Modeling Concurrent and Probabilistic Systems

Lecture 3: Equivalence of CCS Processes

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

Outline

1. Repetition: Syntax and Semantics of CCS
2. Recursive Processes
3. Equivalence of CCS Processes

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:

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

  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), $a_{ij} \in N$ ($a_{i1}, \ldots, a_{in_i}$ pairwise different), and $P_i \in Prc$ (with process identifiers from $\{A_1, \ldots, A_k\}$).

Repetition: Labeled Transition Systems

Goal: represent behavior of system by (infinite) graph

- nodes = system states
- edges = transitions between states

Definition (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

Repetition: Semantics of CCS I

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

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

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

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

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

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

Example

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 (continued)

Complete LTS of parallel two-place buffer:

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

Outline

1. Repetition: Syntax and Semantics of CCS
2. Recursive Processes
3. Equivalence of CCS Processes

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

(nested scalar recursion)

Advantage: only process term level required (no equations)

$\Longrightarrow$ simplification of theory

Disadvantage: bad readability of process definitions

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

(nested scalar recursion)

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

Disadvantage: bad readability of process definitions

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

(nested scalar recursion)

Advantage: only process term level required (no equations)

$\implies$ simplification of theory

Disadvantage: bad readability of process definitions

Outline

1. Repetition: Syntax and Semantics of CCS

2. Recursive Processes

3. Equivalence of CCS Processes

Equivalence Relations

Goal: identify process expressions which have the same "meaning" but differ in their syntax

Definition 3.1 (Equivalence relation)

Let $\cong \subseteq S \times S$ be a binary relation over some set S. Then $\cong$ is called an equivalence relation if it is

- reflexive, i.e., $s \cong s$ for every $s \in S$,
- symmetric, i.e., $s \cong t$ implies $t \cong s$ for every $s, t \in S$, and
- transitive, i.e., $s \cong t$ and $t \cong u$ implies $s \cong u$ for every $s, t, u \in S$.

Equivalence of CCS Processes

- **Generally:** two syntactic objects are equivalent if they have the same "meaning"
- **Here:** two processes are equivalent if they have the same "behavior" (i.e., communication potential)
- Communication potential described by LTS
- **Idea:** define (for processes P, Q)

$$P \cong Q \text{ iff } LTS(P) = LTS(Q)$$

- **But:** yields too many distinctions:

Example 3.2

$$X(a) = a.X(a) \quad Y(a) = a.a.Y(a)$$

LTS:

although both processes can (only) execute infinitely many a-actions, and should be considered equivalent therefore

Equivalence of CCS Processes

- **Generally:** two syntactic objects are equivalent if they have the same "meaning"
- **Here:** two processes are equivalent if they have the same "behavior" (i.e., communication potential)
- Communication potential described by LTS
- **Idea:** define (for processes P, Q)

$$P \cong Q \text{ iff } LTS(P) = LTS(Q)$$

- **But:** yields too many distinctions:

Example 3.2

$$X(a) = a.X(a) \quad Y(a) = a.a.Y(a)$$

LTS:

although both processes can (only) execute infinitely many a-actions, and should be considered equivalent therefore

Equivalence of CCS Processes

- **Generally:** two syntactic objects are equivalent if they have the same "meaning"
- **Here:** two processes are equivalent if they have the same "behavior" (i.e., communication potential)
- Communication potential described by LTS
- **Idea:** define (for processes P, Q)

$$P \cong Q \text{ iff } LTS(P) = LTS(Q)$$

- **But:** yields too many distinctions:

Example 3.2

$$X(a) = a.X(a) \quad Y(a) = a.a.Y(a)$$

LTS:

although both processes can (only) execute infinitely many a-actions, and should be considered equivalent therefore

Equivalence of CCS Processes

- **Generally:** two syntactic objects are equivalent if they have the same "meaning"
- **Here:** two processes are equivalent if they have the same "behavior" (i.e., communication potential)
- Communication potential described by LTS
- **Idea:** define (for processes P, Q)

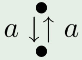
$$P \cong Q \text{ iff } LTS(P) = LTS(Q)$$

- **But:** yields too many distinctions:

Example 3.2

$$X(a) = a.X(a) \quad Y(a) = a.a.Y(a)$$

LTS:



although both processes can (only) execute infinitely many a-actions, and should be considered equivalent therefore

Desired Properties of Equivalence

Wanted: a "feasible" (i.e., efficiently decidable) semantic equivalence between CCS processes which

1. identifies processes whose LTSs coincide,
2. implies trace equivalence, i.e., considers two processes equivalent only if both can execute the same actions sequences (formal definition later), and
3. is a congruence, i.e., allows to replace a subprocess by an equivalent counterpart without changing the overall semantics of the system (formal definition later).

