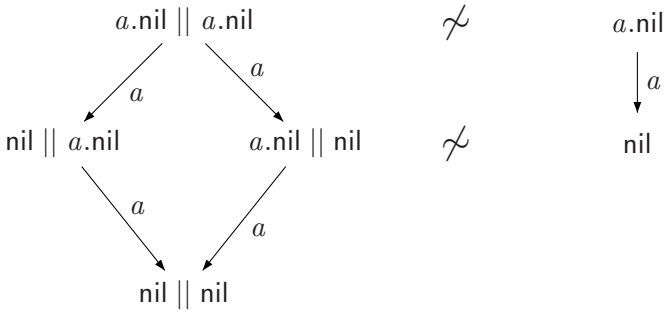
b) $P + P \sim P$:

- $P + P \xrightarrow{a} P'$ implies $P \xrightarrow{a} P'$ for all $a \in A$ and clearly $P' \sim P'$.
- $P \xrightarrow{a} P'$ obviously implies $P + P \xrightarrow{a} P'$ for all $a \in A$ and $P' \sim P'$.

Hence $P + P \sim P$.

c) $P \parallel (Q \parallel R) \sim (P \parallel Q) \parallel R$: Choose $\rho = \{(P \parallel (Q \parallel R), (P \parallel Q) \parallel R \mid P, Q, R \in \mathit{Proc}\}$. ρ is a strong bisimulation.

d) $P \parallel P \not\sim P$: A counterexample is the process definition $P = a.\mathsf{nil}$. The two induced LTS are as follows:



Exercise 3 (3 points)

a) Problem: "race conditions" between external user actions and internal τ-steps

Intended:

$$
\begin{aligned}
&\mathsf{new}\ \vec{\alpha}(\mathit{Open}(\vec{a}) \| \mathit{Unlocked}(\vec{b}) \| \mathit{Key}(\vec{c})) \\
\xrightarrow{\mathit{close}} &\mathsf{new}\ \vec{\alpha}(\mathit{Closed}(\vec{a}) \| \mathit{Unlocked}(\vec{b}) \| \mathit{Key}(\vec{c})) \\
\xrightarrow{\mathit{open}} &\mathsf{new}\ \vec{\alpha}((\mathit{isLocked}.\mathit{Closed}(\vec{a}) + \mathit{isUnlocked}.\mathit{Open}(\vec{a})) \| \mathit{Unlocked}(\vec{b}) \| \mathit{Key}(\vec{c})) \\
\xrightarrow{\tau} &\mathsf{new}\ \vec{\alpha}(\mathit{Open}(\vec{a}) \| \mathit{Unlocked}(\vec{b}) \| \mathit{Key}(\vec{c})) \\
\longrightarrow &\ldots
\end{aligned}
$$

But:

$$
\begin{aligned}
&\mathsf{new}\,\vec{\alpha}((isLocked.Closed(\vec{a}) + isUnlocked.Open(\vec{a}))\|Unlocked(\vec{b})\|Key(\vec{c})) \\
\overset{pressed}{\longrightarrow}\ &\mathsf{new}\,\vec{\alpha}((isLocked.Closed(\vec{a}) + isUnlocked.Open(\vec{a})) \\
&\|Unlocked(\vec{b})\|\overline{activate}.Key(\vec{c})) \\
\overset{\tau}{\longrightarrow}\ &\mathsf{new}\,\vec{\alpha}((isLocked.Closed(\vec{a}) + isUnlocked.Open(\vec{a})) \\
&\|(isOpen.\overline{alarm}.Unlocked(\vec{b}) + isClosed.Locked(\vec{b}))\|Key(\vec{c})) \\
\overset{pressed}{\longrightarrow}\ &\mathsf{new}\,\vec{\alpha}((isLocked.Closed(\vec{a}) + isUnlocked.Open(\vec{a})) \\
&\|(isOpen.\overline{alarm}.Unlocked(\vec{b}) + isClosed.Locked(\vec{b}))\|\overline{activate}.Key(\vec{c}))
\end{aligned}
$$

$\Rightarrow$ deadlock situation

b) Idea: Introduce an additional synchronization process

$$
\begin{aligned}
Sync(\vec{d}) = \underbrace{OPEN}_{\text{user action}}.\ \underbrace{\overline{open}}_{\text{internal action}}\ .\ \underbrace{ack}_{\text{acknowledgement}}\ .Sync(\vec{d})+ \\
CLOSE.\overline{close}.ack.Sync(\vec{d})+ \\
PRESS.\overline{pressed}.ack.Sync(\vec{d})
\end{aligned}
$$

