

Semantics and Verification of Software

Lecture 19: Nondeterminism and Parallelism IV (Equivalence of CCS Processes)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)



noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/svsw13/>

Summer Semester 2013

Online Registration for Seminars and Practical Courses (Praktika) in Winter Term 2013/14

Who?

Students of:

- Master Courses
- Bachelor Informatik (~~Pro~~Seminar!)

Where?

www.graphics.rwth-aachen.de/apse

When?

05.07.2013 - 17.07.2013

- 1 Recapitulation: Calculus of Communicating Systems
- 2 Equivalence of CCS Processes
- 3 Strong Bisimulation

Definition 19.1 (Syntax of CCS)

- Let N be a set of (action) names.
- $\bar{N} := \{\bar{a} \mid a \in N\}$ denotes the set of co-names.
- $Act := N \cup \bar{N} \cup \{\tau\}$ is the set of actions where τ denotes the silent (or: unobservable) action.
- Let Pid be a set of process identifiers.
- The set Prc of process expressions is defined by the following syntax:

$P ::= \text{nil}$	(inaction)
$\alpha.P$	(prefixing)
$P_1 + P_2$	(choice)
$P_1 \parallel P_2$	(parallel composition)
$\text{new } a P$	(restriction)
$A(a_1, \dots, a_n)$	(process call)

where $\alpha \in Act$, $a, a_i \in N$, and $A \in Pid$.

Definition 19.1 (continued)

- A **(recursive) process definition** is an equation system of the form

$$(A_i(a_{i1}, \dots, a_{in_i}) = P_i \mid 1 \leq i \leq k)$$

where $k \geq 1$, $A_i \in \text{Pid}$ (pairwise different), $n_i \in \mathbb{N}$, $a_{ij} \in N$ ($a_{i1}, \dots, a_{in_i}$ pairwise different), and $P_i \in \text{Prc}$ (with process identifiers from $\{A_1, \dots, A_k\}$).

Notational Conventions:

- $\bar{a}$ means a
- $A(a_1, \dots, a_n)$ sometimes written as $A(\vec{a})$, $A()$ as A
- prefixing and restriction binds stronger than composition, composition binds stronger than choice:

$$\text{new } a P + b.Q \parallel R \quad \text{means} \quad (\text{new } a P) + ((b.Q) \parallel R)$$

Definition 19.2 (Semantics of CCS)

A process definition $(A_i(a_{i1}, \dots, a_{ini}) = P_i \mid 1 \leq i \leq k)$ determines the **labeled transition system (LTS)** $(Prc, Act, \longrightarrow)$ whose transitions can be inferred from the following rules ($P, P', Q, Q' \in Prc$, $\alpha \in Act$, $\lambda \in N \cup \bar{N}$, $a, b \in N$, $A \in Pid$):

$$(Act) \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$(Com) \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\bar{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(Sum_1) \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$(Sum_2) \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$(Par_1) \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$$

$$(Par_2) \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}$$

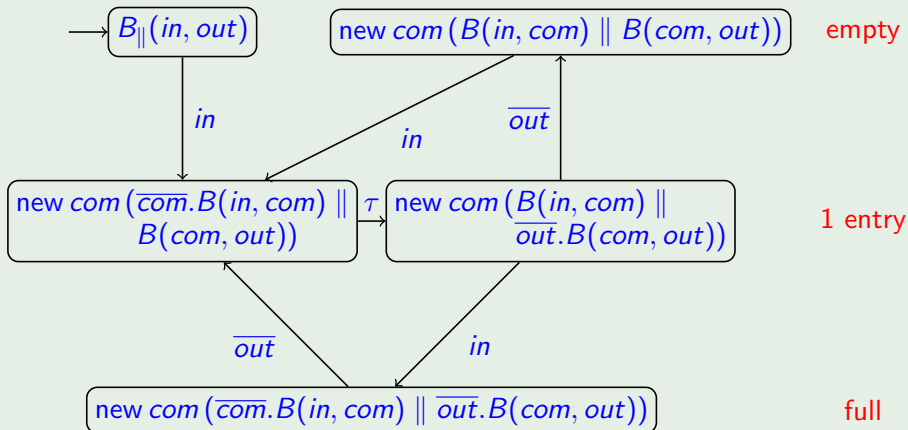
$$(New) \frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin \{a, \bar{a}\})}{\text{new } a \, P \xrightarrow{\alpha} \text{new } a \, P'}$$

$$(Call) \frac{P[\vec{a} \mapsto \vec{b}] \xrightarrow{\alpha} P'}{A(\vec{b}) \xrightarrow{\alpha} P'} \text{ if } A(\vec{a}) = P$$

(Here $P[\vec{a} \mapsto \vec{b}]$ denotes the replacement of every a_i by b_i in P .)

