Plan for this part of the course:
- Informal introduction to CCS
- Syntax of CCS
- Semantics of CCS
How to Describe LTSes?

Syntax
unknown entity

programming language

Semantics
known entity

what (denotational) or how (operational) it computes

Labelled Transition Systems

CCS (Milner 1980)
How to Describe LTSes?

Syntax
unknown entity

programming language

Semantics
known entity

what (denotational) or how (operational) it computes

Labelled Transition Systems

CCS (Milner 1980)
How to Describe LTSes?

Syntax
- unknown entity
- programming language

Semantics
- known entity
- what (denotational) or how (operational) it computes

Labelled Transition Systems

CCS (Milner 1980)
How to Describe LTSes?

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>unknown entity</td>
<td>known entity</td>
</tr>
<tr>
<td>programming language</td>
<td>what (denotational) or how (operational) it computes</td>
</tr>
<tr>
<td>???</td>
<td>Labelled Transition Systems</td>
</tr>
</tbody>
</table>

Milner’s Calculus of Communicating Systems (CCS)
Calculus of Communicating Systems

CCS

Process algebra called “Calculus of Communicating Systems”.

Insight of Robin Milner (1980, developed from earlier work)

Concurrent (parallel) processes have an algebraic structure.

\[ P_1 \text{ op } P_2 \Rightarrow P_1 \text{ op } P_2 \]
Process Algebra

**Basic Principle**

1. Define a few **atomic processes** (modelling the simplest process behaviour).
2. Define **new composition operations** (building more complex process behaviour from simpler 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 \((P_1 \mid P_2)\)

Now e.g. \((x := 1 \mid x := 2); x := x + 2; (x := x - 1 \mid x := x + 5)\) is a process.
Process Algebra

Basic Principle

1. Define a few **atomic processes** (modelling the simplest process behaviour).
2. Define **new composition operations** (building more complex process behaviour from simpler 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 \( (P_1 \mid P_2) \)

Now e.g. \( (x:=1 \mid x:=2); x:=x+2; (x:=x-1 \mid x:=x+5) \) is a process.
What is a CCS Process to its Environment?

A CCS process is a computing agent that may communicate with its environment via its interface. Interface = Collection of communication ports/channels, together with an indication of whether they are used for input or output.

Example: A Computer Scientist

Process interface:
- coffee (input port)
- coin, pub (output ports)

Question: How do we describe the behaviour of the “black-box”?
A CCS Process: Black-Box View

What is a CCS Process to its Environment?

A CCS process is a computing agent that may communicate with its environment via its interface.
Interface = Collection of communication ports/channels, together with an indication of whether they are used for input or output.

Example: A Computer Scientist

Process interface: coffee (input port)
coin, pub (output ports)

Question: How do we describe the behaviour of the “black-box”?
CCS Basics (Sequential Fragment)

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

This is Enough to Describe Sequential Processes

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

- \textit{Nil} (or 0) process (the only atomic process)
- action prefixing \((a.P)\)
- names and recursive definitions \((\text{def})\)
- nondeterministic choice \((+\)\)

This is Enough to Describe Sequential Processes

Any finite LTS can be described (up to isomorphism) by using the operations above.
CCS Basics (Parallelism and Renaming)

- parallel composition (|)  
  (synchronous communication between two components = handshake synchronization)
- restriction ($P \setminus L$)
- relabelling ($P[f]$)
CCS Basics (Parallelism and Renaming)

- parallel composition (||)
  (synchronous communication between two components = handshake synchronization)
- restriction (P \setminus L)
- relabelling (P[f])
CCS Basics (Parallelism and Renaming)

- parallel composition ($|)$
  (synchronous communication between two components = handshake synchronization)
- restriction ($P \setminus L$)
- relabelling ($P[f]$)
Let

- $\mathcal{A}$ be a set of **channel names** (e.g. tea, coffee are channel names)
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of **labels** where
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
    - (elements of $\mathcal{A}$ are called names and those of $\overline{\mathcal{A}}$ are called co-names)
    - by convention $\overline{\overline{a}} = a$
- $\text{Act} = \mathcal{L} \cup \{\tau\}$ is the set of **actions** where
  - $\tau$ is the **internal** or **silent** action
    - (e.g. $\tau$, tea, coffee are actions)
- $\mathcal{K}$ is a set of **process names** (constants) (e.g. CM).
Definition of CCS (channels, actions, process names)

