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)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

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

Winter Semester 2011/12

Outline

Syntax of CCS I

Definition (Syntax of CCS)

- Let N be a set of (action) names.
- $\overline{N} := \{\overline{a} \mid a \in N\}$ denotes the set of co-names.
- $Act := N \cup \overline{N} \cup \{\tau\}$ 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:

$$
\begin{array}{rlll}
P & ::= & \mathrm{nil} & \text{(inaction)} \\
 & \mid & \alpha.P & \text{(prefixing)} \\
 & \mid & P_1 + P_2 & \text{(choice)} \\
 & \mid & P_1 \parallel P_2 & \text{(parallel composition)} \\
 & \mid & \mathrm{new}\ a\ P & \text{(restriction)} \\
 & \mid & A(a_1, \ldots, a_n) & \text{(process call)}
\end{array}
$$

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

Definition (Semantics of CCS)

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

$$\text{(Act)}\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \text{(Com)}\frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$\text{(Sum}_1\text{)}\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \text{(Sum}_2\text{)}\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\text{(Par}_1\text{)}\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad \text{(Par}_2\text{)}\frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}$$

$$\text{(New)}\frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin \{a, \overline{a}\})}{\text{new } a\, P \xrightarrow{\alpha} \text{new } a\, P'} \qquad \text{(Call)}\frac{P[\vec{a} \mapsto \vec{b}] \xrightarrow{\alpha} P'}{A(\vec{b}) \xrightarrow{\alpha} P'} \text{ if } A(\vec{a}) = P$$

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

Semantics of CCS II

Example

(on the board)

1. One-place buffer:

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

2. Sequential two-place buffer:

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

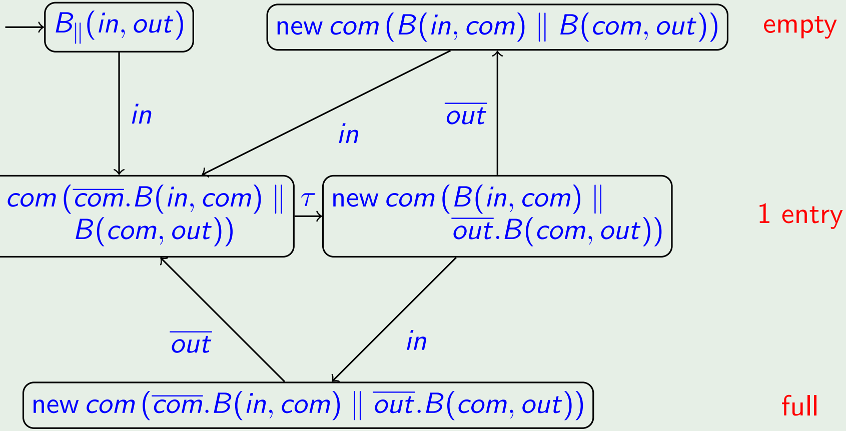
3. Parallel two-place buffer:

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

Semantics of CCS III

Example (continued)

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



Outline

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 $P, Q \in Prc$)

  $$P, Q \text{ are called LTS equivalent if } LTS(P) = LTS(Q)$$

- **But:** yields too many distinctions

Example 21.1

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

LTS: 

$\neq$ 

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

Trace Equivalence I

Second idea: reduce process to its action sequences

Definition 21.2 (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
(where $\xrightarrow{w} := \xrightarrow{a_1} \circ \ldots \circ \xrightarrow{a_1}$ for $w = a_1 \ldots a_n$).

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

Example 21.3 (One-place buffer)

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

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

Trace Equivalence II

Remarks:

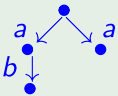
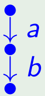
- The trace language of $P \in Prc$ is accepted by the LTS of P, interpreted as an automaton with initial state P and where every state is final.
- Trace equivalence is obviously an equivalence relation (i.e., reflexive, symmetric, and transitive).
- Trace equivalence identifies processes with identical 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 21.4

- P :  and Q : 

  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")

$\Rightarrow$ take into account such deadlock properties

Outline

Definition of Strong Bisimulation I

Observation: equivalence should be sensitive to deadlocks
$\Rightarrow$ 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.

- 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 $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is called a strong bisimulation if $P \rho Q$ implies, for every $\alpha \in \mathit{Act}$,

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

$P, Q \in \mathit{Prc}$ 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 $\square$

Example 21.7

(on the board)

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

$$\begin{array}{ccc} P & \sim & Q_1 \\ \circlearrowleft & & a \downarrow\uparrow a \\ a & & Q_2 \end{array}$$

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

$$\begin{array}{ccccc} P & \not\sim & & Q & \\ \downarrow a & & a \swarrow & & \searrow a \\ P_1 & & Q_1 & & Q_3 \\ \downarrow b & & b \downarrow & & \\ P_2 & & Q_2 & & \end{array}$$

Bisimulation vs. LTS/Trace Equivalence

Theorem 21.8

For every $P, Q \in Prc$,

$$LTS(P) = LTS(Q) \quad \begin{array}{c}\Rightarrow\\ \not\Leftarrow\end{array} \quad P \sim Q \quad \begin{array}{c}\Rightarrow\\ \not\Leftarrow\end{array} \quad Tr(P) = Tr(Q)$$

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)

□

More Examples I

Example 21.9

Binary semaphore
(controls exclusive access to two instances of a resource)
Sequential definition:

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

Parallel definition:

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

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

More Examples II

Example 21.10

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:

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

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

Outline

Semantics of Functional Languages I

- Program = list of function definitions
- Simplest setting: first-order function definitions of the form
$$f(x_1, \ldots, x_n) = t$$
  - function name f
  - formal parameters $x_1, \ldots, x_n$
  - term t over (base and defined) function calls and $x_1, \ldots, x_n$
- Operational semantics (only function calls)
  - call-by-value case:
$$\frac{t_1 \to z_1 \quad \ldots \quad t_n \to z_n \quad t[x_1 \mapsto z_1, \ldots, x_n \mapsto z_n] \to z}{f(t_1, \ldots, t_n) \to z}$$
  - call-by-name case:
$$\frac{t[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n] \to z}{f(t_1, \ldots, t_n) \to z}$$

Semantics of Functional Languages II

- 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]

Outline

Courses Summer 2012

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