Formally: we are looking for a congruence relation $\cong \subseteq Prc \times Prc$ such that

$$LTS(P) = LTS(Q) \implies P \cong Q \implies Tr(P) = Tr(Q)$$

where $Tr(P)$ is the set of all traces of P (see Def. 4.1)

Desired Properties of Equivalence

Wanted: a "feasible" (i.e., efficiently decidable) semantic equivalence between CCS processes which

1. identifies processes whose LTSs coincide,
2. implies trace equivalence, i.e., considers two processes equivalent only if both can execute the same actions sequences (formal definition later), and
3. is a congruence, i.e., allows to replace a subprocess by an equivalent counterpart without changing the overall semantics of the system (formal definition later).

Formally: we are looking for a congruence relation $\cong \subseteq Prc \times Prc$ such that

$$LTS(P) = LTS(Q) \implies P \cong Q \implies Tr(P) = Tr(Q)$$

where $Tr(P)$ is the set of all traces of P (see Def. 4.1)

Desired Properties of Equivalence

Wanted: a "feasible" (i.e., efficiently decidable) semantic equivalence between CCS processes which

1. identifies processes whose LTSs coincide,
2. implies trace equivalence, i.e., considers two processes equivalent only if both can execute the same actions sequences (formal definition later), and
3. is a congruence, i.e., allows to replace a subprocess by an equivalent counterpart without changing the overall semantics of the system (formal definition later).

Formally: we are looking for a congruence relation $\cong \subseteq Prc \times Prc$ such that

$$LTS(P) = LTS(Q) \implies P \cong Q \implies Tr(P) = Tr(Q)$$

where $Tr(P)$ is the set of all traces of P (see Def. 4.1)

Desired Properties of Equivalence

Wanted: a "feasible" (i.e., efficiently decidable) semantic equivalence between CCS processes which

1. identifies processes whose LTSs coincide,
2. implies trace equivalence, i.e., considers two processes equivalent only if both can execute the same actions sequences (formal definition later), and
3. is a congruence, i.e., allows to replace a subprocess by an equivalent counterpart without changing the overall semantics of the system (formal definition later).

Formally: we are looking for a congruence relation $\cong \subseteq Prc \times Prc$ such that

$$LTS(P) = LTS(Q) \implies P \cong Q \implies Tr(P) = Tr(Q)$$

where $Tr(P)$ is the set of all traces of P (see Def. 4.1)

CCS Congruences

Goal: replacing a subcomponent of a system by an equivalent process should yield an equivalent systems
$\Longrightarrow$ modular system development

Definition 3.3 (CCS congruence)

An equivalence relation $\cong \subseteq Prc \times Prc$ is said to be a CCS congruence if it is preserved by the CCS constructs; that is, if $P, Q, R \in Prc$ such that $P \cong Q$ then

$$\begin{aligned} \alpha.P &\cong \alpha.Q \\ P + R &\cong Q + R \\ R + P &\cong R + Q \\ P \parallel R &\cong Q \parallel R \\ R \parallel P &\cong R \parallel Q \\ \mathsf{new}\, a\, P &\cong \mathsf{new}\, a\, Q \end{aligned}$$

for every $\alpha \in Act$ and $a \in N$.

CCS Congruences

Goal: replacing a subcomponent of a system by an equivalent process should yield an equivalent systems
$\Longrightarrow$ modular system development

Definition 3.3 (CCS congruence)

An equivalence relation $\cong \subseteq Prc \times Prc$ is said to be a CCS congruence if it is preserved by the CCS constructs; that is, if $P, Q, R \in Prc$ such that $P \cong Q$ then

$$\begin{aligned} \alpha.P &\cong \alpha.Q \\ P + R &\cong Q + R \\ R + P &\cong R + Q \\ P \parallel R &\cong Q \parallel R \\ R \parallel P &\cong R \parallel Q \\ \mathsf{new}\, a\, P &\cong \mathsf{new}\, a\, Q \end{aligned}$$

for every $\alpha \in Act$ and $a \in N$.