Concurrency Theory

Winter Semester 2015/16

Lecture 8: Extensions of CCS: Value Passing and Mobility

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

http://moves.rwth-aachen.de/teaching/ws-1516/ct/

Syntax of Value-Passing CCS

Value-Passing CCS

- **So far:** pure CCS
  - communication = mere synchronisation
  - no (explicit) exchange of data
- **But:** processes usually do pass around data

⇒ Value-passing CCS

- Introduced in Robin Milner: *Communication and Concurrency*, Prentice-Hall, 1989
- Assumption (for simplicity): only integers as data type

Example 8.1 (One-place buffer with data (cf. Example 2.5))

One-place buffer that outputs successor of stored value:

$$B = in(x).B'(x)$$
$$B'(x) = \overline{out}(x+1).B$$

Syntax of Value-Passing CCS I

Definition 8.2 (Syntax of value-passing CCS)

- Let A, $\overline{A}$, Pid (ranked) as in Definition 2.1.
- Let e and b be integer and Boolean expressions, resp., built from integer variables $x, y, \ldots$
- The set Prc^+ of value-passing process expressions is defined by:

$$
\begin{array}{rlr}
P ::= & \mathrm{nil} & \text{(inaction)} \\
| & a(x).P & \text{(input prefixing)} \\
| & \overline{a}(e).P & \text{(output prefixing)} \\
| & \tau.P & \text{(τ prefixing)} \\
| & P_1 + P_2 & \text{(choice)} \\
| & P_1 \parallel P_2 & \text{(parallel composition)} \\
| & P \setminus L & \text{(restriction)} \\
| & P[f] & \text{(relabelling)} \\
| & \text{if } b \text{ then } P & \text{(conditional)} \\
| & C(e_1, \ldots, e_n) & \text{(process call)}
\end{array}
$$

where $a \in A$, $L \subseteq A$, $C \in Pid$ (of rank $n \in \mathbb{N}$), and $f : A \to A$.

Syntax of Value-Passing CCS II

Definition 8.2 (Syntax of value-passing CCS; continued)

A value-passing process definition is an equation system of the form

$$\left(C_i(x_1, \ldots, x_{n_i}) = P_i \mid 1 \leq i \leq k\right)$$

where

- $k \geq 1$,
- $C_i \in Pid$ of rank n_i (pairwise distinct),
- $P_i \in Prc^+$ (with process identifiers from $\{C_1, \ldots, C_k\}$), and
- all occurrences of an integer variable y in each P_i are bound, i.e., $y \in \{x_1, \ldots, x_{n_i}\}$ or y is in the scope of an input prefix of the form $a(y)$ (to ensure well-definedness of values).

Example 8.3

1. $C(x) = \overline{a}(x + 1).b(y).C(y)$ is allowed
2. $C(x) = \overline{a}(x + 1).\overline{a}(y + 2).\mathrm{nil}$ is disallowed as y is not bound

Semantics of Value-Passing CCS I

Definition 8.4 (Semantics of value-passing CCS)

A value-passing process definition $(C_i(x_1, \ldots, x_{n_i}) = P_i \mid 1 \leq i \leq k)$ determines the LTS $(Prc^+, Act, \longrightarrow)$ with $Act := (A \cup \overline{A}) \times \mathbb{Z} \cup \{\tau\}$ whose transitions can be inferred from the following rules ($P, P', Q, Q' \in Prc^+$, $a \in A$, x_i integer variables, e_i/b integer/Boolean expressions, $z \in \mathbb{Z}$, $\alpha \in Act$, $\lambda \in (A \cup \overline{A}) \times \mathbb{Z}$):

$$\text{(In)}\ \frac{}{a(x).P \xrightarrow{a(z)} P[z/x]} \qquad \text{(Out)}\ \frac{(z \text{ value of } e)}{\overline{a}(e).P \xrightarrow{\overline{a}(z)} P} \qquad \text{(Tau)}\ \frac{}{\tau.P \xrightarrow{\tau} P}$$

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

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

Semantics of Value-Passing CCS II

Definition 8.4 (Semantics of value-passing CCS; continued)

$$\text{(Rel)}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad \text{(Res)}\ \frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin (L \cup \overline{L}) \times \mathbb{Z})}{P \setminus L \xrightarrow{\alpha} P' \setminus L}$$

$$\text{(If)}\ \frac{P \xrightarrow{\alpha} P' \quad (b \text{ true})}{\text{if } b \text{ then } P \xrightarrow{\alpha} P'} \qquad \text{(Call)}\ \frac{\begin{array}{c} P[z_1/x_1, \ldots, z_n/x_n] \xrightarrow{\alpha} P' \\ (C(x_1, \ldots, x_n) = P, z_i \text{ value of } e_i) \end{array}}{C(e_1, \ldots, e_n) \xrightarrow{\alpha} P'}$$

