An Introduction to Process Calculi

Gabriele Cecilia

University of Milan
Dept. of Mathematics

13 December 2023

Table of Contents

Introduction

Concurrent systems are systems (programs, problems, environments, ...) in which different units or processes can perform operations independently or simultaneously with each other.

Examples: concurrent programming, railway systems, ...

Problems: deadlock, correctness, efficiency, ...

Introduction

Process calculi (or algebras) are mathematical models which allow to formally describe concurrent systems.

They provide:

- formal specification of processes
- description of process behaviour and interactions
- analysis and verification of process properties

Table of Contents

Calculus of Communicating Systems (R. Milner, 1980)

In CCS each process is a black box, with a name and a collection of communicating ports (*channels*) used for input/output.

Operational semantics can be given in terms of automata: processes can be represented by vertices of certain labelled oriented graphs, and a change of process state caused by performing an action can be seen as moving along a certain edge.

Calculus of Communicating Systems (R. Milner, 1980)

In CCS each process is a black box, with a name and a collection of communicating ports (*channels*) used for input/output.

Operational semantics can be given in terms of automata: processes can be represented by vertices of certain labelled oriented graphs, and a change of process state caused by performing an action can be seen as moving along a certain edge.

It can be used to describe simple situations in which different units interact with each other. The focus is on *interactions* themselves, rather than data exchanged.

Examples: people using vending machines, vehicles using toll booths, ...

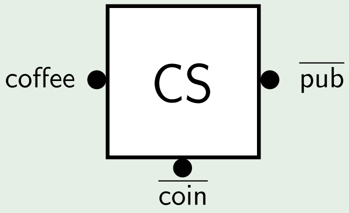
Calculus of Communicating Systems (R. Milner, 1980)

More specifically, CCS models *communication*, *parallel composition* of processes, *choice* of actions, *relabeling* and *scope restriction*; its base presentation doesn't support exchange of data between processes or scope extension.

It can be extended so that process are enabled to send or receive some sort of data *(Value Passing CCS)*.

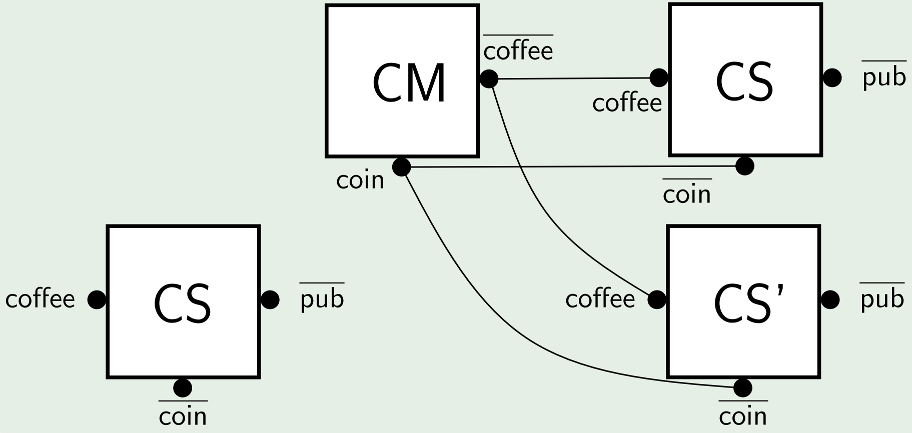
CCS

Examples of processes



CCS

Examples of processes



Let's assume the existence of a countably infinite set $\mathcal{A}$ of (channel) names.

We denote the set of complementary names as $\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$.

Let's assume the existence of a countably infinite set $\mathcal{A}$ of (channel) names.

We denote the set of complementary names as $\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$.

Definition

We define the set of actions as $Act = \mathcal{A} \cup \bar{\mathcal{A}} \cup \{\tau\}$.

Let's assume the existence of a countably infinite set $\mathcal{A}$ of (channel) names.

We denote the set of complementary names as $\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$.

Definition

We define the set of actions as $Act = \mathcal{A} \cup \bar{\mathcal{A}} \cup \{\tau\}$.

Actions given by channel names represent *input actions*.
Actions given by overlined channel names represent *output actions*.
τ is a standard label which represents *communication* between two processes changing their state simultaneously.

CCS
Syntax

Definition

The syntax of CCS processes is the following:

$$P, Q := 0 \mid K \mid \alpha.P \mid P + Q \mid P \mid Q \mid P[f] \mid P \setminus L$$

where K is a constant process, $\alpha \in Act$, f is a function from Act to Act such that $f(\bar{a}) = \overline{f(a)}$ and $f(\tau) = \tau$, $L \subseteq \mathcal{A} \cup \bar{\mathcal{A}}$.

