Models and Semantics of Computation

Jorge A. Pérez
Bernoulli Institute
University of Groningen

October 4, 2023

CCS: A Calculus of Communicating Systems (I)

- informal introduction
- syntax and operational semantics
- value-passing CCS

Acknowledgment

This set of slides was originally produced by Jiri Srba, and makes part of the course material for the book

Reactive Systems: Modelling, Specification and Verification
by L. Aceto, A. Ingolfsdottir, K. G. Larsen and J. Srba
URL: `http://rsbook.cs.aau.dk`

I have adapted them slightly for the purposes of this course.

Classical View

Characterization of a Classical Program

Program transforms an input into an output.

- Denotational semantics:
  a meaning of a program is a partial function

$$states \hookrightarrow states$$

- Nontermination is bad!
- In case of termination, the result is unique.

Is this all we need?

Reactive systems

What about:

- Operating systems?
- Communication protocols?
- Control programs?
- Mobile phones?
- Vending machines?

Reactive systems

Characterization of a Reactive System

Reactive System is a system that computes by reacting to stimuli from its environment.

Key Issues:

- communication and interaction
- parallelism

Nontermination is good!

The result (if any) does not have to be unique.

Reactive systems

Characterization of a Reactive System

Reactive System is a system that computes by reacting to stimuli from its environment.

Key Issues:

- communication and interaction
- parallelism

Nontermination is good!

The result (if any) does not have to be unique.

Analysis of Reactive Systems

Questions

- How can we develop (design) a system that "works"?
- How do we analyze (verify) such a system?

Fact of Life

Even short parallel programs may be hard to analyze.

The Need for a Theory

Conclusion

We need formal/systematic methods (tools), otherwise ...

- Intel's Pentium-II bug in floating-point division unit
- Ariane-5 crash due to a conversion of 64-bit real to 16-bit integer
- Mars Pathfinder
- ...

Classical vs. Reactive Computing

	Classical	Reactive/Parallel
interaction	no	yes
nontermination	undesirable	often desirable
unique result	yes	no
semantics	$states \hookrightarrow states$	**?**

How to Model Reactive Systems

Question

What is the most abstract view of a reactive system (process)?

How to Model Reactive Systems

Question

What is the most abstract view of a reactive system (process)?

Answer

A process performs an action and becomes another process.

Labelled Transition System

Definition

A labelled transition system (LTS) is a triple $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ where

- $Proc$ is a set of states (or processes),
- Act is a set of labels (or actions), and
- for every $a \in Act$, $\xrightarrow{a} \subseteq Proc \times Proc$ is a binary relation on states called the transition relation.

We will use the infix notation $s \xrightarrow{a} s'$ meaning that $(s, s') \in \xrightarrow{a}$.

Sometimes we distinguish the initial (or start) state.

Sequencing, Nondeterminism and Parallelism

LTS explicitly focuses on interaction.

LTS can also describe:

- sequencing $(a; b)$
- choice (nondeterminism) $(a + b)$
- limited notion of parallelism (by using interleaving) $(a \parallel b)$

Binary Relations

Definition

A binary relation $\mathcal{R}$ on a set A is a subset of $A \times A$.

$$\mathcal{R} \subseteq A \times A$$

Sometimes we write $x \, \mathcal{R} \, y$ instead of $(x, y) \in \mathcal{R}$.

Properties

- $\mathcal{R}$ is reflexive if $(x, x) \in \mathcal{R}$ for all $x \in A$
- $\mathcal{R}$ is symmetric if $(x, y) \in \mathcal{R}$ implies that $(y, x) \in \mathcal{R}$ for all $x, y \in A$
- $\mathcal{R}$ is transitive if $(x, y) \in \mathcal{R}$ and $(y, z) \in \mathcal{R}$ implies that $(x, z) \in \mathcal{R}$ for all $x, y, z \in A$

Closures

Let $\mathcal{R}$, $\mathcal{R}'$ and $\mathcal{R}''$ be binary relations on a set A.