Remarks:

- $P[z_1/x_1, \ldots, z_n/x_n]$ denotes the substitution of each free (i.e., unbound) occurrence of x_i by z_i $(1 \leq i \leq n)$
- Relabelling functions are extended to actions by letting

$$f(a(z)) := f(a)(z) \quad \text{and} \quad f(\overline{a}(z)) := \overline{f(a)}(z) \quad (\text{and } f(\tau) := \tau)$$

Semantics of Value-Passing CCS III

Further remarks:

- The binding restriction ensures that all integer and Boolean expressions have a defined value
- The two-armed conditional if b then P else Q can be defined by

$$(\text{if } b \text{ then } P) + (\text{if } \neg b \text{ then } Q)$$

Example 8.5

One-place buffer that outputs non-negative predecessor of stored value:

$$B = in(x).B'(x)$$
$$B'(x) = (\text{if } x = 0 \text{ then } \overline{out}(0).B) + (\text{if } x > 0 \text{ then } \overline{out}(x-1).B)$$

(on the board)

Translation of Value-Passing into Pure CCS I

- **To show:** value-passing process definitions can be represented in pure CCS
- **Idea:** each parametrised construct ($a(x)$, $\overline{a}(e)$, $C(e_1, \ldots, e_n)$) corresponds to a family of constructs in pure CCS, one for each possible integer value
- Requires extension of pure CCS by infinite choices ("$\sum \ldots$"), restrictions, and process definitions

Translation of Value-Passing into Pure CCS II

Definition 8.6 (Translation of value-passing into pure CCS)

For each $P \in Prc^+$ without free variables, its translated form $\widehat{P} \in Prc$ is given by

$$\widehat{\text{nil}} := \text{nil} \qquad \widehat{\tau.P} := \tau.\widehat{P}$$

$$\widehat{a(x).P} := \sum_{z \in \mathbb{Z}} a_z.\widehat{P[z/x]} \qquad \widehat{\overline{a}(e).P} := \overline{a_z}.\widehat{P} \quad (z \text{ value of } e)$$

$$\widehat{P_1 + P_2} := \widehat{P_1} + \widehat{P_2} \qquad \widehat{P_1 \parallel P_2} := \widehat{P_1} \parallel \widehat{P_2}$$

$$\widehat{P \setminus L} := \widehat{P} \setminus \{a_z \mid a \in L, z \in \mathbb{Z}\} \qquad \widehat{P[f]} := \widehat{P}[\widehat{f}] \quad (\widehat{f}(a_z) := f(a)_z)$$

$$\widehat{\text{if } b \text{ then } P} := \begin{cases} \widehat{P} & \text{if } b \text{ true} \\ \text{nil} & \text{otherwise} \end{cases} \qquad \widehat{C(e_1, \ldots, e_n)} := C_{z_1,\ldots,z_n} \quad (z_i \text{ value of } e_i)$$

Moreover, each defining equation $C(x_1, \ldots, x_n) = P$ of a process identifier is translated into the indexed collection of process definitions

$$\left(C_{z_1,\ldots,z_n} = \widehat{P[z_1/x_1, \ldots, z_n/x_n]} \mid v_1, \ldots, v_n \in \mathbb{Z} \right)$$

Translation of Value-Passing into Pure CCS III

Example 8.7 (cf. Example 8.5)

$$B = in(x).B'(x)$$
$$B'(x) = (\text{if } x = 0 \text{ then } \overline{out}(0).B) + (\text{if } x > 0 \text{ then } \overline{out}(x-1).B)$$

(on the board)

Theorem 8.8 (Correctness of translation)

For all $P, P' \in Prc^+$ *and* $\alpha \in Act$*,*

$$P \xrightarrow{\alpha} P' \iff \widehat{P} \xrightarrow{\widehat{\alpha}} \widehat{P'}$$

where $\widehat{a(z)} := a_z$, $\widehat{\overline{a}(z)} := \overline{a}_z$, *and* $\widehat{\tau} := \tau$.