Definition

The syntax of CCS processes is the following:

$$P, Q := 0 \mid K \mid \alpha.P \mid P + Q \mid P \mid Q \mid P[f] \mid P \setminus L$$

where K is a constant process, $\alpha \in Act$, f is a function from Act to Act such that $f(\bar{a}) = \overline{f(a)}$ and $f(\tau) = \tau$, $L \subseteq \mathcal{A} \cup \bar{\mathcal{A}}$.

We denote the set of CCS processes as *Proc*.

We also allow definitions of constant process through their (eventually recursive) defining equation $K \overset{\text{def}}{=} P$.

CCS
Syntax

Examples of processes

$\mathsf{CM} \overset{\text{def}}{=} \mathsf{coin}.\overline{\mathsf{coffee}}.\mathsf{CM}$
$\mathsf{CS} \overset{\text{def}}{=} \overline{\mathsf{pub}}.\overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS}$

Examples of processes

$\mathrm{CM} \overset{\mathrm{def}}{=} \mathrm{coin}.\overline{\mathrm{coffee}}.\mathrm{CM}$
$\mathrm{CS} \overset{\mathrm{def}}{=} \overline{\mathrm{pub}}.\overline{\mathrm{coin}}.\mathrm{coffee}.\mathrm{CS}$

$\mathrm{CTM} \overset{\mathrm{def}}{=} \mathrm{coin}.(\overline{\mathrm{coffee}}.\mathrm{CTM} + \overline{\mathrm{tea}}.\mathrm{CTM})$
$\mathrm{CM} \mid \mathrm{CS} \overset{\mathrm{def}}{=} (\mathrm{coin}.\overline{\mathrm{coffee}}.\mathrm{CM}) \mid (\overline{\mathrm{pub}}.\overline{\mathrm{coin}}.\mathrm{coffee}.\mathrm{CS})$

Examples of processes

$\mathsf{CM} \overset{\text{def}}{=} \mathsf{coin}.\overline{\mathsf{coffee}}.\mathsf{CM}$
$\mathsf{CS} \overset{\text{def}}{=} \overline{\mathsf{pub}}.\overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS}$

$\mathsf{CTM} \overset{\text{def}}{=} \mathsf{coin}.(\overline{\mathsf{coffee}}.\mathsf{CTM} + \overline{\mathsf{tea}}.\mathsf{CTM})$

$\mathsf{CM} \mid \mathsf{CS} \overset{\text{def}}{=} (\mathsf{coin}.\overline{\mathsf{coffee}}.\mathsf{CM}) \mid (\overline{\mathsf{pub}}.\overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS})$

$\mathsf{VM} \overset{\text{def}}{=} \mathsf{coin}.\overline{\mathsf{item}}.\mathsf{VM} \quad \rightarrow \quad \mathsf{CM} \overset{\text{def}}{=} \mathsf{VM}[\mathsf{coffee}/\mathsf{item}]$

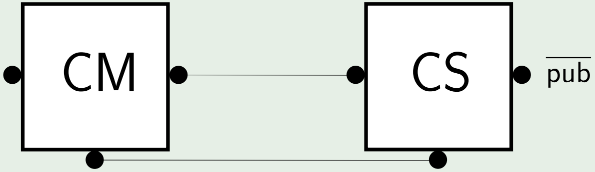
Examples of processes

$\mathsf{SM} \overset{\text{def}}{=} (\mathsf{CM} \mid \mathsf{CS}) \setminus \{\mathsf{coin}, \mathsf{coffee}\}$

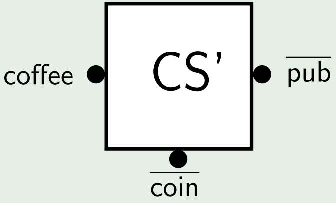
CCS

Examples of processes

SM $\overset{\text{def}}{=}$ (CM | CS) \ {coin, coffee}



SM | CS'



CCS

Semantics

Process behaviour in CCS is defined through a labelled transition semantics: processes make transitions through actions, returning another process.

Transitions describe change of state of processes after input/output or interaction with other processes.

Process behaviour in CCS is defined through a labelled transition semantics: processes make transitions through actions, returning another process.

Transitions describe change of state of processes after input/output or interaction with other processes.