Reflexive Closure

$\mathcal{R}'$ is the reflexive closure of $\mathcal{R}$ if and only if

1. $\mathcal{R} \subseteq \mathcal{R}'$,
2. $\mathcal{R}'$ is reflexive, and
3. $\mathcal{R}'$ is the smallest relation that satisfies the two conditions above, i.e., for any relation $\mathcal{R}''$:
   if $\mathcal{R} \subseteq \mathcal{R}''$ and $\mathcal{R}''$ is reflexive, then $\mathcal{R}' \subseteq \mathcal{R}''$.

Closures

Let $\mathcal{R}$, $\mathcal{R}'$ and $\mathcal{R}''$ be binary relations on a set A.

Symmetric Closure

$\mathcal{R}'$ is the symmetric closure of $\mathcal{R}$ if and only if

1. $\mathcal{R} \subseteq \mathcal{R}'$,
2. $\mathcal{R}'$ is symmetric, and
3. $\mathcal{R}'$ is the smallest relation that satisfies the two conditions above, i.e., for any relation $\mathcal{R}''$:
   if $\mathcal{R} \subseteq \mathcal{R}''$ and $\mathcal{R}''$ is symmetric, then $\mathcal{R}' \subseteq \mathcal{R}''$.

Closures

Let $\mathcal{R}$, $\mathcal{R}'$ and $\mathcal{R}''$ be binary relations on a set A.

Transitive Closure

$\mathcal{R}'$ is the transitive closure of $\mathcal{R}$ if and only if

1. $\mathcal{R} \subseteq \mathcal{R}'$,
2. $\mathcal{R}'$ is transitive, and
3. $\mathcal{R}'$ is the smallest relation that satisfies the two conditions above, i.e., for any relation $\mathcal{R}''$:
   if $\mathcal{R} \subseteq \mathcal{R}''$ and $\mathcal{R}''$ is transitive, then $\mathcal{R}' \subseteq R''$.

Labelled Transition Systems – Notation

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

- we extend $\xrightarrow{a}$ to the elements of Act^*
- $\longrightarrow = \bigcup_{a \in Act} \xrightarrow{a}$
- $\longrightarrow^*$ is the reflexive and transitive closure of $\longrightarrow$
- $s \xrightarrow{a}$ and $s \stackrel{a}{\not\longrightarrow}$
- reachable states

How to Describe LTS?

Syntax unknown entity	$\longrightarrow$	Semantics known entity
programming language	$\longrightarrow$	what (denotational) or how (operational) it computes
???	$\longrightarrow$	Labelled Transition Systems

How to Describe LTS?

Syntax unknown entity	$\longrightarrow$	Semantics known entity
programming language	$\longrightarrow$	what (denotational) or how (operational) it computes
???	$\longrightarrow$	Labelled Transition Systems

How to Describe LTS?

Syntax unknown entity	$\longrightarrow$	Semantics known entity
programming language	$\longrightarrow$	what (denotational) or how (operational) it computes
CCS	$\longrightarrow$	Labelled Transition Systems

Calculus of Communicating Systems

CCS

Process calculus called "Calculus of Communicating Systems".

Insight of Robin Milner (1989)

Concurrent (parallel) processes have an algebraic structure.

$$\boxed{P_1} \ op \ \boxed{P_2} \Rightarrow \boxed{P_1 \ op \ P_2}$$

Process Calculus

Basic Principle

1. Define a few atomic processes (modeling the simplest process behavior).
2. Define compositionally new operations (building more complex process behavior from simple ones).

Example

1. atomic instruction: assignment (e.g. x:=2 and x:=x+2)
2. new operators:
   - sequential composition ($P_1; P_2$)
   - parallel composition ($P_1 \parallel P_2$)

   E.g. (x:=1 $\parallel$ x:=2); x:=x+2; (x:=x-1 $\parallel$ x:=x+5) is a process.

Process Calculus

Basic Principle

1. Define a few atomic processes (modeling the simplest process behavior).
2. Define compositionally new operations (building more complex process behavior from simple ones).

Example

1. atomic instruction: assignment (e.g. x:=2 and x:=x+2)
2. new operators:
   - sequential composition ($P_1; P_2$)
   - parallel composition ($P_1 \parallel P_2$)

   E.g. (x:=1 $\parallel$ x:=2); x:=x+2; (x:=x-1 $\parallel$ x:=x+5) is a process.

