

Semantics and Verification of Software

Lecture 21: Nondeterminism and Parallelism IV (Equivalence of CCS Processes & Wrap-Up)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)



noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/svsw11/>

Winter Semester 2011/12

- 1 Repetition: Calculus of Communicating Systems
- 2 Equivalence of CCS Processes
- 3 Strong Bisimulation
- 4 Further Topics in Formal Semantics
- 5 Upcoming Courses

Definition (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 (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

(on the board)

- ① One-place buffer:

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

- ② Sequential two-place buffer:

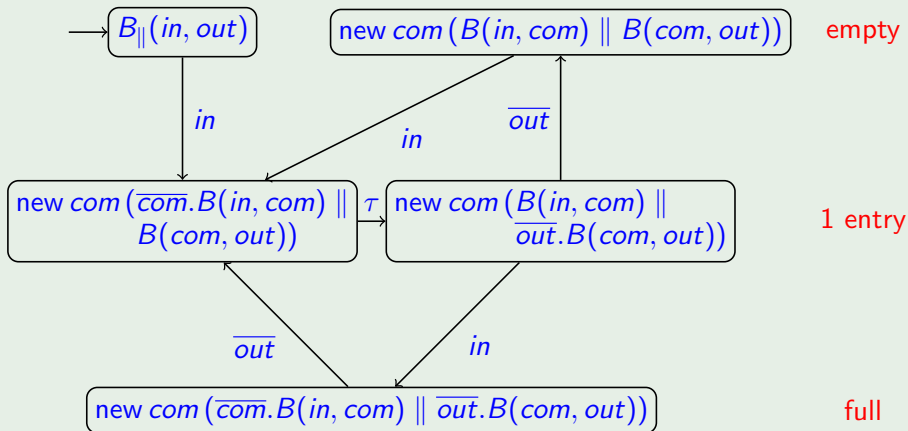
$$\begin{aligned}B_0(in, out) &= in.B_1(in, out) \\ B_1(in, out) &= \overline{out}.B_0(in, out) + in.B_2(in, out) \\ B_2(in, out) &= \overline{out}.B_1(in, out)\end{aligned}$$

- ③ Parallel two-place buffer:

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

Example (continued)

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



- 1 Repetition: Calculus of Communicating Systems
- 2 Equivalence of CCS Processes
- 3 Strong Bisimulation
- 4 Further Topics in Formal Semantics
- 5 Upcoming Courses

Equivalence of CCS Processes

- **Generally:** two syntactic objects are equivalent if they have the same “meaning”

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

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 $\text{LTS}(P) = \text{LTS}(Q)$
- **But:** yields too many distinctions

Example 21.1

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

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 21.1

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

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 21.2 (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 $\text{Tr}(P) = \text{Tr}(Q)$.

Second idea: reduce process to its action sequences

Definition 21.2 (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 21.3 (One-place buffer)

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

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

Remarks:

- The trace language of $P \in \text{Prc}$ is accepted by the LTS of P , interpreted as an automaton with initial state P and where every state is final.

Remarks:

- The trace language of $P \in \text{Prc}$ 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).

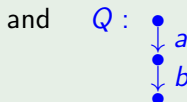
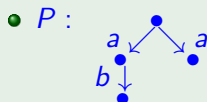
Remarks:

- The trace language of $P \in \text{Prc}$ 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 identifies processes with **identical LTSs**: the trace language of a process consists of the (finite) paths in the LTS. Thus:

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

Are we satisfied with trace equivalence? No!

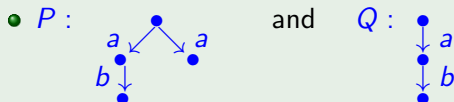
Example 21.4



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

Are we satisfied with trace equivalence? No!

Example 21.4



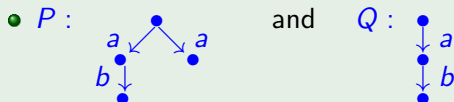
are **trace equivalent** ($Tr(P) = Tr(Q) = \{\varepsilon, a, ab\}$)

- But P and Q are **distinguishable**:
 - both can execute ab
 - but P can deny b 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”)

Are we satisfied with trace equivalence? No!

Example 21.4



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 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 Repetition: Calculus of Communicating Systems
- 2 Equivalence of CCS Processes
- 3 Strong Bisimulation**
- 4 Further Topics in Formal Semantics
- 5 Upcoming Courses

Definition of Strong Bisimulation I

Observation: equivalence should be sensitive to deadlocks

⇒ needs to take **branching structure** of processes into account

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

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

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**; not considered here)
- Unidirectional version: **simulation**
(not considered here)

Definition of Strong Bisimulation II

Definition 21.5 (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 $P\rho Q$.

Definition of Strong Bisimulation II

Definition 21.5 (Strong bisimulation)

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

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

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

Theorem 21.6

$\sim$ is an equivalence relation.