Proof.

by induction on the structure of P (omitted) □

Mobility in Concurrent Systems I

Observation: CCS imposes a static communication structure: if $P, Q \in Prc$ want to communicate, then both must syntactically refer to the same action name

$\Longrightarrow$ every potential communication partner known beforehand, no dynamic passing of communication links

$\Longrightarrow$ lack of modelling capabilities for mobility

Goal: develop calculus in the spirit of CCS which supports mobility

$\Longrightarrow$ π-Calculus

Mobility in Concurrent Systems II

Example 8.9 (Dynamic access to resources)

- Server S controls access to printer P
- Client C wishes to use P
- In CCS: P and C must share some action name a
  $\implies$ C could access P without being granted it by S
- In π-Calculus:
  - initially only S has access to P (using link a)
  - using another link b, C can request access to P
- Formally:

$$\underbrace{\overline{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P}$$
$$\xrightarrow{\tau} S' \parallel \overline{a}\langle d\rangle.C' \parallel a(e).P'$$
$$\xrightarrow{\tau} S' \parallel C' \parallel P'[d/e]$$

- a: link to P
- b: link between S and C
- c: “placeholder” for a
- d: data to be printed
- e: “placeholder” for d

Mobility in Concurrent Systems III

Example 8.9 (Dynamic access to resources; continued)

- Different rôles of action name a:
  - in interaction between S and C: object transferred from S to C
  - in interaction between C and P: name of communication link
- Intuitively, names represent access rights:
  - a: for P
  - b: for S
  - d: for data to be printed
- If a is only way to access P
  $\Longrightarrow$ P "moves" from S to C

Mobile Clients I

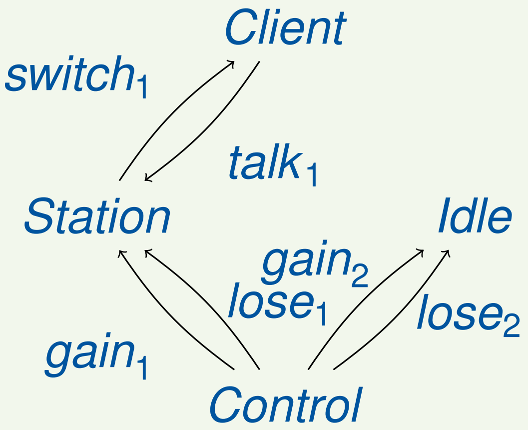
Example 8.10 (Hand-over protocol)

Scenario:

- client devices moving around (phones, PCs, sensors, ...)
- each radio-connected to some base station
- stations wired to central control
- some event (e.g., signal fading) may cause a client to be switched to another station
- essential: specification of switching process (“hand-over protocol”)

Simplest case:

two stations, one client



Another Example: Mobile Clients

Mobile Clients II

Example 8.10 (Hand-over protocol; continued)

- Every station is in one of two modes: *Station* (active; four links) or *Idle* (inactive; two links)
- *Client* can talk via *Station*, and at any time *Control* can request *Station*/*Idle* to lose/gain *Client*:

$$\begin{aligned} Station(talk, switch, gain, lose) &= talk.Station(talk, switch, gain, lose) + \\ &\quad lose(t, s).\overline{switch}\langle t, s\rangle.Idle(gain, lose) \\ Idle(gain, lose) &= gain(t, s).Station(t, s, gain, lose) \end{aligned}$$

- If *Control* decides *Station* to lose *Client*, it issues a new pair of channels to be shared by *Client* and *Idle*:

$$\begin{aligned} Control_1 &= \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2 \\ Control_2 &= \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1 \end{aligned}$$

- *Client* can either talk or, if requested, switch to a new pair of channels:

$$Client(talk, switch) = \overline{talk}.Client(talk, switch) + switch(t, s).Client(t, s)$$

Mobile Clients III

Example 8.10 (Hand-over protocol; continued)

- As usual, the whole system is a restricted composition of processes:

$$System_1 = \text{new}\, L\, (Client_1 \parallel Station_1 \parallel Idle_2 \parallel Control_1)$$

  where

$$\begin{aligned} Client_i &:= Client(talk_i, switch_i) \\ Station_i &:= Station(talk_i, switch_i, gain_i, lose_i) \\ Idle_i &:= Idle(gain_i, lose_i) \\ L &:= (talk_i, switch_i, gain_i, lose_i \mid i \in \{1, 2\}) \end{aligned}$$

- After having formally defined the π-Calculus we will see that this protocol is correct, i.e., that the hand-over does indeed occur:

$$System_1 \longrightarrow^* System_2$$

  where

$$System_2 = \text{new}\, L\, (Client_2 \parallel Idle_1 \parallel Station_2 \parallel Control_2)$$