CCS Basics (Sequential Fragment)

- Nil (or $\mathbf{0}$) process (the only atomic process)
- action prefixing ($a.P$)
- names and recursive definitions ($\stackrel{\mathrm{def}}{=}$)
- nondeterministic choice ($+$)

This is Enough to Describe Sequential Processes

Any finite LTS can be (up to isomorphism) described by using the operations above.

CCS Basics (Sequential Fragment)

- Nil (or $\mathbf{0}$) process (the only atomic process)
- action prefixing ($a.P$)
- names and recursive definitions ($\stackrel{\mathrm{def}}{=}$)
- nondeterministic choice ($+$)

This is Enough to Describe Sequential Processes

Any finite LTS can be (up to isomorphism) described by using the operations above.

CCS Basics (Parallelism and Renaming)

- parallel composition ($\|$)
  (synchronous communication between two components $=$ handshake synchronization)
- restriction of a set of actions ($P \setminus L$)
  alternative notation: $(\nu\tilde{a})P$
- relabelling ($P[f]$)

CCS Basics (Parallelism and Renaming)

- parallel composition ($\parallel$)
  (synchronous communication between two components $=$ handshake synchronization)
- restriction of a set of actions ($P \setminus L$)
  alternative notation: $(\nu \tilde{a})P$
- relabelling ($P[f]$)

CCS Basics (Parallelism and Renaming)

- parallel composition ($\parallel$)
  (synchronous communication between two components = handshake synchronization)
- restriction of a set of actions ($P \setminus L$)
  alternative notation: $(\nu\tilde{a})P$
- relabelling ($P[f]$)

Some Examples

Assigning names to processes (as in procedures) allows us to give recursive definitions of process behaviors.

Some examples:

- $Clock \stackrel{\text{def}}{=} tick.Clock$
- $CM \stackrel{\text{def}}{=} coin.\overline{coffee}.CM$
- $VM \stackrel{\text{def}}{=} coin.\overline{item}.VM$
- $CTM \stackrel{\text{def}}{=} coin.(\overline{coffee}.CTM + \overline{tea}.CTM)$
- $CS \stackrel{\text{def}}{=} \overline{pub}.\overline{coin}.coffee.CS$
- $SmUni \stackrel{\text{def}}{=} (CM \parallel CS) \setminus coin \setminus coffee$

Some Examples, in CAAL

Small CCS processes can be simulated in CAAL:
`http://caal.cs.aau.dk`.

The syntax is very similar to CCS expressions:

```
Clock = tick.Clock;
CM = coin.'coffee.CM;
VM = coin.'item.VM;
CTM = coin.('coffee.CTM + 'tea.CTM);
CS = 'pub.'coin.coffee.CS;
SmUni = (CM | CS) \ {coin,coffee};
```

In CAAL you may “explore” process transitions.

Defining CCS (channels, actions, process names)

Let

- $\mathcal{A}$ be a set of channel names (e.g. tea, $coffee$)

- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels where
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
    ($\mathcal{A}$ are called names and $\overline{\mathcal{A}}$ are called co-names)
  - by convention $\overline{\overline{a}} = a$

- $Act = \mathcal{L} \cup \{\tau\}$ is the set of actions where
  - τ is the internal or silent action

  (e.g. τ, tea, $\overline{coffee}$ are actions)

- $\mathcal{K}$ is a set of process names (constants) (e.g. CM).

Defining CCS (channels, actions, process names)

Let

- $\mathcal{A}$ be a set of channel names (e.g. tea, $coffee$)
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels where
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
    ($\mathcal{A}$ are called names and $\overline{\mathcal{A}}$ are called co-names)
  - by convention $\overline{\overline{a}} = a$
- $Act = \mathcal{L} \cup \{\tau\}$ is the set of actions where
  - τ is the internal or silent action

  (e.g. τ, tea, $\overline{coffee}$ are actions)
- $\mathcal{K}$ is a set of process names (constants) (e.g. CM).

Defining CCS (channels, actions, process names)

Let

- $\mathcal{A}$ be a set of channel names (e.g. tea, $coffee$)
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels where
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
    ($\mathcal{A}$ are called names and $\overline{\mathcal{A}}$ are called co-names)
  - by convention $\overline{\overline{a}} = a$
- $Act = \mathcal{L} \cup \{\tau\}$ is the set of actions where
  - τ is the internal or silent action

  (e.g. τ, tea, $\overline{coffee}$ are actions)
- $\mathcal{K}$ is a set of process names (constants) (e.g. CM).