$\overline{ack}$ has to be introduced in other processes of the system:

$$
\begin{array}{lcl}
Door(\vec{a}) & = & Open(\vec{a}) \\
Open(\vec{a}) & = & \overline{isOpen}.Open(\vec{a}) + close.\overline{ack}.Closed(\vec{a}) + open.\overline{ack}.Open(\vec{a}) \\
Closed(\vec{a}) & = & \overline{isClosed}.Closed(\vec{a})+ \\
 & & open.(isLocked.\overline{ack}.Closed(\vec{a}) + isUnlocked.\overline{ack}.Open(\vec{a})) + close.\overline{ack}.Closed(\vec{a}) \\
 & & \\
Locker(\vec{b}) & = & Unlocked(\vec{b}) \\
Unlocked(\vec{b}) & = & \overline{isUnlocked}.Unlocked(\vec{b})+ \\
 & & activate.(isOpen.\overline{ALARM}.\overline{ack}.Unlocked(\vec{b}) + isClosed.Locked(\vec{b})) \\
Locked(\vec{b}) & = & \overline{isLocked}.Locked(\vec{b}) + activate.\overline{ack}.Unlocked(\vec{b}) \\
 & & \\
Key(\vec{c}) & = & pressed.\overline{activate}.Key(\vec{c}) \\
 & & \\
System(\vec{e}) & = & \mathrm{new}\vec{d}\,(Door(\vec{a})\|Locker(\vec{b}\|Key(\vec{c})\|Sync(\vec{d}))
\end{array}
$$

where $\vec{d} = open, close, ack, pressed, isOpen, isClosed, isLocked, isUnlocked, activate.$

Exercise 4 **(1 + 3 points)**

$P \in Proc$ **well terminating**, if for every $P \rightarrow^* P'$,

$$
(*)\left\{\begin{array}{ll} (i) & P' \overset{done}{\not\longrightarrow} \quad \text{and} \\ (ii) & P' \overset{\overline{done}}{\longrightarrow}\Rightarrow P' \sim \overline{done}.\mathsf{nil} \end{array}\right\}
$$

a)

nil well terminating: trivial

Let $P_1, P_2 \in Proc$ be well terminating.

$\alpha.P_1 (\alpha \notin \{done, \overline{done}\})$ w.t.:
Let $\alpha.P_1 \rightarrow^n P' \quad (n \geq 0)$
If $n = 0 : P' = \alpha.P_1 \xrightarrow{done,\overline{done}}\!\!\!\!\!\!\not\Rightarrow (*)$
If $n \geq 1 : P_1 \rightarrow^{n-1} P' \Rightarrow (*)$

$P_1 + P_2$ w.t.:
Let $P_1 + P_2 \rightarrow^* P'$
$\Rightarrow P_1 \rightarrow^* P'$ or $P_2 \rightarrow^* P'$
$\Rightarrow (*)$

$\mathsf{new}\,\vec{\alpha}\,P_1$ w.t.:
Let $\mathsf{new}\,\vec{\alpha}\,P_1 \rightarrow^* P'$
$\Rightarrow$ ex. Q' with $P' = \mathsf{new}\,\vec{\alpha}\,Q'$ and $P_1 \rightarrow^* Q'$
$\Rightarrow (*)$ applies to Q'
$\Rightarrow (*)$ applies to P' (even if $done \in \vec{\alpha}$)

Well-termination is **not** preserved by parallel composition:
$P = \overline{done}.\mathsf{nil}$
but $P \| P \xrightarrow{\overline{done}} \mathsf{nil} \| \overline{done}.\mathsf{nil} \xrightarrow{\overline{done}} \mathsf{nil} \| \mathsf{nil}$
$\Rightarrow \overline{done}.\mathsf{nil} \| \overline{done}.\mathsf{nil} \not\sim \overline{done}.\mathsf{nil}$

b)

$Done = \overline{done}.\mathsf{nil}$

$P\,Seq\,Q = \mathsf{new}\,d(P[done \mapsto d] \| d.Q)$

$P\,Par\,Q = \mathsf{new}\,d_1, d_2(P[done \mapsto d_1]) \| Q[done \mapsto d_2]) \| d_1.d_2.Done)$

Seq, Par preserve well-termination:
Induction on the length of the computation