Definition

We define the transition relation $\xrightarrow{(-)}$ as the smallest relation which is a subset of $Proc \times Act \times Proc$ and satisfies the following rules:

Definition

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \textsc{S-Act} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\ \textsc{S-Sum-L} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}\ \textsc{S-Sum-R}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{S-Par-L} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \textsc{S-Par-R} \qquad \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\bar{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \textsc{S-Com}$$

$$\frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{\alpha} P'[f]}\ \textsc{S-Rel} \qquad \frac{P \xrightarrow{\alpha} P' \qquad \alpha, \bar{\alpha} \notin L}{P \setminus L \xrightarrow{\alpha} P' \setminus L}\ \textsc{S-Res} \qquad \frac{P \xrightarrow{\alpha} P' \qquad K \overset{\text{def}}{=} P}{K \xrightarrow{\alpha} P'}\ \textsc{S-Def}$$

CCS

Semantics

Examples of transitions

- Behaviour of a coffee and tea machine CTM, with $\mathrm{CTM} \overset{\mathrm{def}}{=} \mathrm{coin}.(\overline{\mathrm{coffee}}.\mathrm{CTM} + \overline{\mathrm{tea}}.\mathrm{CTM})$

Examples of transitions

- Behaviour of a coffee and tea machine CTM, with $\text{CTM} \overset{\text{def}}{=} \text{coin}.(\overline{\text{coffee}}.\text{CTM} + \overline{\text{tea}}.\text{CTM})$

First, the machine receives a coin as an input:

$$\textsc{S-Act}\ \frac{}{\text{coin}.(\overline{\text{coffee}}.\text{CTM} + \overline{\text{tea}}.\text{CTM}) \xrightarrow{\text{coin}} \overline{\text{coffee}}.\text{CTM} + \overline{\text{tea}}.\text{CTM}}$$

Examples of transitions

- Behaviour of a coffee and tea machine CTM, with $\mathsf{CTM} \overset{\text{def}}{=} \mathsf{coin}.(\overline{\mathsf{coffee}}.\mathsf{CTM} + \overline{\mathsf{tea}}.\mathsf{CTM})$

  First, the machine receives a coin as an input:

$$\textsc{S-Act}\ \frac{}{\mathsf{coin}.(\overline{\mathsf{coffee}}.\mathsf{CTM} + \overline{\mathsf{tea}}.\mathsf{CTM}) \xrightarrow{\mathsf{coin}} \overline{\mathsf{coffee}}.\mathsf{CTM} + \overline{\mathsf{tea}}.\mathsf{CTM}}$$

  Then, the machine can either output tea or coffee, getting back to the initial configuration:

$$\textsc{S-Sum-R}\ \frac{\overline{\mathsf{tea}}.\mathsf{CTM} \xrightarrow{\overline{\mathsf{tea}}} \mathsf{CTM}}{\overline{\mathsf{coffee}}.\mathsf{CTM} + \overline{\mathsf{tea}}.\mathsf{CTM} \xrightarrow{\overline{\mathsf{tea}}} \mathsf{CTM}}$$

Examples of transitions

- Behaviour of the system {coffee machine, computer scientist}

$$\frac{}{\mathsf{CS} \mid \mathsf{CM}, \text{ with } \mathsf{CM} \mid \mathsf{CS} \stackrel{\text{def}}{=} (\mathsf{coin}.\overline{\mathsf{coffee}}.\mathsf{CM}) \mid (\overline{\mathsf{pub}}.\overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS})}$$

CCS
Semantics

Examples of transitions

- Behaviour of the system {coffee machine, computer scientist}

$\mathsf{CS} \mid \mathsf{CM}$, with $\mathsf{CM} \mid \mathsf{CS} \overset{\text{def}}{=} (\mathsf{coin}.\overline{\mathsf{coffee}}.\mathsf{CM}) \mid (\overline{\mathsf{pub}}.\overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS})$

First, the computer scientist produces a publication:

$$\textsc{S-Par-R}\ \frac{\mathsf{CS} \xrightarrow{\overline{\mathsf{pub}}} \overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS}}{\mathsf{CM} \mid \mathsf{CS} \xrightarrow{\overline{\mathsf{pub}}} \mathsf{CM} \mid \overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS}}$$

Examples of transitions

- Behaviour of the system {coffee machine, computer scientist}

$\underline{\mathsf{CS} \mid \mathsf{CM}, \text{ with } \mathsf{CM} \mid \mathsf{CS} \overset{\text{def}}{=} (\mathsf{coin}.\overline{\mathsf{coffee}}.\mathsf{CM}) \mid (\overline{\mathsf{pub}}.\overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS})}$