Defining CCS (channels, actions, process names)

Let

- $\mathcal{A}$ be a set of channel names (e.g. tea, $coffee$)

- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels where
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
    ($\mathcal{A}$ are called names and $\overline{\mathcal{A}}$ are called co-names)
  - by convention $\overline{\overline{a}} = a$

- $Act = \mathcal{L} \cup \{\tau\}$ is the set of actions where
  - τ is the internal or silent action

  (e.g. τ, tea, $\overline{coffee}$ are actions)

- $\mathcal{K}$ is a set of process names (constants) (e.g. CM).

Definition of CCS (expressions)

$P :=$	K	$\mid$	process constants ($K \in \mathcal{K}$)
	$\alpha.P$	$\mid$	prefixing ($\alpha \in Act$)
	$\sum_{i \in I} P_i$	$\mid$	summation (I is an arbitrary index set)
	$P_1 \parallel P_2$	$\mid$	parallel composition
	$P \smallsetminus L$	$\mid$	restriction ($L \subseteq \mathcal{A}$)
	$P[f]$	$\mid$	relabelling ($f : Act \to Act$) such that

- $f(\tau) = \tau$
- $f(\overline{a}) = \overline{f(a)}$

The set of all terms generated by the abstract syntax is called CCS process expressions (and denoted by $\mathcal{P}$).

Notation

$$P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \qquad Nil = 0 = \sum_{i \in \emptyset} P_i$$

Definition of CCS (expressions)

$$
\begin{array}{rlll}
P := & K & | & \text{process constants } (K \in \mathcal{K}) \\
& \alpha.P & | & \text{prefixing } (\alpha \in Act) \\
& \sum_{i \in I} P_i & | & \text{summation } (I \text{ is an arbitrary index set}) \\
& P_1 \parallel P_2 & | & \text{parallel composition} \\
& P \setminus L & | & \text{restriction } (L \subseteq \mathcal{A}) \\
& P[f] & | & \text{relabelling } (f : Act \to Act) \text{ such that}
\end{array}
$$

- $f(\tau) = \tau$
- $f(\overline{a}) = \overline{f(a)}$

The set of all terms generated by the abstract syntax is called CCS process expressions (and denoted by $\mathcal{P}$).

Notation

$$P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \qquad Nil = 0 = \sum_{i \in \emptyset} P_i$$

Definition of CCS (expressions)

$$
\begin{array}{rlcl}
P := & K & \mid & \text{process constants } (K \in \mathcal{K}) \\
& \alpha.P & \mid & \text{prefixing } (\alpha \in Act) \\
& \sum_{i \in I} P_i & \mid & \text{summation } (I \text{ is an arbitrary index set}) \\
& P_1 \parallel P_2 & \mid & \text{parallel composition} \\
& P \smallsetminus L & \mid & \text{restriction } (L \subseteq \mathcal{A}) \\
& P[f] & \mid & \text{relabelling } (f : Act \rightarrow Act) \text{ such that}
\end{array}
$$

- $f(\tau) = \tau$
- $f(\overline{a}) = \overline{f(a)}$

The set of all terms generated by the abstract syntax is called CCS process expressions (and denoted by $\mathcal{P}$).

Notation

$$P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \qquad Nil = 0 = \sum_{i \in \emptyset} P_i$$

Precedence

Precedence

1. restriction and relabelling (tightest binding)
2. action prefixing
3. parallel composition
4. summation

Example: $R + a.P \parallel b.Q \setminus L$ means $R + \big((a.P) \parallel (b.(Q \setminus L))\big)$.

Precedence

Precedence

1. restriction and relabelling (tightest binding)
2. action prefixing
3. parallel composition
4. summation

Example: $R + a.P \parallel b.Q \setminus L$ means $R + \big((a.P) \parallel (b.(Q \setminus L))\big)$.

Definition of CCS (defining equations)

CCS program

A collection of defining equations of the form

$$K \stackrel{\text{def}}{=} P$$

where $K \in \mathcal{K}$ is a process constant and $P \in \mathcal{P}$ is a CCS process expression.

