Modeling Concurrent and Probabilistic Systems

Lecture 2: Semantics of CCS

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: 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:

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

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

Repetition: Syntax of CCS II

Definition (continued)

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

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

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

Outline

Labeled Transition Systems

Goal: represent behavior of system by (infinite) graph

- nodes = system states
- edges = transitions between states

Definition 2.1 (Labeled transition system)

A (Act-)labeled transition system (LTS) is a triple $(S, Act, \longrightarrow)$ consisting of

- a set S of states
- a set Act of (action) labels
- a transition relation $\longrightarrow \subseteq S \times Act \times S$

If $(s, \alpha, s') \in \longrightarrow$ we write $s \xrightarrow{\alpha} s'$. An LTS is called finite if S is so.

Remarks:

- sometimes an initial state $s_0 \in S$ is distinguished
- (finite) LTSs correspond to (finite) automata without final states

Semantics of CCS I

We define the assignment

$$\begin{array}{rcl} \text{syntax} & \longrightarrow & \text{semantics} \\ \text{process definition} & \longmapsto & \text{LTS} \end{array}$$

by induction over the syntactic structure of process expressions. Here we employ derivation rules of the form

$$\frac{\text{premise(s)}}{\text{conclusion}}\ \text{rule name}$$

which can be composed to complete derivation trees.

Definition 2.2 (Semantics of CCS)

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

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

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

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

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

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

Semantics of CCS III

Example 2.3

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:

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

(on the board)

Example 2.3 (continued)

Complete LTS of parallel two-place buffer:

$$
\begin{array}{ccc}
B_{\|}(in, out) & & \mathsf{new}\ com\,(B(in, com) \parallel B(com, out)) \\
\downarrow in \quad \swarrow in & & \uparrow \overline{out} \\
\mathsf{new}\ com\,(\overline{com}.B(in, com) \parallel B(com, out)) & \xrightarrow{\tau} & \mathsf{new}\ com\,(B(in, com) \parallel \overline{out}.B(com, out)) \\
\nwarrow \overline{out} & & \swarrow in \\
\multicolumn{3}{c}{\mathsf{new}\ com\,(\overline{com}.B(in, com) \parallel \overline{out}.B(com, out))}
\end{array}
$$

Recursive Processes

Here: recursive processes defined using equations such as

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

(simultaneous recursion)

Alternative: explicit fixpoint operator

- syntax: $P ::= \mathsf{nil} \mid \ldots \mid \mathsf{fix}\, A\, P \in Prc \quad$ (where $A \in Pid$)
- semantics: $\dfrac{P[A \mapsto P] \xrightarrow{\alpha} P'}{\mathsf{fix}\, A\, P \xrightarrow{\alpha} \mathsf{fix}\, A\, P'}\,(\mathsf{Fix})$
- example: $\dfrac{\dfrac{}{in.\overline{out}.in.\overline{out}.B \xrightarrow{in} \overline{out}.in.\overline{out}.B}\,(\mathsf{Act})}{\mathsf{fix}\, B\, in.\overline{out}.B \xrightarrow{in} \mathsf{fix}\, B\, \overline{out}.in.\overline{out}.B}\,(\mathsf{Fix})$

(nested scalar recursion)

Advantage: only process term level required (no equations)
$\Longrightarrow$ simplification of theory

Disadvantage: bad readability of process definitions