Then, the scientist gives a coin to the machine: the two processes are interacting, changing their state simultaneously.

$$\textsc{S-Com}\ \frac{\mathsf{CM} \xrightarrow{\mathsf{coin}} \overline{\mathsf{coffee}}.\mathsf{CM} \qquad \overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS} \xrightarrow{\overline{\mathsf{coin}}} \mathsf{coffee}.\mathsf{CS}}{\mathsf{CM} \mid \overline{\mathsf{coin}}.\mathsf{coffee}.\mathsf{CS} \xrightarrow{\tau} \overline{\mathsf{coffee}}.\mathsf{CM} \mid \mathsf{coffee}.\mathsf{CS}}$$

Examples of transitions

- Behaviour of the system {coffee machine, computer scientist}

$\underline{\text{CS} \mid \text{CM, with CM} \mid \text{CS} \overset{\text{def}}{=} (\text{coin}.\overline{\text{coffee}}.\text{CM}) \mid (\overline{\text{pub}}.\overline{\text{coin}}.\text{coffee}.\text{CS})}$

Finally, the machine gives coffee to the scientist.

$$\textsc{S-Com}\ \frac{\overline{\text{coffee}}.\text{CM} \xrightarrow{\overline{\text{coffee}}} \text{CM} \qquad \text{coffee}.\text{CS} \xrightarrow{\text{coffee}} \text{CS}}{\overline{\text{coffee}}.\text{CM} \mid \text{coffee}.\text{CS} \xrightarrow{\tau} \text{CM} \mid \text{CS}}$$

CCS
Behavioural Equivalence

When can we say that two processes are equivalent?

CCS

Behavioural Equivalence

When can we say that two processes are equivalent?

Two processes are equivalent when they have the *same behaviour*, i.e. when the same *observations* about them can be made.

Equivalent process then have the same properties, such as the possibility of deadlock or making a certain transition.

CCS
Behavioural Equivalence

According to different interpretations of *behaviour* or *observations*, there are different notions of behavioural equivalence.

Strong bisimilarity requires that equivalent processes can make transitions through the same actions, reaching processes which, in turn, are equivalent.

According to different interpretations of *behaviour* or *observations*, there are different notions of behavioural equivalence.

Strong bisimilarity requires that equivalent processes can make transitions through the same actions, reaching processes which, in turn, are equivalent.

According to strong bisimilarity, two processes P and $\tau.P$ are not equivalent; but the τ-labelled transition is *internal* and thus *unobservable*, so we might want these processes to be equivalent.
Weak bisimilarity is a notion of bisimilarity that allows to abstract from internal transitions in process behaviours.

Definition

A binary relation $\mathcal{R}$ over *Proc* is a strong bisimulation if, whenever $P \mathcal{R} Q$ and α is an action:

- if $P \xrightarrow{\alpha} P'$, then there is a process Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \mathcal{R} Q'$;
- if $Q \xrightarrow{\alpha} Q'$, then there is a process P' such that $P \xrightarrow{\alpha} P'$ and $P' \mathcal{R} Q'$.

Definition

A binary relation $\mathcal{R}$ over *Proc* is a strong bisimulation if, whenever $P \mathcal{R} Q$ and α is an action:

- if $P \xrightarrow{\alpha} P'$, then there is a process Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \mathcal{R} Q'$;
- if $Q \xrightarrow{\alpha} Q'$, then there is a process P' such that $P \xrightarrow{\alpha} P'$ and $P' \mathcal{R} Q'$.

Definition

Two processes P and Q are strongly bisimilar ($P \sim Q$) if there is a strong bisimulation that relates them.

It can be shown that strong bisimilarity $\sim$ is *the largest strong bisimulation*: it is a bisimulation itself and it contains every other bisimulation.

It follows that strong bisimulation is an example of coinductive predicate: it is the largest relation satisfying certain conditions.

It can be shown that strong bisimilarity $\sim$ is *the largest strong bisimulation*: it is a bisimulation itself and it contains every other bisimulation.

It follows that strong bisimulation is an example of coinductive predicate: it is the largest relation satisfying certain conditions.

On the other hand, the other relations defined in this presentation (such as the transition relation) are *inductive predicates*.

