

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/mcps09/>

Summer Semester 2009

1 Repetition: Syntax of CCS

2 Semantics of CCS

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: $P ::=$

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

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

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

Meaning of CCS Constructs

- nil is an **inactive process** that can do nothing.
- $\alpha.P$ can execute α and then behaves as P .
- An action $a \in N$ ($\bar{a} \in \bar{N}$) is interpreted as an **input** (**output**, resp.) operation. Both are complementary: if executed in parallel (i.e., in $P_1 \parallel P_2$), they are merged into a τ -action.
- $P_1 + P_2$ represents the **non-deterministic choice** between P_1 and P_2 .
- $P_1 \parallel P_2$ denotes the **concurrent execution** of P_1 and P_2 , involving **interleaving** or **communication**.
- The **restriction** $\text{new } a P$ declares a as a local name which is only known in P .
- The behavior of a **process call** $A(a_1, \dots, a_n)$ is defined by the right-hand side of the corresponding equation where $a_1, \dots, a_n$ replace the formal name parameters.

Example

- ➊ One-place buffer
- ➋ Two-place buffer
- ➌ Parallel specification of two-place buffer

(on the board)

- $\overline{\overline{a}}$ means a
- $P_1 + \dots + P_n$ ($n \in \mathbb{N}$) sometimes written as $\sum_{i=1}^n P_i$ where $\sum_{i=1}^0 P_i := \text{nil}$
- “.nil” can be omitted: $a.b$ means $a.b.\text{nil}$
- $\text{new } a, b P$ means $\text{new } a \text{ new } b P$
- $A(a_1, \dots, a_n)$ sometimes written as $A(\vec{a})$, $A()$ as A
- prefixing and restriction binds stronger than composition, composition binds stronger than choice:

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

1 Repetition: Syntax of CCS

2 Semantics of CCS

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** $\longrightarrow \subseteq S \times Act \times S$

For $(s, \alpha, s') \in \longrightarrow$ 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

We define the assignment

$$\begin{array}{ll} \text{syntax} & \rightarrow \text{ semantics} \\ \text{process definition} & \mapsto \text{ LTS} \end{array}$$

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

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

which can be composed to complete **derivation trees**.

Definition 2.2 (Semantics of CCS)

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

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

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

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

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

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

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

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

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

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

Example 2.3

- ❶ One-place buffer:

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

- ❷ Sequential two-place buffer:

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

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

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

- ❸ Parallel two-place buffer:

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

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

(on the board)

Example 2.3 (continued)

Complete LTS of parallel two-place buffer:

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