Concurrency Theory

Winter Semester 2019/20

Lecture 2: Calculus of Communicating Systems (CCS)

Joost-Pieter Katoen and Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-19-20/ct/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Outline of Lecture 2

The Approach

Syntax of CCS

Intuitive Meaning and Examples

Formal Semantics of CCS

The Calculus of Communicating Systems

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

The Approach

The Calculus of Communicating Systems

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, ...)

$\Rightarrow$ concurrent system reduced to communication potential

Outline of Lecture 2

The Approach

Syntax of CCS

Intuitive Meaning and Examples

Formal Semantics of CCS

Syntax of CCS I

Definition 2.1 (Syntax of CCS)

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

Syntax of CCS I

Definition 2.1 (Syntax of CCS)

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

Syntax of CCS I

Definition 2.1 (Syntax of CCS)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.

Syntax of CCS I

Definition 2.1 (Syntax of CCS)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.
- Let Pid be a set of process identifiers.

Syntax of CCS I

Definition 2.1 (Syntax of CCS)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.
- Let Pid be a set of process identifiers.
- The set Prc of process expressions is defined by the following syntax:

$$
\begin{array}{rlll}
P ::= & \mathrm{nil} & & \text{(inaction)} \\
\mid & \alpha.P & & \text{(prefixing)} \\
\mid & P_1 + P_2 & & \text{(choice)} \\
\mid & P_1 \parallel P_2 & & \text{(parallel composition)} \\
\mid & P \setminus L & & \text{(restriction)} \\
\mid & P[f] & & \text{(relabelling)} \\
\mid & C & & \text{(process call)}
\end{array}
$$

where $\alpha \in Act$, $\emptyset \neq L \subseteq A$, $C \in Pid$, and $f : Act \to Act$ such that $f(\tau) = \tau$ and $f(\overline{a}) = \overline{f(a)}$ for each $a \in A$.

Syntax of CCS II

Definition 2.1 (continued)

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

$$(C_i = P_i \mid 1 \leq i \leq k)$$

where $k \geq 1$, $C_i \in Pid$ (pairwise distinct), and $P_i \in Prc$ (with identifiers from $\{C_1, \ldots, C_k\}$).

Syntax of CCS II

Definition 2.1 (continued)

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

$$(C_i = P_i \mid 1 \leq i \leq k)$$

where $k \geq 1$, $C_i \in Pid$ (pairwise distinct), and $P_i \in Prc$ (with identifiers from $\{C_1, \ldots, C_k\}$).

Notational Conventions:

- $\overline{\overline{a}}$ means a
- $\sum_{i=1}^{n} P_i$ ($n \in \mathbb{N}$) means $P_1 + \ldots + P_n$ (where $\sum_{i=1}^{0} P_i :=$ nil)
- $P \setminus a$ abbreviates $P \setminus \{a\}$
- $[a_1 \mapsto b_1, \ldots, a_n \mapsto b_n]$ stands for $f : Act \to Act$ with $f(a_i) = b_i$ for $i \in [n]$ and $f(\alpha) = \alpha$ otherwise
- restriction and relabelling bind stronger than prefixing, prefixing stronger than composition, composition stronger than choice:

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

Outline of Lecture 2

The Approach

Syntax of CCS

Intuitive Meaning and Examples

Formal Semantics of CCS

Intuitive Meaning and Examples

Meaning of CCS Constructs

- nil is an inactive process that can do nothing.

Intuitive Meaning and Examples

Meaning of CCS Constructs

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

Intuitive Meaning and Examples

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 $a \in A$ ($\overline{a} \in \overline{A}$) 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.

Intuitive Meaning and Examples

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 $a \in A$ ($\overline{a} \in \overline{A}$) 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 $a \in A$ ($\overline{a} \in \overline{A}$) 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 $a \in A$ ($\overline{a} \in \overline{A}$) 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 $P \setminus L$ declares each $a \in L$ 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 $a \in A$ ($\overline{a} \in \overline{A}$) 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 $P \setminus L$ declares each $a \in L$ as a local name which is only known within P.
- The relabelling $P[f]$ allows to adapt the naming of actions.

Intuitive Meaning and Examples

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 $a \in A$ ($\overline{a} \in \overline{A}$) 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 $P \setminus L$ declares each $a \in L$ as a local name which is only known within P.
- The relabelling $P[f]$ allows to adapt the naming of actions.
- The behaviour of a process call C is given by the right-hand side of the corresponding equation.

Intuitive Meaning and Examples

CCS Examples

Example 2.2

1. One-place buffer
2. Two-place buffer
3. Parallel specification of two-place buffer

(on the board)

Outline of Lecture 2

Labelled Transition Systems

Goal: represent behaviour of system by (infinite) graph

- nodes = system states
- edges = transitions between states

Labelled Transition Systems

Goal: represent behaviour of system by (infinite) graph

- nodes = system states
- edges = transitions between states

Definition 2.3 (Labelled transition system)

An (*Act*-)labelled 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.

Formal Semantics of CCS

Labelled Transition Systems

Goal: represent behaviour of system by (infinite) graph

- nodes = system states
- edges = transitions between states

Definition 2.3 (Labelled transition system)

An (Act-)labelled 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 ("$LTS(s_0)$")
- (finite) LTSs correspond to (finite) automata without final states

Semantics of CCS I

We define the assignment

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

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 are composed to form derivation trees (where axioms, i.e., rules without premises, correspond to leaves).

Semantics of CCS II

Definition 2.4 (Semantics of CCS)

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

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

$$\text{(Par}_1\text{)}\ \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad \text{(Par}_2\text{)}\ \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \qquad \text{(Com)}\ \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$\text{(Res)}\ \frac{P \xrightarrow{\alpha} P' \quad (\alpha, \overline{\alpha} \notin L)}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \qquad \text{(Rel)}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad \text{(Call)}\ \frac{P \xrightarrow{\alpha} P' \quad (C = P)}{C \xrightarrow{\alpha} P'}$$

Semantics of CCS III

Example 2.5

1. One-place buffer:

$$B = in.\overline{out}.B$$

2. Sequential two-place buffer:

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

3. Parallel two-place buffer:

$$\begin{aligned} B_\parallel &= (B[f] \parallel B[g]) \setminus com \\ B &= in.\overline{out}.B \end{aligned}$$

   where $f := [out \mapsto com]$ and $g := [in \mapsto com]$

(on the board)

Semantics of CCS IV

Example 2.5 (continued)

Complete LTS of parallel two-place buffer:

$$
\begin{tikzcd}
{} \arrow[r] & B_{\|} \arrow[d, "in"] & & (B[f] \parallel B[g]) \setminus com \arrow[dll, "in"'] & \text{empty} \\
& ((\overline{out}.B)[f] \parallel B[g]) \setminus com \arrow[rr, "\tau"] & & (B[f] \parallel (\overline{out}.B)[g]) \setminus com \arrow[u, "\overline{out}"'] \arrow[dl, "in"] & \text{one entry} \\
& & ((\overline{out}.B)[f] \parallel (\overline{out}.B)[g]) \setminus com \arrow[ul, "\overline{out}"] & & \text{full}
\end{tikzcd}
$$