Example of bisimulation

$$\begin{tikzcd}
& P \arrow[dl, "a"'] \arrow[dr, "a"] & \\
P_1 \arrow[rr, "b"'] & & P_2
\end{tikzcd}$$

$$\begin{tikzcd}
Q \arrow[d, "a"'] \\
Q_1 \arrow[loop right, "b"]
\end{tikzcd}$$

$$\begin{aligned}
P &\overset{\text{def}}{=} a.P_1 + a.P_2 \\
P_1 &\overset{\text{def}}{=} b.P_2 \\
P_2 &\overset{\text{def}}{=} b.P_2
\end{aligned}$$

$$\begin{aligned}
Q &\overset{\text{def}}{=} a.Q_1 \\
Q_1 &\overset{\text{def}}{=} b.Q_1
\end{aligned}$$

CCS

Behavioural Equivalence

Example of bisimulation

$$\begin{tikzcd}
 & P \arrow[dl, "a"'] \arrow[dr, "a"] \arrow[rr, dotted, dash, bend left] & & Q \arrow[d, "a"'] \\
P_1 \arrow[rr, "b"'] \arrow[rrr, dotted, dash, bend left] & & P_2 \arrow[r, dotted, dash, bend left] & Q_1 \arrow[loop below, "b"]
\end{tikzcd}$$

$$\mathcal{R} = \{(P, Q), (P_1, Q_1), (P_2, Q_1)\}$$

Table of Contents

Value Passing CCS
Syntax

In Value Passing CCS, processes can also send and receive data (e.g. natural numbers).

Value Passing CCS
Syntax

In Value Passing CCS, processes can also send and receive data (e.g. natural numbers).

Definition

We define the set of actions (Act) as follows:

$$\alpha, \beta := a(n) \mid \bar{a}(n) \mid \tau \qquad \text{where } a \in Names,\ n \in \mathbb{N}.$$

Value Passing CCS

Syntax

Definition

We define the set of processes ($Proc$) as follows:

$$P, Q := 0 \mid K \mid a(x).P \mid \bar{a}(e).P \mid \tau.P \mid P+Q \mid P \mid Q \mid P[f] \mid P \backslash L$$

where K is a constant process, $a \in Names$, x is a variable, e is an expression in $\mathbb{N}$, f is a function from Act to Act such that $f(\bar{a}) = \overline{f(a)}$ and $f(\tau) = \tau$, $L \subseteq \mathcal{A} \cup \bar{\mathcal{A}}$.

Equations defining constant processes can now carry a parameter, such as in $K(x) \stackrel{\text{def}}{=} P$.

Value Passing CCS

Semantics

Definition

We define the transition relation $\xrightarrow{(-)}$ as the smallest relation which satisfies the same CCS rules, but with the following new operational rules for input/output prefixing and for parametrized constant processes:

$$\textsc{S-In}\ \frac{}{a(x).P \xrightarrow{a(n)} P\{n/x\}}\ n \in \mathbb{N}$$

$$\textsc{S-Out}\ \frac{}{\bar{a}(e).P \xrightarrow{\bar{a}(n)} P}\ e \text{ has value } n$$

$$\textsc{S-Def}\ \frac{P\{n/x\} \xrightarrow{\alpha} P' \qquad K(x) \overset{\text{def}}{=} P}{K(e) \xrightarrow{\alpha} P'}\ e \text{ has value } n$$

Value Passing CCS

Examples of processes and transitions

In Value Passing CCS we can define a simple calculator Calc, which receives a natural number as input and outputs (for example) its successor.

$\mathrm{Calc} \overset{\mathrm{def}}{=} \mathrm{in}(x).\mathrm{Calc}(x)$
$\mathrm{Calc}(x) \overset{\mathrm{def}}{=} \overline{\mathrm{out}}(x+1).\mathrm{Calc}$

Value Passing CCS

Examples of processes and transitions

In Value Passing CCS we can define a simple calculator Calc, which receives a natural number as input and outputs (for example) its successor.

$\mathsf{Calc} \overset{\text{def}}{=} \mathsf{in}(x).\mathsf{Calc}(x)$
$\mathsf{Calc}(x) \overset{\text{def}}{=} \overline{\mathsf{out}}(x+1).\mathsf{Calc}$

The calculator receives the input:

$$\textsc{S-In}\ \frac{}{\mathsf{in}(x).\mathsf{Calc}(x) \xrightarrow{\mathsf{in}(n)} \mathsf{Calc}(n)}$$

Value Passing CCS

Examples of processes and transitions

In Value Passing CCS we can define a simple calculator Calc, which receives a natural number as input and outputs (for example) its successor.