Let

- $\mathcal{A}$ be a set of channel names (e.g. tea, coffee are channel names)
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels where
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
  - (elements of $\mathcal{A}$ are called names and those of $\overline{\mathcal{A}}$ are called co-names)
  - by convention $\overline{\overline{a}} = a$

- $\text{Act} = \mathcal{L} \cup \{\tau\}$ is the set of actions where
  - $\tau$ is the internal or silent action
    (e.g. $\tau$, tea, coffee are actions)

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

Let

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

Let

- $\mathcal{A}$ be a set of channel names (e.g. `tea`, `coffee` are channel names)
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels where
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
  (elements of $\mathcal{A}$ are called names and those of $\overline{\mathcal{A}}$ are called co-names)
  - by convention $\overline{\overline{a}} = a$

- $\text{Act} = \mathcal{L} \cup \{\tau\}$ is the set of actions where
  - $\tau$ is the internal or silent action
  (e.g. $\tau$, `tea`, `coffee` are actions)

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

\[ P ::= K \quad \mid \quad \alpha.P \quad \mid \quad \sum_{i \in I} P_i \quad \mid \quad P_1 | P_2 \quad \mid \quad P \setminus L \quad \mid \quad P[f] \]

- process constants \((K \in \mathcal{K})\)
- prefixing \((\alpha \in \text{Act})\)
- summation \((I\) is an arbitrary index set\)
- parallel composition
- restriction \((L \subseteq \mathcal{A})\)
- relabelling \((f : \text{Act} \rightarrow \text{Act})\) such that
  \[
  f(\tau) = \tau \quad \text{and} \quad f(\overline{a}) = f(a)
  \]

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

Notation

\[
P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \quad \text{and} \quad \text{Nil} = 0 = \sum_{i \in \emptyset} P_i
\]
Definition of CCS (expressions)

\[ P := K \quad \mid \quad \alpha.P \quad \mid \quad \sum_{i \in I} P_i \quad \mid \quad P_1 | P_2 \quad \mid \quad P \setminus L \quad \mid \quad P[f] \]

- process constants \((K \in \mathcal{K})\)
- prefixing \((\alpha \in \text{Act})\)
- summation \((I \text{ is an arbitrary index set})\)
- parallel composition
- restriction \((L \subseteq \mathcal{A})\)
- relabelling \((f : \text{Act} \rightarrow \text{Act})\) such that
  \- \(f(\tau) = \tau\)
  \- \(f(\overline{a}) = f(a)\)

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

Notation

\[ P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \quad \quad \text{Nil} = 0 = \sum_{i \in \emptyset} P_i \]
Definition of CCS (expressions)

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

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

Notation

\[ P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \quad \quad Nil = 0 = \sum_{i \in \emptyset} P_i \]
Definition of CCS (expressions)

\[ P := \begin{align*}
K & \mid \text{process constants (} K \in \mathcal{K} \text{)} \\
\alpha.P & \mid \text{prefixing (} \alpha \in \text{Act} \text{)} \\
\sum_{i \in I} P_i & \mid \text{summation (} I \text{ is an arbitrary index set)} \\
P_1|P_2 & \mid \text{parallel composition} \\
P \setminus L & \mid \text{restriction (} L \subseteq \mathcal{A} \text{)} \\
P[f] & \mid \text{relabelling (} f : \text{Act} \rightarrow \text{Act} \text{) such that} \\
& \quad f(\tau) = \tau \\
& \quad f(\overline{a}) = f(a)
\end{align*} \]

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

Notation

\[ P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \quad \text{Nil} = 0 = \sum_{i \in \emptyset} P_i \]
Definition of CCS (expressions)

\[ P ::= K \mid \alpha.P \mid \sum_{i \in I} P_i \mid P_1 \parallel P_2 \mid P \setminus L \mid P[f] \]

- process constants \((K \in \mathcal{K})\)
- prefixing \((\alpha \in \text{Act})\)
- summation \((I\) is an arbitrary index set\)
- parallel composition
- restriction \((L \subseteq A)\)
- relabelling \((f : \text{Act} \rightarrow \text{Act})\) such that
  \[
  f(\tau) = \tau \\
  f(\overline{a}) = f(a)
  \]