- Only one defining equation per process constant.
- Recursion is allowed: e.g. $A \stackrel{\text{def}}{=} \overline{a}.A \parallel A$.

Semantics of CCS

Syntax $\longrightarrow$ Semantics

CCS LTS

(collection of defining equations) (labelled transition systems)

HOW?

Semantics of CCS

Syntax
CCS
(collection of defining equations)

$\longrightarrow$

Semantics
LTS
(labelled transition systems)

HOW?

Semantics of CCS

Syntax

CCS

(collection of defining equations)

$\longrightarrow$

Semantics

LTS

(labelled transition systems)

HOW?

Structural Operational Semantics for CCS

Structural Operational Semantics (SOS) – Plotkin 1981

Small-step operational semantics where the behaviour of a system is inferred using syntax driven rules.

Given a collection of CCS defining equations, we define the following LTS $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$:

- $Proc = \mathcal{P}$ (the set of all CCS process expressions)
- $Act = \mathcal{L} \cup \{\tau\}$ (the set of all CCS actions including τ)
- transition relation is given by SOS rules of the form:

$$\text{RULE} \quad \frac{premises}{conclusion} \quad conditions$$

SOS rules for CCS ($\alpha \in Act$, $a \in \mathcal{L}$)

ACT $\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$

SUM$_j$ $\dfrac{P_j \xrightarrow{\alpha} P'_j}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_j}$ $j \in I$

COM1 $\dfrac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$

COM2 $\dfrac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}$

COM3 $\dfrac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$

RES $\dfrac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L}$ $\alpha, \overline{\alpha} \notin L$

REL $\dfrac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$

CON $\dfrac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'}$ $K \stackrel{\text{def}}{=} P$

Deriving Transitions in CCS

Let $A \stackrel{\text{def}}{=} a.A$. Then

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

Deriving Transitions in CCS

Let $A \stackrel{\text{def}}{=} a.A$. Then

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$\text{REL}\ \frac{}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}$$

Deriving Transitions in CCS

Let $A \stackrel{\text{def}}{=} a.A$. Then

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$\text{REL}\ \dfrac{\text{COM1}\ \dfrac{}{(A \parallel \overline{a}.Nil) \parallel b.Nil \xrightarrow{a} (A \parallel \overline{a}.Nil) \parallel b.Nil}}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}$$

Deriving Transitions in CCS

Let $A \stackrel{\text{def}}{=} a.A$. Then

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$\text{REL}\ \dfrac{\text{COM1}\ \dfrac{\text{COM1}\ \dfrac{}{A \parallel \overline{a}.Nil \xrightarrow{a} A \parallel \overline{a}.Nil}}{(A \parallel \overline{a}.Nil) \parallel b.Nil \xrightarrow{a} (A \parallel \overline{a}.Nil) \parallel b.Nil}}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}$$

Deriving Transitions in CCS

Let $A \stackrel{\text{def}}{=} a.A$. Then

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$\text{REL}\ \dfrac{\text{COM1}\ \dfrac{\text{COM1}\ \dfrac{\text{CON}\ \dfrac{}{A \xrightarrow{a} A}\ A \stackrel{\text{def}}{=} a.A}{A \parallel \overline{a}.Nil \xrightarrow{a} A \parallel \overline{a}.Nil}}{(A \parallel \overline{a}.Nil) \parallel b.Nil \xrightarrow{a} (A \parallel \overline{a}.Nil) \parallel b.Nil}}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}$$

Deriving Transitions in CCS

Let $A \stackrel{\mathrm{def}}{=} a.A$. Then

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$\text{REL}\ \dfrac{\text{COM1}\ \dfrac{\text{COM1}\ \dfrac{\text{CON}\ \dfrac{\text{ACT}\ \dfrac{}{a.A \xrightarrow{a} A}}{A \xrightarrow{a} A}\, A \stackrel{\mathrm{def}}{=} a.A}{A \parallel \overline{a}.Nil \xrightarrow{a} A \parallel \overline{a}.Nil}}{(A \parallel \overline{a}.Nil) \parallel b.Nil \xrightarrow{a} (A \parallel \overline{a}.Nil) \parallel b.Nil}}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}$$