$\mathsf{Calc} \overset{\mathsf{def}}{=} \mathsf{in}(x).\mathsf{Calc}(x)$
$\mathsf{Calc}(x) \overset{\mathsf{def}}{=} \overline{\mathsf{out}}(x+1).\mathsf{Calc}$

Then, the result of the expression "$x+1$" with $n = x$ is evaluated and the calculator outputs the result:

$$\textsc{S-Out}\ \frac{}{\overline{\mathsf{out}}(n+1).\mathsf{Calc} \xrightarrow{\overline{\mathsf{out}}(n+1)} \mathsf{Calc}}$$

Table of Contents

π-calculus (R. Milner, J. Parrow & D. Walker, 1992)

In π-calculus there is no difference between channel names and data exchanged between processes: channel names are transferred along the channels themselves.

π-calculus (R. Milner, J. Parrow & D. Walker, 1992)

In π-calculus there is no difference between channel names and data exchanged between processes: channel names are transferred along the channels themselves.

Therefore, π-calculus models processes whose interconnections change as they interact.

π-calculus
Syntax

Let's assume the existence of an infinite countable set of names, represented by the symbols $x, y, \ldots$.

Definition

The syntax of processes is the following:

$$P, Q := \; 0 \mid x(y).P \mid \bar{x}y.P \mid P + Q \mid P \,|\, Q \mid !P \mid (\nu x)P$$

Let's assume the existence of an infinite countable set of names, represented by the symbols $x, y, \ldots$.

Definition

The syntax of processes is the following:

$$P, Q := 0 \mid x(y).P \mid \bar{x}y.P \mid P + Q \mid P \mid Q \mid !P \mid (\nu x)P$$

Recursive definitions through defining equations have been replaced by replication, and only one name is restricted at a time.

Other process constructs can be added to π-calculus, such as the match operator ($if\ x = y\ then\ P$) or mismatch operator ($if\ x \neq y\ then\ P$).

π-calculus

Semantics

Semantics can be given either through labelled transitions as before, or through congruence and reduction.

π-calculus

Semantics: Labelled Transition Semantics

Definition

We define the set of actions (Act) as follows:

$$\alpha, \beta := \; x(y) \;\mid\; \bar{x}y \;\mid\; \bar{x}(y) \;\mid\; \tau$$

The action $\bar{x}(y)$ represents a bound output action: the bound name y is sent along x.

π-calculus
Semantics: Labelled Transition Semantics

Definition

We define the set of actions (Act) as follows:

$$\alpha, \beta := x(y) \mid \bar{x}y \mid \bar{x}(y) \mid \tau$$

The action $\bar{x}(y)$ represents a bound output action: the bound name y is sent along x.

Definition

Given an action α, we define $\mathrm{n}(\alpha)$, $\mathrm{fn}(\alpha)$ and $\mathrm{bn}(\alpha)$ as the names occurring in α (free/bound respectively).

A similar definition for names occurring in processes can be given.

π-calculus

Semantics: Labelled Transition Semantics

Definition

We define the transition relation $\xrightarrow{(-)}$ as the smallest relation which is a subset of *Proc* $\times$ *Act* $\times$ *Proc* and satisfies the following rules:

$$\textsc{S-In}\ \frac{}{x(z).P \xrightarrow{x(y)} P\{y/z\}} \qquad \textsc{S-Out}\ \frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P} \qquad \textsc{S-Tau}\ \frac{}{\tau.P \xrightarrow{\tau} P}$$

$$\textsc{S-Sum-L}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \textsc{S-Par-L}\ \frac{P \xrightarrow{\alpha} P' \qquad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\textsc{S-Sum-R}\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'} \qquad \textsc{S-Par-R}\ \frac{Q \xrightarrow{\alpha} Q' \qquad \mathrm{bn}(\alpha) \cap \mathrm{fn}(P) = \emptyset}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$$

π-calculus

Semantics: Labelled Transition Semantics

Definition

S-Com-L
$$\frac{P \xrightarrow{\bar{x}y} P' \qquad Q \xrightarrow{x(y)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

S-Com-R
$$\frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\bar{x}y} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

S-Repl
$$\frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

S-Res
$$\frac{P \xrightarrow{\alpha} P' \qquad z \notin \mathrm{n}(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$$

S-Close-L
$$\frac{P \xrightarrow{\bar{x}(z)} P' \qquad Q \xrightarrow{x(z)} Q' \qquad z \notin \mathrm{fn}(Q)}{P \mid Q \xrightarrow{\tau} (\nu z)(P' \mid Q')}$$

S-Open
$$\frac{P \xrightarrow{\bar{x}z} P' \qquad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)} P'}$$

S-Close-R
$$\frac{P \xrightarrow{x(z)} P' \qquad Q \xrightarrow{\bar{x}(z)} Q' \qquad z \notin \mathrm{fn}(P)}{P \mid Q \xrightarrow{\tau} (\nu z)(P' \mid Q')}$$

Examples of transitions

Suppose a server S controls access to a printer P and a client C wish to use it.