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

Notation

\[ P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \]
\[ Nil = 0 = \sum_{i \in \emptyset} P_i \]
Definition of CCS (expressions)

\[ P := K \mid \alpha.P \mid \sum_{i \in I} P_i \mid P_1 \parallel P_2 \mid P \setminus L \mid P[f] \]

- Process constants \((K \in \mathcal{K})\)
- Prefixing \((\alpha \in \text{Act})\)
- Summation \((I)\) is an arbitrary index set
- Parallel composition
- Restriction \((L \subseteq \mathcal{A})\)
- Relabelling \((f : \text{Act} \rightarrow \text{Act})\) such that
  - \(f(\tau) = \tau\)
  - \(f(\overline{a}) = f(a)\)

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

**Notation**

\[ P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \quad \text{Nil} = 0 = \sum_{i \in \emptyset} P_i \]
Definition of CCS (expressions)

\[ P ::= K \mid \alpha.P \mid \sum_{i \in I} P_i \mid P_1|P_2 \mid P \setminus L \mid P[f] \]

- \( P \ combustion (K \in K) \)
- \( \alpha.P \) prefixing (\( \alpha \in Act \) )
- \( \sum_{i \in I} P_i \) summation (\( I \) is an arbitrary index set)
- \( P_1|P_2 \) parallel composition
- \( P \setminus L \) restriction (\( L \subseteq A \) )
- \( P[f] \) relabelling (\( f : Act \rightarrow Act \) ) such that
  - \( f(\tau) = \tau \)
  - \( f(\overline{a}) = f(a) \)

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

Notation

\[ P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \quad \text{Nil} = 0 = \sum_{i \in \emptyset} P_i \]
Precedence

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

Example: $R + a.P|b.Q \setminus L$ means $R + ((a.P)\mid(b.(Q \setminus L))).$
Precedence

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

Example: \( R + a.P \mid b.Q \setminus L \) means \( R + ((a.P)\parallel (b.(Q \setminus L))) \).
Definition of CCS (defining equations)

CCS program

A collection of defining equations of the form

\[ K \overset{\text{def}}{=} P \]

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. \( A \overset{\text{def}}{=} \overline{a}.A \mid A \).
Semantics of CCS

Syntax

**CCS**
(collection of defining equations)

Semantics

**LTS**
(labelled transition systems)

HOW?
Semantics of CCS

**Syntax**

CCS
(collection of defining equations)

**Semantics**

LTS
(labelled transition systems)

**HOW?**
Semantics of CCS

Syntax

CCS
(collection of defining equations)

Semantics

LTS
(labelled transition systems)

HOW?
Structural Operational Semantics (SOS)—G. 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 \((\text{Proc}, \text{Act}, \{\xrightarrow{a}\mid a \in \text{Act}\})\):

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

\[
\text{RULE} \quad \frac{\text{premises}}{\text{conclusion}} \quad \text{conditions}
\]
Structural Operational Semantics for CCS

Structural Operational Semantics (SOS)—G. 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 \( (\text{Proc}, \text{Act}, \{ \frac{a}{\text{Proc}} \mid a \in \text{Act} \}) \):

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

\[
\text{RULE}\quad \begin{array}{c}
\text{premises} \\
\hline
\text{conclusion} \\
\end{array} \quad \text{conditions}
\]
SOS rules for CCS \((\alpha \in \text{Act}, \ a \in \mathcal{L})\)

- **ACT**
  \[
  \alpha . P \xrightarrow{\alpha} P 
  \]

- **SUM**
  \[
  \sum_{i \in I} P_i \xrightarrow{\alpha} P'_i 
  \]

- **COM1**
  \[
  P \xrightarrow{\alpha} P' \rightarrow P | Q \xrightarrow{\alpha} P' | Q 
  \]

- **COM2**
  \[
  Q \xrightarrow{\alpha} Q' \rightarrow P | Q \xrightarrow{\alpha} P | Q' 
  \]

- **COM3**
  \[
  P \xrightarrow{a} P' \rightarrow Q \xrightarrow{\overline{a}} Q' \rightarrow P | Q \xrightarrow{\tau} P' | Q' 
  \]

