Modeling Concurrent and Probabilistic Systems

Lecture 4: Trace Equivalence

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/mcps09/

Summer Semester 2009

Outline

1. Repetition: Equivalence of CCS Processes
2. Trace Equivalence
3. Deadlocks

Repetition: Equivalence Relations

Goal: identify process expressions which have the same "meaning" but differ in their syntax

Definition (Equivalence relation)

Let $\cong \subseteq S \times S$ be a binary relation over some set S. Then $\cong$ is called an equivalence relation if it is

- reflexive, i.e., $s \cong s$ for every $s \in S$,
- symmetric, i.e., $s \cong t$ implies $t \cong s$ for every $s, t \in S$, and
- transitive, i.e., $s \cong t$ and $t \cong u$ implies $s \cong u$ for every $s, t, u \in S$.

- **Generally:** two syntactic objects are equivalent if they have the same "meaning"
- **Here:** two processes are equivalent if they have the same "behavior" (i.e., communication potential)
- Communication potential described by LTS
- **Idea:** define (for processes P, Q)

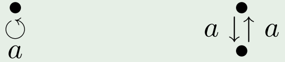
$$P \cong Q \text{ iff } LTS(P) = LTS(Q)$$

- **But:** yields too many distinctions:

Example

$$X(a) = a.X(a) \quad Y(a) = a.a.Y(a)$$

LTS:



although both processes can (only) execute infinitely many a-actions, and should be considered equivalent therefore

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).

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

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

where $Tr(P)$ is the set of all traces of P (see Def. 4.1)

Repetition: CCS Congruences

Goal: replacing a subcomponent of a system by an equivalent process should yield an equivalent systems
$\implies$ 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 $P, Q, R \in Prc$ such that $P \cong Q$ 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 $\alpha \in Act$ and $a \in N$.

Outline

Trace Equivalence I

Definition 4.1 (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 4.2 (One-place buffer)

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

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

Trace Equivalence II

Remarks:

- The trace language of $P \in Prc$ is accepted by the LTS of P, interpreted as an automaton with initial state P and where every state is final.
- Trace equivalence is obviously an equivalence relation (i.e., reflexive, symmetric, and transitive).
- Trace equivalence possesses the postulated properties of a process equivalence:
  1. it identifies processes with identical LTSs: the trace language of a process consists of the (finite) paths in the LTS. Hence processes with identical LTSs are trace equivalent.
  2. it implies trace equivalence: trivial
  3. it is a congruence:

Trace Equivalence III

Theorem 4.3

Trace equivalence is a congruence.

Proof.

(only for +; remaining operators analogously)
Clearly we have:

$$Tr(P_1 + P_2) = Tr(P_1) \cup Tr(P_2)$$

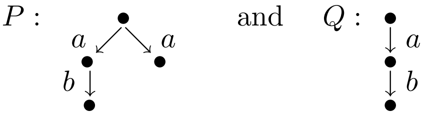
Now let $P, Q, R \in Prc$ with $Tr(P) = Tr(Q)$. Then:

$$\begin{array}{ll} Tr(P+R) & \\ = & Tr(P) \cup Tr(R) \\ = & Tr(Q) \cup Tr(R) \\ = & Tr(Q+R) \\ \Longrightarrow & P+R, Q+R \text{ trace equiv.} \end{array} \qquad \begin{array}{ll} Tr(R+P) & \\ = & Tr(R) \cup Tr(P) \\ = & Tr(R) \cup Tr(Q) \\ = & Tr(R+Q) \\ \Longrightarrow & R+P, R+Q \text{ trace equiv.} \end{array}$$

□

Trace Equivalence IV

- We have found a process equivalence with the three required properties.
- Are we satisfied? No!



are trace equivalent ($Tr(P) = Tr(Q) = \{\varepsilon, a, ab\}$)

- But P and Q are distinguishable:
  - both can execute ab
  - but P can deny b
  - while Q always has to offer b after a

(e.g., a = "insert coin", b = "return coffee")

$\Longrightarrow$ take into account such deadlock properties

Outline

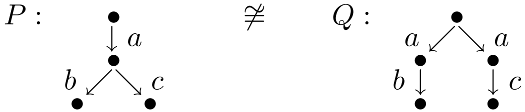
Deadlocks I

Definition 4.4 (Deadlock)

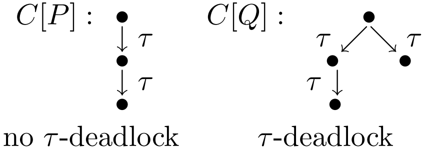
Let $P, Q \in Prc$ and $w \in Act^*$ such that $P \xrightarrow{w} Q$ and $Q \not\longrightarrow$.
Then Q is called a w-deadlock of P.

- 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.
- We therefore extend our set of postulates: our semantic equivalence $\cong$ should
  1. identify processes with identical LTSs;
  2. imply trace equivalence;
  3. be a congruence; and
  4. be 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).

The combination of congruence and deadlock sensitivity also excludes the following equivalence:



If $P \cong Q$, by congruence this equivalence should hold in every context. But $C[\cdot] := \mathsf{new}\, a, b, c\, (\overline{a}.\overline{b}.\mathsf{nil} \parallel \cdot)$ yields the following conflict:



Remarks:

- Another motivation: elevator control with a = "call elevator", b = "choose 1st floor", c = "choose 2nd floor",
- P and Q are obviously trace equivalent