Access to the printer is represented by a name z, which is originally *local* to P and S; the server S can send this private link to other processes alongside some channel y.

Examples of transitions

Suppose a server S controls access to a printer P and a client C wish to use it.

Access to the printer is represented by a name z, which is originally *local* to P and S; the server S can send this private link to other processes alongside some channel y.

The system {server, printer} can make the following transition through a bound output action:

$$\textsc{S-Open}\quad \frac{\bar{y}z.S \mid P \xrightarrow{\bar{y}z} S \mid P \qquad y \neq z}{(\nu z)(\bar{y}z.S \mid P) \xrightarrow{\bar{y}(z)} S \mid P}$$

Examples of transitions

The client C can receive the name z through the same channel y; we can represent this through the following transition:

$$\textsc{S-In}\quad \frac{}{y(x).C \xrightarrow{\bar{y}(z)} C\{z/x\}}$$

π-calculus

Examples of transitions

If we consider the system $\{$server, printer, client$\}$ as a whole, we can say that the client received the private link z from the server.

The link z, which used to be local to S and P, is now local to the client as well: the scope of z has changed.
This phenomenon is called scope extension.

π-calculus

Semantics: Labelled Transition Semantics

Examples of transitions

If we consider the system {server, printer, client} as a whole, we can say that the client received the private link z from the server.

The link z, which used to be local to S and P, is now local to the client as well: the scope of z has changed.
This phenomenon is called scope extension.

We can represent this with the following transition:

$$\textsc{S-Close-L}\quad \frac{(\nu z)(\bar{y}z.S \mid P) \xrightarrow{\bar{y}(z)} S \mid P \qquad y(x).C \xrightarrow{\bar{y}z} C\{z/x\}}{(\,(\nu z)(\bar{y}z.S \mid P)\,) \mid y(x).C \xrightarrow{\tau} (\nu z)(\,(S \mid P) \mid C\{z/x\}\,)}$$

π-calculus
Semantics: Reduction Semantics

In reduction semantics, congruence relates syntactically equivalent processes; then processes are reduced up to syntactic equivalence.

π-calculus
Semantics: Reduction Semantics

In reduction semantics, congruence relates syntactically equivalent processes; then processes are reduced up to syntactic equivalence.

Definition

We define the congruence relation $\equiv$ as the smallest relation over *Proc*, closed under compatibility and equivalence laws, satisfying the following axioms:

$$\frac{}{P + (Q + R) \equiv (P + Q) + R}\ \textsc{Sum-Assoc} \qquad \frac{}{P + 0 \equiv P}\ \textsc{Sum-Unit} \qquad \frac{}{P + Q \equiv Q + P}\ \textsc{Sum-Comm}$$

$$\frac{}{P \mid (Q \mid R) \equiv (P \mid Q) \mid R}\ \textsc{Par-Assoc} \qquad \frac{}{P \mid 0 \equiv P}\ \textsc{Par-Unit} \qquad \frac{}{P \mid Q \equiv Q \mid P}\ \textsc{Par-Comm}$$

π-calculus
Semantics: Reduction Semantics

Definition

$$\textsc{Sc-Ext-Zero}\quad \frac{}{(\nu x)0 \;\equiv\; 0}$$

$$\textsc{Sc-Ext-Sum}\quad \frac{x \notin \mathsf{fn}(Q)}{(\nu x)P + Q \;\equiv\; (\nu x)(P + Q)}$$

$$\textsc{Sc-Ext-Par}\quad \frac{x \notin \mathsf{fn}(Q)}{(\nu x)P \mid Q \;\equiv\; (\nu x)(P \mid Q)}$$

$$\textsc{Sc-Ext-Res}\quad \frac{}{(\nu x)(\nu y)P \;\equiv\; (\nu y)(\nu x)P}$$

π-calculus

Semantics: Reduction Semantics

Definition

We define the reduction relation $\rightarrow$ as the smallest relation over *Proc* satisfying the following rules:

