

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/





Outline of Lecture 2

The Approach

Syntax of CCS

Intuitive Meaning and Examples

Formal Semantics of CCS

2 of 15 Concurrency Theory Winter Semester 2019/20 Lecture 2: Calculus of Communicating Systems (CCS)





The Approach

The Calculus of Communicating Systems

History:

3 of 15

- 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

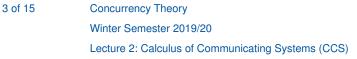
The Calculus of Communicating Systems

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

4 of 15 Concurrency Theory Winter Semester 2019/20 Lecture 2: Calculus of Communicating Systems (CCS)



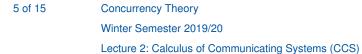


Syntax of CCS

Syntax of CCS I

Definition 2.1 (Syntax of CCS)

• Let *A* be a set of (action) names.







Syntax of CCS

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.

Ρ

- 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:

$$::=$$
 nil(inaction) $| \alpha.P$ (prefixing) $| P_1 + P_2$ (choice) $| P_1 || P_2$ (parallel composition) $| P \setminus L$ (restriction) $| P[f]$ (relabelling) $| C$ (process call)

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

5 of 15

Concurrency Theory Winter Semester 2019/20 Lecture 2: Calculus of Communicating Systems (CCS)





Syntax of CCS II

6 of 15

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

Notational Conventions:

- 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:

 $P \setminus a + b.Q \parallel R$ means $(P \setminus a) + ((b.Q) \parallel R)$





Outline of Lecture 2

The Approach

Syntax of CCS

Intuitive Meaning and Examples

Formal Semantics of CCS

7 of 15 Concurrency Theory Winter Semester 2019/20 Lecture 2: Calculus of Communicating Systems (CCS)





• nil is an inactive process that can do nothing.





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





- nil is an inactive process that can do nothing.
- α .*P* can execute α and then behaves as *P*.
- An action a ∈ A (ā ∈ Ā) is interpreted as an input (output, resp.) operation. Both are complementary: if executed in parallel (i.e., in P₁ || P₂), they are merged into a *τ*-action.





- nil is an inactive process that can do nothing.
- α .*P* can execute α and then behaves as *P*.
- An action a ∈ A (ā ∈ Ā) is interpreted as an input (output, resp.) operation. Both are complementary: if executed in parallel (i.e., in P₁ || P₂), they are merged into a *τ*-action.
- $P_1 + P_2$ represents the nondeterministic choice between P_1 and P_2 .





- nil is an inactive process that can do nothing.
- α .*P* can execute α and then behaves as *P*.
- An action a ∈ A (ā ∈ Ā) is interpreted as an input (output, resp.) operation. Both are complementary: if executed in parallel (i.e., in P₁ || P₂), 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.





8 of 15

- nil is an inactive process that can do nothing.
- α .*P* can execute α and then behaves as *P*.
- An action a ∈ A (ā ∈ Ā) is interpreted as an input (output, resp.) operation. Both are complementary: if executed in parallel (i.e., in P₁ || P₂), they are merged into a *τ*-action.
- $P_1 + P_2$ represents the nondeterministic choice between P_1 and P_2 .
- P₁ || P₂ denotes the parallel execution of P₁ and P₂, involving interleaving or communication.
- The restriction $P \setminus L$ declares each $a \in L$ as a local name which is only known within P.





- nil is an inactive process that can do nothing.
- α .*P* can execute α and then behaves as *P*.
- An action a ∈ A (ā ∈ Ā) is interpreted as an input (output, resp.) operation. Both are complementary: if executed in parallel (i.e., in P₁ || P₂), 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.





- nil is an inactive process that can do nothing.
- α .*P* can execute α and then behaves as *P*.
- An action a ∈ A (ā ∈ Ā) is interpreted as an input (output, resp.) operation. Both are complementary: if executed in parallel (i.e., in P₁ || P₂), 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.





CCS Examples

Example 2.2

- 1. One-place buffer
- 2. Two-place buffer
- 3. Parallel specification of two-place buffer
- (on the board)

9 of 15







Outline of Lecture 2

The Approach

Syntax of CCS

Intuitive Meaning and Examples

Formal Semantics of CCS

10 of 15 Concurrency Theory Winter Semester 2019/20 Lecture 2: Calculus of Communicating Systems (CCS)





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, \rightarrow) 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.





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, \rightarrow) 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

syntax \rightarrow semantics

process definition \mapsto LTS

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

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

$$(Act) \xrightarrow{(Act)} \overline{\alpha.P \xrightarrow{\alpha} P} \qquad (Sum_1) \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad (Sum_2) \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$(Par_1) \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad (Par_2) \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \qquad (Con) \frac{P \xrightarrow{\lambda} P' Q \xrightarrow{\lambda} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(Con) \frac{P \xrightarrow{\lambda} P' Q \xrightarrow{\lambda} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(Con) \frac{P \xrightarrow{\lambda} P' Q \xrightarrow{\lambda} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(Con) \frac{P \xrightarrow{\alpha} P' Q \xrightarrow{\lambda} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(Con) \frac{P \xrightarrow{\alpha} P' Q \xrightarrow{\lambda} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(Con) \frac{P \xrightarrow{\alpha} P' Q \xrightarrow{\lambda} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(Con) \frac{P \xrightarrow{\alpha} P' Q \xrightarrow{\lambda} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(Con) \frac{P \xrightarrow{\alpha} P' Q \xrightarrow{\lambda} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(Con) \frac{P \xrightarrow{\alpha} P' Q \xrightarrow{\lambda} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

13 of 15 Concurrency Theory Winter Semester 2019/20 Lecture 2: Calculus of Communicating Systems (CCS)





Semantics of CCS III

Example 2.5

1. One-place buffer:

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

2. Sequential two-place buffer:

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

3. Parallel two-place buffer:

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

where $f := [out \mapsto com]$ and $g := [in \mapsto com]$ (on the board)

 14 of 15
 Concurrency Theory

 Winter Semester 2019/20

 Lecture 2: Calculus of Communicating Systems (CCS)



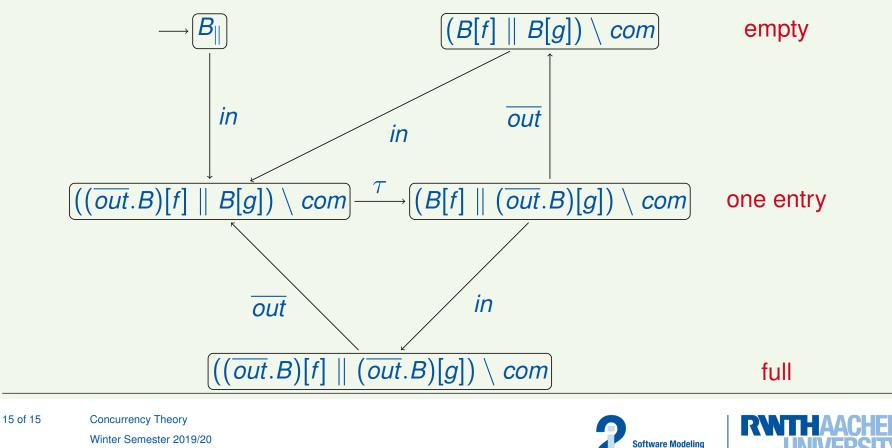


Semantics of CCS IV

Example 2.5 (continued)

Complete LTS of parallel two-place buffer:

Lecture 2: Calculus of Communicating Systems (CCS)



d Verification Chair