- **RES**
  \[
  P \xrightarrow{\alpha} P' \rightarrow P \setminus L \xrightarrow{\alpha} P' \setminus L \quad \alpha, \overline{\alpha} \notin \mathcal{L} 
  \]

- **REL**
  \[
  P \xrightarrow{\alpha} P' \rightarrow P[f] \xrightarrow{f(\alpha)} P'[f] 
  \]

- **CON**
  \[
  P \xrightarrow{\alpha} P' \rightarrow K \xrightarrow{\alpha} P' \quad K \overset{\text{def}}{=} P 
  \]
Let $A \overset{\text{def}}{=} a.A$. Then

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

Why?
Let $A \overset{\text{def}}{=} a\cdot A$. Then

$$((A \mid a.\text{Nil}) \mid b.\text{Nil})[c/a] \xrightarrow{c} ((A \mid a.\text{Nil}) \mid b.\text{Nil})[c/a].$$

**Why?**

\[
\text{REL} \quad \frac{((A \mid a.\text{Nil}) \mid b.\text{Nil})[c/a] \xrightarrow{c} ((A \mid a.\text{Nil}) \mid b.\text{Nil})[c/a]}{}
\]
Deriving Transitions in CCS

Let \( A \overset{\text{def}}{=} a.A \). Then

\[
((A | \overline{a}.\text{Nil}) | b.\text{Nil})[c/a] \xrightarrow{c} ((A | \overline{a}.\text{Nil}) | b.\text{Nil})[c/a].
\]

Why?

\[
\begin{align*}
\text{COM1} & : (A | \overline{a}.\text{Nil}) | b.\text{Nil} \xrightarrow{a} (A | \overline{a}.\text{Nil}) | b.\text{Nil} \\
\text{REL} & : ((A | \overline{a}.\text{Nil}) | b.\text{Nil})[c/a] \xrightarrow{c} ((A | \overline{a}.\text{Nil}) | b.\text{Nil})[c/a]
\end{align*}
\]
Let $A \overset{\text{def}}{=} a.A$. Then

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

Why?

\[
\begin{align*}
\text{COM1} & \quad A \mid \overline{a}.Nil \xrightarrow{a} A \mid \overline{a}.Nil \\
\text{COM1} & \quad (A \mid \overline{a}.Nil) \mid b.Nil \xrightarrow{a} (A \mid \overline{a}.Nil) \mid b.Nil \\
\text{REL} & \quad ((A \mid \overline{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{\cdot} ((A \mid \overline{a}.Nil) \mid b.Nil)[c/a]
\end{align*}
\]
Let $A \overset{\text{def}}{=} a.A$. Then

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

**Why?**

\[
\begin{align*}
\text{CON} & \quad A \xrightarrow{a} A  \\
\text{COM1} & \quad A \overset{\text{def}}{=} a.A \\
\text{COM1} & \quad A | \overline{a}.\text{Nil} \xrightarrow{a} A | \overline{a}.\text{Nil} \\
\text{REL} & \quad ((A | \overline{a}.\text{Nil}) | b.\text{Nil}) \xrightarrow{a} (A | \overline{a}.\text{Nil}) | b.\text{Nil} \\
& \quad ((A | \overline{a}.\text{Nil}) | b.\text{Nil})[c/a] \xrightarrow{c} ((A | \overline{a}.\text{Nil}) | b.\text{Nil})[c/a].
\end{align*}
\]
Let $A \overset{\text{def}}{=} a.A$. Then

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

Why?