$$\textsc{R-Com}\quad \frac{}{(R + \bar{x}y.P) \mid (S + x(z).Q) \;\rightarrow\; P \mid Q\{y/z\}}$$

$$\textsc{R-Par}\quad \frac{P \;\rightarrow\; Q}{P \mid R \;\rightarrow\; Q \mid R}$$

$$\textsc{R-Res}\quad \frac{P \;\rightarrow\; Q}{(\nu x)P \;\rightarrow\; (\nu x)Q}$$

$$\textsc{R-Struct}\quad \frac{P \equiv P' \qquad P' \rightarrow Q' \qquad Q' \equiv Q}{P \rightarrow Q}$$

π-calculus
Semantics: Reduction Semantics

Examples of congruences and reductions

Let's consider the same system {server, printer, client} and describe its behaviour with the reduction semantics.

First, we can work on the process representing the system {server, printer, client}. Since the name z doesn't occur free in $y(x).C$, we have the following congruence:

$$\big(\,(\nu z)(\bar{y}z.S \mid P)\,\big) \mid y(x).C \;\equiv\; (\nu z)\big(\,(\bar{y}z.S \mid P) \mid y(x).C\,\big) \qquad (1)$$

Examples of congruences and reductions

Let's consider the same system {server, printer, client} and describe its behaviour with the reduction semantics.

First, we can work on the process representing the system {server, printer, client}. Since the name z doesn't occur free in $y(x).C$, we have the following congruence:

$$(\,(\nu z)(\bar{y}z.S \mid P)\,) \mid y(x).C \;\equiv\; (\nu z)(\,(\bar{y}z.S \mid P) \mid y(x).C\,) \qquad (1)$$

The last process can make the following reduction through a R-Com rule, followed by a R-Res rule:

$$\frac{}{(\nu z)(\,(\bar{y}z.S \mid P) \mid y(x).C\,) \;\rightarrow\; (\nu z)(\,(S \mid P) \mid C\{z/x\}\,)}$$

π-calculus

Semantics: Reduction Semantics

Examples of congruences and reductions

The process $(\nu z)(\,(S \mid P) \mid C\{z/x\}\,)$ is syntactically equivalent to itself by reflexivity (2).

As a consequence, by applying a $\textsc{R-Struct}$ rule we obtain the following reduction:

$$\frac{(1) \qquad (\nu z)(\,(\bar{y}z.S \mid P) \mid y(x).C\,) \;\rightarrow\; (\nu z)(\,(S \mid P) \mid C\{z/x\}\,) \qquad (2)}{(\,(\nu z)(\bar{y}z.S \mid P)\,) \mid y(x).C \;\rightarrow\; (\nu z)(\,(S \mid P) \mid C\{z/x\}\,)}$$

It can be proved that the two semantics are equivalent, in the sense given by the following theorems:

Theorem 1

$P \xrightarrow{\tau} Q$ implies $P \to Q$.

It can be proved that the two semantics are equivalent, in the sense given by the following theorems:

Theorem 1

$P \xrightarrow{\tau} Q$ implies $P \to Q$.

Theorem 2

$P \to Q$ implies the existence of a Q' such that $P \xrightarrow{\tau} Q'$ and $Q \equiv Q'$.

As in the CCS, behavioural equivalence can be given by bisimilarity.

There are different notions of bisimilarity according to different definitions of observations.

Future works

My thesis work consists in the formalization of the aforementioned definitions and theorems through Beluga.

Future works

My thesis work consists in the formalization of the aforementioned definitions and theorems through Beluga.

Beluga is a functional programming language designed for reasoning about formal systems. It has been developed at the Complogic group at McGill University, led by Professor Brigitte Pientka.

In Beluga, binding constructs are encoded with a *higher order abstract syntax* (HOAS). Beluga also supports *contexts* and *contextual objects*, which can be used for hypothetical and parametric derivations. Proofs in Beluga are represented by *recursive programs*, according to the Curry-Howard isomorphism.

Future works

More specifically, after formalizing CCS definitions, I worked on the formalization of π-calculus syntax and semantics, following Honsell's approach to implement labelled transition semantics.

I'm currently working on the formalization of the theorems regarding the equivalence of the two semantics for π-calculus.

This work contributes to the *Concurrent Calculi Formalization Benchmark*, a set of benchmarks intended to stimulate the research and development of techniques used to formalize process calculi.

Thank You