Modeling Concurrent and Probabilistic Systems

Lecture 4: Definition of Strong Bisimulation

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: Desired Properties of Equivalence

Wanted: a "feasible" (i.e., efficiently decidable) semantic equivalence between CCS processes which

1. identifies processes whose LTSs coincide,
2. implies trace equivalence, i.e., considers two processes equivalent only if both can execute the same actions sequences (formal definition later), and
3. is a congruence, i.e., allows to replace a subprocess by an equivalent counterpart without changing the overall semantics of the system (formal definition later).
4. is deadlock sensitive, i.e., if $P \cong Q$ and if P has a w-deadlock, then Q has a w-deadlock (and vice versa, by equivalence).

Formally: we are looking for a deadlock-sensitive congruence relation $\cong \subseteq Prc \times Prc$ such that

$$LTS(P) = LTS(Q) \implies P \cong Q \implies Tr(P) = Tr(Q)$$

Repetition: Trace Equivalence

Definition (Trace language)

For every $P \in Prc$, let

$$Tr(P) := \{w \in Act^* \mid \text{ex. } P' \in Prc \text{ such that } P \xrightarrow{w}{}^* P'\}$$

be the trace language of P.

$P, Q \in Prc$ are called trace equivalent if $Tr(P) = Tr(Q)$.

Example (One-place buffer)

$B(in, out) = in.\overline{out}.B(in, out)$

$\Longrightarrow \quad Tr(B) = (in \cdot \overline{out})^* \cdot (in + \varepsilon)$

Repetition: CCS Congruences

Goal: replacing a subcomponent of a system by an equivalent process should yield an equivalent systems

$\Longrightarrow$ modular system development

Definition (CCS congruence)

An equivalence relation $\cong \subseteq Prc \times Prc$ is said to be a CCS congruence if it is preserved by the CCS constructs; that is, if $P \cong Q$ then

$$\begin{aligned} \alpha.P &\cong \alpha.Q \\ P + R &\cong Q + R \\ R + P &\cong R + Q \\ P \parallel R &\cong Q \parallel R \\ R \parallel P &\cong R \parallel Q \\ \mathsf{new}\, a\, P &\cong \mathsf{new}\, a\, Q \end{aligned}$$

for every $\alpha \in Act$, $R \in Prc$, and $a \in N$.

Repetition: Deadlocks

Definition (Deadlock)

Let $P, Q \in Prc$ and $w \in Act^*$ such that $P \xrightarrow{w}{}^* Q$ and $Q \not\longrightarrow$.
Then Q is called a w-deadlock of P.

- Thus $P := a.b.\mathsf{nil} + a.\mathsf{nil}$ has an a-deadlock, in contrast to $Q := a.b.\mathsf{nil}$.
- Such properties are important since it can be crucial that a certain communication is eventually possible.

Outline

Definition of Strong Bisimulation I

Observation: equivalence should be deadlock sensitive
$\Longrightarrow$ needs to take branching structure of processes into account

This is guaranteed by a definition according to the following scheme:

Bisimulation scheme

$P, Q \in Prc$ are equivalent iff, for every $\alpha \in Act$, every α-successor of P is equivalent to some α-successor of Q, and vice versa.

In the first version we will ignore the special function of the silent action τ ($\Longrightarrow$ *weak bisimulation*)

Definition of Strong Bisimulation II

Definition 4.1 (Strong bisimulation)

A relation $\rho \subseteq Prc \times Prc$ is called a strong bisimulation if $P \rho Q$ implies, for every $\alpha \in Act$,

1. $P \xrightarrow{\alpha} P' \implies$ ex. $Q' \in Prc$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \rho Q'$
2. $Q \xrightarrow{\alpha} Q' \implies$ ex. $P' \in Prc$ such that $P \xrightarrow{\alpha} P'$ and $P' \rho Q'$

$P, Q \in Prc$ are called strongly bisimilar (notation: $P \sim Q$) if there exists a strong bisimulation ρ such that $P \rho Q$.

Theorem 4.2

$\sim$ *is an equivalence relation.*

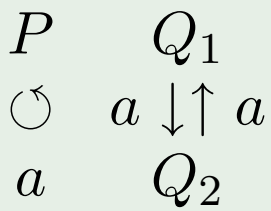
Proof.

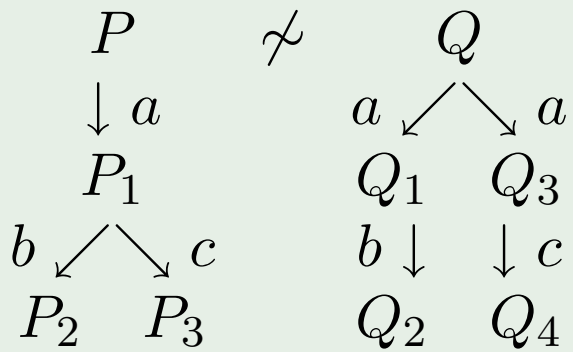
on the board $\square$

Examples I

Example 4.3

(on the board)

1. 

2. 

(remember: $Tr(P) = Tr(Q)$)

Examples II

Example 4.4

Binary semaphore

(controls exclusive access to two instances of a resource)

Sequential definition:

$$
\begin{aligned}
Sem_0(get, put) &= get.Sem_1(get, put) \\
Sem_1(get, put) &= get.Sem_2(get, put) + put.Sem_0(get, put) \\
Sem_2(get, put) &= put.Sem_1(get, put)
\end{aligned}
$$

Parallel definition:

$$
\begin{aligned}
S(get, put) &= S_0(get, put) \parallel S_0(get, put) \\
S_0(get, put) &= get.S_1(get, put) \\
S_1(get, put) &= put.S_0(get, put)
\end{aligned}
$$

Proposition: $Sem_0(get, put) \sim S(get, put)$ (see 3rd ex. sheet)

Examples III

Example 4.5

Two-place buffer

Sequential definition:

$$\begin{aligned} B_0(in, out) &= in.B_1(in, out) \\ B_1(in, out) &= \overline{out}.B_0(in, out) + in.B_2(in, out) \\ B_2(in, out) &= \overline{out}.B_1(in, out) \end{aligned}$$

Parallel definition:

$$\begin{aligned} B_{\|}(in, out) &= \mathsf{new}\ com\ (B(in, com) \parallel B(com, out)) \\ B(in, out) &= in.\overline{out}.B(in, out) \end{aligned}$$

Proposition: $B_0(in, out) \not\sim B_{\|}(in, out)$ (see 3rd ex. sheet)