LTS of the Process $a.Nil \parallel \overline{a}.Nil$

$$
\begin{tikzcd}
 & a.Nil \parallel \overline{a}.Nil \arrow[dl, "a"'] \arrow[dr, "\overline{a}"] \arrow[dd, "\tau"] & \\
Nil \parallel \overline{a}.Nil \arrow[dr, "\overline{a}"] & & a.Nil \parallel Nil \arrow[dl, "a"'] \\
 & Nil \parallel Nil &
\end{tikzcd}
$$

Value Passing CCS

Main Idea

Handshake synchronization is extended with the possibility to exchange integer values.

$$\overline{pay(6)}.Nil \parallel pay(x).\overline{save(x/2)}.Nil$$

Value Passing CCS

Main Idea

Handshake synchronization is extended with the possibility to exchange integer values.

$$\overline{pay(6)}.Nil \parallel pay(x).\overline{save(x/2)}.Nil$$

$$\downarrow \tau$$

$$Nil \parallel \overline{save(3)}.Nil$$

Value Passing CCS

Main Idea

Handshake synchronization is extended with the possibility to exchange integer values.

$$\overline{pay(6)}.Nil \;\parallel\; pay(x).\overline{save(x/2)}.Nil$$

$$\downarrow \tau$$

$$Nil \;\parallel\; \overline{save(3)}.Nil$$

Parametrized Process Constants

For example: $Bank(total) \stackrel{\text{def}}{=} save(x).Bank(total + x)$.

Value Passing CCS

Main Idea

Handshake synchronization is extended with the possibility to exchange integer values.

$$\overline{pay(6)}.Nil \parallel pay(x).\overline{save(x/2)}.Nil \parallel Bank(100)$$

$$\downarrow \tau$$

$$Nil \parallel \overline{save(3)}.Nil \parallel Bank(100)$$

Parametrized Process Constants

For example: $Bank(total) \stackrel{\text{def}}{=} save(x).Bank(total + x)$.

Value Passing CCS

Main Idea

Handshake synchronization is extended with the possibility to exchange integer values.

$$\overline{pay(6)}.Nil \parallel pay(x).\overline{save(x/2)}.Nil \parallel Bank(100)$$

$$\downarrow \tau$$

$$Nil \parallel \overline{save(3)}.Nil \parallel Bank(100)$$

$$\downarrow \tau$$

$$Nil \parallel Nil \parallel Bank(103)$$

Parametrized Process Constants

For example: $Bank(total) \stackrel{\text{def}}{=} save(x).Bank(total + x)$.

Translation of Value Passing CCS to Standard CCS

Value Passing CCS

$$C \stackrel{\text{def}}{=} in(x).C'(x)$$

$$C'(x) \stackrel{\text{def}}{=} \overline{out(x)}.C$$

$\longrightarrow$

Standard CCS

$$C \stackrel{\text{def}}{=} \sum_{i \in \mathbb{N}} in(i).C'_i$$

$$C'_i \stackrel{\text{def}}{=} \overline{out(i)}.C$$

$$\begin{tikzcd}
C \arrow[d, bend left, "in(x)"] \\
C'(x) \arrow[u, bend left, "\overline{out(x)}"]
\end{tikzcd}$$

symbolic LTS

$$\begin{tikzcd}
 & \cdots & \\
C'_i \arrow[r, bend left, "\overline{out(i)}"] & C \arrow[l, bend left, "in(i)"] \arrow[r, bend left, "in(2)"] \arrow[d, bend left, "in(1)"] & C'_2 \arrow[l, bend left, "\overline{out(2)}"] \\
 & C'_1 \arrow[u, bend left, "\overline{out(1)}"] &
\end{tikzcd}$$

infinite LTS

CCS Has Full Turing Power

Fact

CCS can simulate a computation of any Turing machine.

Remark

Hence CCS is as expressive as any other programming language but its use is to rather describe the behaviour of reactive systems than to perform specific calculations.

CCS Has Full Turing Power

Fact

CCS can simulate a computation of any Turing machine.

Remark

Hence CCS is as expressive as any other programming language but its use is to rather describe the behaviour of reactive systems than to perform specific calculations.