

Semantics and Verification of Software

Lecture 18: Nondeterminism and Parallelism III (Calculus of Communicating Systems)

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

1 Calculus of Communicating Systems

2 Semantics of CCS

History:

- Robin Milner: *A Calculus of Communicating Systems*
LNCS 92, Springer, 1980
- Robin Milner: *Communication and Concurrency*
Prentice-Hall, 1989
- Robin Milner: *Communicating and Mobile Systems: the π -calculus*
Cambridge University Press, 1999

History:

- Robin Milner: *A Calculus of Communicating Systems*
LNCS 92, Springer, 1980
- Robin Milner: *Communication and Concurrency*
Prentice-Hall, 1989
- Robin Milner: *Communicating and Mobile Systems: the π -calculus*
Cambridge University Press, 1999

Approach: describing parallelism on a **simple and abstract level**, using only a few basic primitives

- no explicit storage (variables)
 - no explicit representation of values (numbers, Booleans, ...)
- ⇒ parallel system reduced to **communication potential**

Definition 18.1 (Syntax of CCS)

- Let N be a set of (action) names.

Definition 18.1 (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.

Definition 18.1 (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.

Definition 18.1 (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.

Definition 18.1 (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:

$P ::= \text{nil}$	(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 18.1 (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 $k \geq 1$, $A_i \in Pid$ (pairwise different), $n_i \in \mathbb{N}$, $a_{ij} \in N$ (a_{i1}, \dots, a_{in_i} pairwise different), and $P_i \in Prc$ (with process identifiers from $\{A_1, \dots, A_k\}$).

Definition 18.1 (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:

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

- `nil` is an `inactive process` that can do nothing.

Meaning of CCS Constructs

- nil is an **inactive process** that can do nothing.
- $\alpha.P$ can execute α and then behaves as P .

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 $a \in N$ ($\bar{a} \in \overline{N}$) 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.

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 $a \in N$ ($\bar{a} \in \overline{N}$) 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 **nondeterministic choice** between P_1 and P_2 .

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 $a \in N$ ($\bar{a} \in \overline{N}$) 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 **nondeterministic choice** between P_1 and P_2 .
- $P_1 \parallel P_2$ denotes the **parallel execution** of P_1 and P_2 , involving **interleaving** or **communication**.

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 $a \in N$ ($\bar{a} \in \overline{N}$) 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 **nondeterministic choice** between P_1 and P_2 .
- $P_1 \parallel P_2$ denotes the **parallel 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 within P .

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 **nondeterministic choice** between P_1 and P_2 .
- $P_1 \parallel P_2$ denotes the **parallel 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 within 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 18.2

(on the board)

- ➊ One-place buffer (see Example 17.1(1) for a CSP implementation)
- ➋ Two-place buffer
- ➌ Parallel specification of two-place buffer
(see Example 17.1(2) for a CSP implementation)

1 Calculus of Communicating Systems

2 Semantics of CCS

Definition 18.3 (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 ($P, P', Q, Q' \in Prc$, $\alpha \in Act$, $\lambda \in N \cup \overline{N}$, $a, b \in N$, $A \in Pid$):

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

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

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

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

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

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

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

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

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

Example 18.4

(on the board)

- ① One-place buffer:

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

Example 18.4

(on the board)

- ① One-place buffer:

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

- ② Sequential two-place buffer:

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

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

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

Example 18.4

(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 18.4 (continued)

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