Example 19.3

Complete LTS of parallel two-place buffer ($=: LTS(B_{\parallel}(in, out))$):



- 1 Recapitulation: Calculus of Communicating Systems
- 2 Equivalence of CCS Processes
- 3 Strong Bisimulation

Equivalence of CCS Processes

- **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**
- **First idea:** define (for $P, Q \in \text{Prc}$)
 P, Q are called **LTS equivalent** if $LTS(P) = LTS(Q)$
- **But:** yields **too many distinctions**

Example 19.1

$$\begin{array}{ccc} X(a) = a.X(a) & & Y(a) = a.a.Y(a) \\ \text{LTS:} & \begin{array}{c} \bullet \\ \circ \\ a \end{array} & \neq \begin{array}{c} \bullet \\ \downarrow \uparrow \\ a \end{array} \end{array}$$

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

Second idea: reduce process to its action sequences

Definition 19.2 (Trace language)

For every $P \in \text{Prc}$, let

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

be the **trace language** of P

(where $\xrightarrow{w} := \xrightarrow{a_1} \circ \dots \circ \xrightarrow{a_n}$ for $w = a_1 \dots a_n$).

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

Example 19.3 (One-place buffer)

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

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

Remarks:

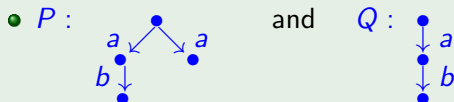
- The trace language of $P \in \text{Prc}$ is accepted by the LTS of P , interpreted as a (finite or infinite) automaton with **initial state** P and where **every state is final**.
- Trace equivalence is obviously an **equivalence relation** (i.e., reflexive, symmetric, and transitive).
- Trace equivalence identifies processes with **identical LTSs**: the trace language of a process consists of the (finite) paths in the LTS. Thus:

$$LTS(P) = LTS(Q) \Rightarrow Tr(P) = Tr(Q)$$

Trace Equivalence III

Are we satisfied with trace equivalence? No!

Example 19.4



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

• But P and Q are **distinguishable**:

- both can execute ab
- but P can deny b after a
- while Q always has to offer b after a

(e.g., consider a model of vending machine with a = “insert coin”, b = “return coffee”)

⇒ take into account such **deadlock properties**

- 1 Recapitulation: Calculus of Communicating Systems
- 2 Equivalence of CCS Processes
- 3 Strong Bisimulation

Definition of Strong Bisimulation I

Observation: equivalence should be sensitive to deadlocks
⇒ needs to take **branching structure** of processes into account

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

Bisimulation scheme

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

- **Strong** version ignores special function of silent action τ
(alternative: **weak bisimulation**; considered later)
- Unidirectional version: **simulation**
(not considered here)

Definition of Strong Bisimulation II

Definition 19.5 (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 $P \xrightarrow{\alpha} P' \Rightarrow \text{ex. } Q' \in Prc \text{ such that } Q \xrightarrow{\alpha} Q' \text{ and } P'\rho Q'$
- ② $Q \xrightarrow{\alpha} Q' \Rightarrow \text{ex. } P' \in Prc \text{ such that } P \xrightarrow{\alpha} P' \text{ and } P'\rho Q'$

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

Theorem 19.6

$\sim$ is an equivalence relation.

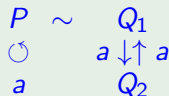
Proof.

omitted □

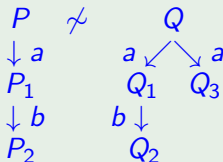
Example 19.7

(on the board)

- 1 Bisimilar but not LTS equivalent (cf. Example 19.1):



- 2 Trace equivalent (cf. Example 19.4) but not bisimilar:



Bisimulation vs. LTS/Trace Equivalence

Theorem 19.8

For every $P, Q \in \text{Prc}$,

$$LTS(P) = LTS(Q) \not\Rightarrow P \sim Q \not\Rightarrow Tr(P) = Tr(Q)$$

Proof.

- $LTS(P) = LTS(Q) \Rightarrow P \sim Q$: clear as Definition 19.5 (of $\sim$) is directly based on $LTS(p)$ and $LTS(Q)$
- $P \sim Q \not\Rightarrow LTS(p) = LTS(Q)$: see Example 19.7(1)
- $P \sim Q \Rightarrow Tr(P) = Tr(Q)$: by contradiction
(show: $\exists w \in Tr(P) \setminus Tr(Q) \Rightarrow P \not\sim Q$ by induction on $|w|$)
- $Tr(P) = Tr(Q) \not\Rightarrow P \sim Q$: see Example 19.7(2)



Example 19.9

Binary semaphore

(controls exclusive access to two instances of a resource)

Sequential definition:

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

Parallel definition:

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

Proposition: $Sem_0(\text{get}, \text{put}) \sim S(\text{get}, \text{put})$

Example 19.10

Two-place buffer

Sequential definition:

$$B_0(in, out) = in.B_1(in, out)$$

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

$$B_2(in, out) = \overline{out}.B_1(in, out)$$

Parallel definition:

$$B_{\parallel}(in, out) = \text{new } com (B(in, com) \parallel B(com, out))$$

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

Proposition: $B_0(in, out) \not\sim B_{\parallel}(in, out)$