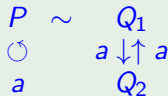
Proof.

omitted □

Example 21.7

(on the board)

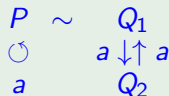
- 1 Bisimilar but not LTS equivalent (cf. Example 21.1):



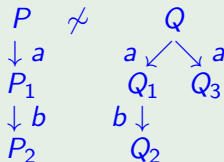
Example 21.7

(on the board)

- 1 Bisimilar but not LTS equivalent (cf. Example 21.1):



- 2 Trace equivalent (cf. Example 21.4) but not bisimilar:



Bisimulation vs. LTS/Trace Equivalence

Theorem 21.8

For every $P, Q \in Prc$,

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

Bisimulation vs. LTS/Trace Equivalence

Theorem 21.8

For every $P, Q \in Prc$,

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

Proof.

- $LTS(P) = LTS(Q) \Rightarrow P \sim Q$: clear as Definition 21.5 (of $\sim$) is directly based on $LTS(p)$ and $LTS(Q)$

Bisimulation vs. LTS/Trace Equivalence

Theorem 21.8

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 21.5 (of $\sim$) is directly based on $LTS(p)$ and $LTS(Q)$
- $P \sim Q \not\Rightarrow LTS(p) = LTS(Q)$: see Example 21.7(1)

Bisimulation vs. LTS/Trace Equivalence

Theorem 21.8

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 21.5 (of $\sim$) is directly based on $LTS(p)$ and $LTS(Q)$
- $P \sim Q \not\Rightarrow LTS(p) = LTS(Q)$: see Example 21.7(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 $|w|$)

Bisimulation vs. LTS/Trace Equivalence

Theorem 21.8

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 21.5 (of $\sim$) is directly based on $LTS(p)$ and $LTS(Q)$
- $P \sim Q \not\Rightarrow LTS(p) = LTS(Q)$: see Example 21.7(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 21.7(2)



Example 21.9

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 21.10

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

- 1 Repetition: Calculus of Communicating Systems
- 2 Equivalence of CCS Processes
- 3 Strong Bisimulation
- 4 Further Topics in Formal Semantics**
- 5 Upcoming Courses

- Program = list of **function definitions**

Semantics of Functional Languages I

- Program = list of **function definitions**
- Simplest setting: **first-order** function definitions of the form

$$f(x_1, \dots, x_n) = t$$

- function name f
- formal parameters $x_1, \dots, x_n$
- term t over (base and defined) function calls and $x_1, \dots, x_n$

Semantics of Functional Languages I

- Program = list of **function definitions**
- Simplest setting: **first-order** function definitions of the form

$$f(x_1, \dots, x_n) = t$$

- function name f
 - formal parameters $x_1, \dots, x_n$
 - term t over (base and defined) function calls and $x_1, \dots, x_n$
- **Operational semantics** (only function calls)
 - **call-by-value** case:

$$\frac{t_1 \rightarrow z_1 \quad \dots \quad t_n \rightarrow z_n \quad t[x_1 \mapsto z_1, \dots, x_n \mapsto z_n] \rightarrow z}{f(t_1, \dots, t_n) \rightarrow z}$$

- **call-by-name** case:

$$\frac{t[x_1 \mapsto t_1, \dots, x_n \mapsto t_n] \rightarrow z}{f(t_1, \dots, t_n) \rightarrow z}$$

- Denotational semantics

- program = **equation system** (for functions)
- induces call-by-value and call-by-name **functional**
- **monotonic and continuous** w.r.t. graph inclusion
- semantics := **least fixpoint** (Tarski/Knaster Theorem)
- **coincides** with operational semantics

- Denotational semantics
 - program = **equation system** (for functions)
 - induces call-by-value and call-by-name **functional**
 - **monotonic and continuous** w.r.t. graph inclusion
 - semantics := **least fixpoint** (Tarski/Knaster Theorem)
 - **coincides** with operational semantics
- **Extensions:** higher-order types, data types, ...

- Denotational semantics
 - program = **equation system** (for functions)
 - induces call-by-value and call-by-name **functional**
 - **monotonic and continuous** w.r.t. graph inclusion
 - semantics := **least fixpoint** (Tarski/Knaster Theorem)
 - **coincides** with operational semantics
- **Extensions:** higher-order types, data types, ...
- see [Winskel 1996, Sct. 9] and **Functional Programming** course [Giesl]

- 1 Repetition: Calculus of Communicating Systems
- 2 Equivalence of CCS Processes
- 3 Strong Bisimulation
- 4 Further Topics in Formal Semantics
- 5 Upcoming Courses

- Course **Advanced Model Checking** [Katoen]
- Course **Compiler Construction** [Noll] (“Hiwi” jobs available!)