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT</td>
<td>$a.A \xrightarrow{a} A$</td>
<td>$A \overset{\text{def}}{=} a.A$</td>
</tr>
<tr>
<td>CON</td>
<td>$A \xrightarrow{a} A$</td>
<td>$A \overset{\text{def}}{=} a.A$</td>
</tr>
<tr>
<td>COM1</td>
<td>$A \mid \overline{a}.\text{Nil} \xrightarrow{a} A \mid \overline{a}.\text{Nil}$</td>
<td>$A \mid \overline{a}.\text{Nil} \xrightarrow{a} A \mid \overline{a}.\text{Nil}$</td>
</tr>
<tr>
<td>COM1</td>
<td>$A \mid \overline{a}.\text{Nil} \xrightarrow{a} A \mid \overline{a}.\text{Nil}$</td>
<td>$A \mid \overline{a}.\text{Nil} \xrightarrow{a} A \mid \overline{a}.\text{Nil}$</td>
</tr>
<tr>
<td>REL</td>
<td>$((A \mid \overline{a}.\text{Nil}) \mid b.\text{Nil})[c/a] \xrightarrow{c} ((A \mid \overline{a}.\text{Nil}) \mid b.\text{Nil})[c/a]$</td>
<td>$((A \mid \overline{a}.\text{Nil}) \mid b.\text{Nil})[c/a] \xrightarrow{c} ((A \mid \overline{a}.\text{Nil}) \mid b.\text{Nil})[c/a]$</td>
</tr>
</tbody>
</table>
LTS of the Process $a.Net | \overline{a}.Nil$
Handshake synchronization is extended with the possibility to exchange data (e.g., integers).

\[
pay(6).\text{Nil} \mid pay(x).\text{save}(x/2).\text{Nil}
\]
Main Idea

Handshake synchronization is extended with the possibility to exchange data (e.g., integers).

\[
\text{pay}(6).\text{Nil} \mid \text{pay}(x).\text{save}(x/2).\text{Nil} \\
\downarrow \tau \\
\text{Nil} \mid \text{save}(3).\text{Nil}
\]
Value Passing CCS

Main Idea

Handshake synchronization is extended with the possibility to exchange data (e.g., integers).

\[
\overline{\text{pay}(6).\text{Nil}} \mid \overline{\text{pay}(x).\text{save}(x/2).\text{Nil}} \downarrow \tau \\
\overline{\text{Nil}} \mid \overline{\text{save}(3).\text{Nil}}
\]

Parametrized Process Constants

For example: \(\text{Bank}(total) \overset{\text{def}}{=} \text{save}(x).\text{Bank}(total + x)\).
Main Idea

Handshake synchronization is extended with the possibility to exchange data (e.g., integers).

\[
\text{pay}(6).\text{Nil} \; | \; \text{pay}(x).\text{save}(x/2).\text{Nil} \; | \; \text{Bank}(100)
\]
\[
\downarrow \tau
\]
\[
\text{Nil} \; | \; \text{save}(3).\text{Nil} \; | \; \text{Bank}(100)
\]

Parametrized Process Constants

For example: \( \text{Bank}(\text{total}) \overset{\text{def}}{=} \text{save}(x).\text{Bank}(\text{total} + x) \).
Main Idea

Handshake synchronization is extended with the possibility to exchange data (e.g., integers).

\[
\text{pay}(6).\text{Nil} \mid \text{pay}(x).\text{save}(x/2).\text{Nil} \mid \text{Bank}(100)
\downarrow \tau
\text{Nil} \mid \text{save}(3).\text{Nil} \mid \text{Bank}(100)
\downarrow \tau
\text{Nil} \mid \text{Nil} \mid \text{Bank}(103)
\]

Parametrized Process Constants

For example: \(\text{Bank}(\text{total}) \overset{\text{def}}{=} \text{save}(x).\text{Bank}(\text{total} + x)\).
Translation of Value Passing CCS to Standard CCS

Value Passing CCS

\[ C \overset{\text{def}}{=} \text{in}(x).C'(x) \]

\[ C'(x) \overset{\text{def}}{=} \text{out}(x).C \]

Symbolic LTS

\[ \stackrel{\text{out}(x)}{C'}(x) \]

\[ \text{in}(x) \]

\[ \text{out}(x) \]

\[ C \]

Standard CCS

\[ C \overset{\text{def}}{=} \sum_{i \in \mathbb{N}} \text{in}(i).C'_i \]

\[ C'_i \overset{\text{def}}{=} \text{out}(i).C \]

Infinite LTS

\[ \stackrel{\text{out}(i)}{C'_i} \]

\[ \cdots \]

\[ \text{in}(2) \]

\[ \text{out}(1) \]

\[ \text{in}(1) \]

\[ \text{out}(2) \]

\[ C \]

\[ C_1 \]

\[ C_2 \]
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.