Semantics and Verification of Software

Lecture 20: Nondeterminism and Parallelism III (Calculus of Communicating Systems)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/svsw11/

Winter Semester 2011/12

Outline

Syntax of CSP

Definition (Syntax of CSP)

The syntax of CSP is given by

$$
\begin{aligned}
a &::= z \mid x \mid a_1{+}a_2 \mid a_1{-}a_2 \mid a_1{*}a_2 \in AExp \\
b &::= t \mid a_1{=}a_2 \mid a_1{>}a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp \\
c &::= \texttt{skip} \mid x := a \mid \alpha?x \mid \alpha!a \mid \\
&\quad\;\; c_1; c_2 \mid \texttt{if}\ gc\ \texttt{fi} \mid \texttt{do}\ gc\ \texttt{od} \mid c_1 \parallel c_2 \in Cmd \\
gc &::= b \rightarrow c \mid b \wedge \alpha?x \rightarrow c \mid b \wedge \alpha!a \rightarrow c \mid gc_1 \,\Box\, gc_2 \in GCmd
\end{aligned}
$$

- In $c_1 \parallel c_2$, statements c_1 and c_2 must not use common variables (only local store)
- Guarded command $gc_1 \,\Box\, gc_2$ represents an alternative
- In $b \rightarrow c$, b acts as a guard that enables the execution of c only if evaluated to true
- $b \wedge \alpha?x \rightarrow c$ and $b \wedge \alpha!a \rightarrow c$ additionally require the respective I/O operation to be enabled
- If none of its alternatives is enabled, a guarded command gc fails (state fail)
- `if` nondeterministically picks an enabled alternative
- A `do` loop is iterated until its body fails

CSP Examples

Example

(on the board)

1. $\texttt{do}\ (\text{true} \wedge \alpha?x \rightarrow \beta!x)\ \texttt{od}$
   describes a process that repeatedly receives a value along α and forwards it along β (i.e., a one-place buffer)
2. $\texttt{do}\ \text{true} \wedge \alpha?x \rightarrow \beta!x\ \texttt{od} \parallel \texttt{do}\ \text{true} \wedge \beta?y \rightarrow \gamma!y\ \texttt{od}$
   specifies a two-place buffer that receives along α and sends along γ (using β for internal communication)
3. Nondeterministic choice between input channels:
   1. $\texttt{if}\ (\text{true} \wedge \alpha?x \rightarrow c_1 \,\Box\, \text{true} \wedge \beta?y \rightarrow c_2)\ \texttt{fi}$
   2. $\texttt{if}\ (\text{true} \rightarrow (\alpha?x; c_1) \,\Box\, \text{true} \rightarrow (\beta?y; c_2))\ \texttt{fi}$

   Expected: progress whenever environment provides data on α or β
   1. correct
   2. incorrect (can deadlock)

Outline

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 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, ...)

$\Rightarrow$ parallel system reduced to communication potential

Syntax of CCS I

Definition 20.1 (Syntax of CCS)

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

Syntax of CCS I

Definition 20.1 (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.

Syntax of CCS I

Definition 20.1 (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.

Syntax of CCS I

Definition 20.1 (Syntax of CCS)

- Let N be a set of (action) names.
- $\overline{N} := \{\bar{a} \mid a \in N\}$ 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 20.1 (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:

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

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

Syntax of CCS II

Definition 20.1 (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), $n_i \in \mathbb{N}$, $a_{ij} \in N$ ($a_{i1}, \ldots, a_{in_i}$ pairwise different), and $P_i \in Prc$ (with process identifiers from $\{A_1, \ldots, A_k\}$).

Syntax of CCS II

Definition 20.1 (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), $n_i \in \mathbb{N}$, $a_{ij} \in N$ ($a_{i1}, \ldots, a_{in_i}$ pairwise different), and $P_i \in Prc$ (with process identifiers from $\{A_1, \ldots, A_k\}$).

Notational Conventions:

- $\overline{\overline{a}}$ means a
- $A(a_1, \ldots, a_n)$ sometimes written as $A(\vec{a})$, $A()$ as A
- prefixing and restriction binds stronger than composition, composition binds stronger than choice:

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

Meaning of CCS Constructs

- 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 $a \in N$ ($\overline{a} \in \overline{N}$) 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 $a \in N$ ($\overline{a} \in \overline{N}$) 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 $a \in N$ ($\overline{a} \in \overline{N}$) 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 $a \in N$ ($\overline{a} \in \overline{N}$) 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 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 $a \in N$ ($\overline{a} \in \overline{N}$) 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 new $a\,P$ declares a as a local name which is only known within P.
- The behavior of a process call $A(a_1, \ldots, a_n)$ is defined by the right-hand side of the corresponding equation where $a_1, \ldots, a_n$ replace the formal name parameters.

CCS Examples

Example 20.2

(on the board)

1. One-place buffer (see Example 19.1(1) for a CSP implementation)
2. Two-place buffer
3. Parallel specification of two-place buffer
   (see Example 19.1(2) for a CSP implementation)

Outline

Definition 20.3 (Semantics of CCS)

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

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

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

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

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

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

Example 20.4

(on the board)

1. One-place buffer:

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

Example 20.4

(on the board)

1. One-place buffer:

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

2. Sequential two-place buffer:

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

Example 20.4

(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:

$$\begin{aligned} B_{\parallel}(in, out) &= \text{new}\, com\, (B(in, com) \parallel B(com, out)) \\ B(in, out) &= in.\overline{out}.B(in, out) \end{aligned}$$

Example 20.4 (continued)

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