Process Calculi

A Brief, Gentle Introduction

Jorge A. Pérez

University of Brasilia
July 20, 2015

Acknowledgment

A part of 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 for the purposes of this talk.

Outline

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, 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 $(y, x) \in R$ 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 $x, y, z \in A$

We assume usual definitions of closures (reflexive, symmetric, transitive).

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 $s \not\xrightarrow{a}$
- reachable states

Outline

How to Describe LTS?

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

CCS

How to Describe LTS?

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

CCS

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

CCS

How to Describe LTS?

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

CCS

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 ($\stackrel{\text{def}}{=}$)
- 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 ($\parallel$)
  (synchronous communication between two components = handshake synchronization)
- restriction ($(\nu a_1, \ldots, a_n)P$)
  Alternative notation: $P \setminus L$, with $L = \{a_1, \ldots, a_n\}$
- relabelling ($P[f]$)

CCS Basics (Parallelism and Renaming)

- parallel composition ($\parallel$)
  (synchronous communication between two components = handshake synchronization)
- restriction ($(\nu a_1, \ldots, a_n)P$)
  Alternative notation: $P \setminus L$, with $L = \{a_1, \ldots, a_n\}$
- relabelling ($P[f]$)

CCS Basics (Parallelism and Renaming)

- parallel composition ($\parallel$)
  (synchronous communication between two components = handshake synchronization)
- restriction $((\nu a_1, \ldots, a_n)P)$
  Alternative notation: $P \setminus L$, with $L = \{a_1, \ldots, a_n\}$
- relabelling $(P[f])$

Some Examples

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

Some examples:

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

Definition of CCS

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

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

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

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
	$(\nu a_1, \ldots, a_n)P$	$\mid$	restriction ($\{a_1, \ldots, a_n\} \subseteq \mathcal{A}$)
	$P[f]$	$\mid$	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

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

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

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?

Semantics of CCS

Syntax		Semantics
CCS (collection of defining equations)	$\longrightarrow$	LTS (labelled transition systems)

HOW?

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 $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$:

- $Proc = \mathcal{P}$ (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:

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

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 $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$:

- $Proc = \mathcal{P}$ (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 $A \stackrel{\text{def}}{=} a.A$. Then

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

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

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

From Value Passing CCS to Standard CCS

Value Passing CCS

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

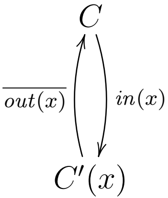
$$C'(x) \stackrel{\text{def}}{=} \overline{out(x)}.C$$

$\longrightarrow$

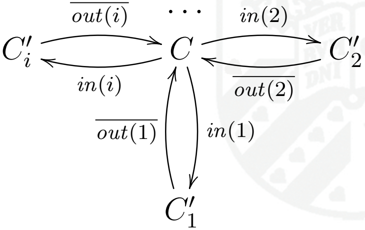
Standard CCS

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

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



symbolic LTS



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.

Behavioural Equivalence

Implementation

$$CM \stackrel{\text{def}}{=} coin.\overline{coffee}.CM$$

$$CS \stackrel{\text{def}}{=} \overline{pub}.\overline{coin}.coffee.CS$$

$$Uni \stackrel{\text{def}}{=} (\nu\, coin, coffee)(CM \parallel CS)$$

Specification

$$Spec \stackrel{\text{def}}{=} \overline{pub}.Spec$$

Question

Are the processes Uni and $Spec$ behaviorally equivalent?

$$Uni \equiv Spec$$

Behavioural Equivalence

Implementation

$$CM \stackrel{\text{def}}{=} coin.\overline{coffee}.CM$$

$$CS \stackrel{\text{def}}{=} \overline{pub}.\overline{coin}.coffee.CS$$

$$Uni \stackrel{\text{def}}{=} (\nu\, coin, coffee)(CM \parallel CS)$$

Specification

$$Spec \stackrel{\text{def}}{=} \overline{pub}.Spec$$

Question

Are the processes Uni and $Spec$ behaviorally equivalent?

$$Uni \equiv Spec$$

Goals

What should a reasonable behavioral equivalence satisfy?

- abstract from states (consider only the behavior – actions)
- abstract from nondeterminism
- abstract from internal behavior

What else should a reasonable behavioural equivalence satisfy?

- reflexivity $P \equiv P$ for any process P
- transitivity $Spec_0 \equiv Spec_1 \equiv Spec_2 \equiv \cdots \equiv Impl$ gives that
$$Spec_0 \equiv Impl$$
- symmetry $P \equiv Q$ iff $Q \equiv P$

Goals

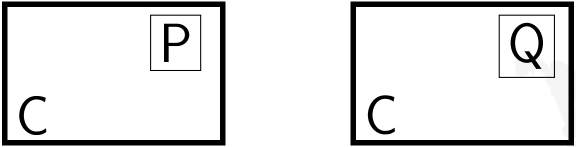
What should a reasonable behavioral equivalence satisfy?

- abstract from states (consider only the behavior – actions)
- abstract from nondeterminism
- abstract from internal behavior

What else should a reasonable behavioural equivalence satisfy?

- reflexivity $P \equiv P$ for any process P
- transitivity $\mathit{Spec}_0 \equiv \mathit{Spec}_1 \equiv \mathit{Spec}_2 \equiv \cdots \equiv \mathit{Impl}$ gives that
$$\mathit{Spec}_0 \equiv \mathit{Impl}$$
- symmetry $P \equiv Q$ iff $Q \equiv P$

Congruence



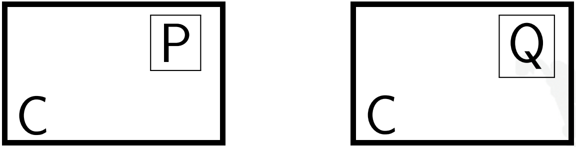
$C(P)$ $C(Q)$

- We would like "equal" processes P and Q to "behave the same" under any context $C(\cdot)$.
- A context is a process with a hole.
  When the hole is filled in with a process P, we obtain another process (usually noted $C(P)$ or $C[P]$).

Congruence Property

$P \equiv Q$ implies that $C(P) \equiv C(Q)$

Congruence



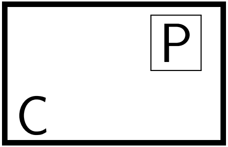
$C(P)$ $C(Q)$

- We would like "equal" processes P and Q to "behave the same" under any context $C(\cdot)$.
- A context is a process with a hole.
  When the hole is filled in with a process P, we obtain another process (usually noted $C(P)$ or $C[P]$).

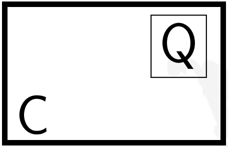
Congruence Property

$P \equiv Q$ implies that $C(P) \equiv C(Q)$

Congruence



$C(P)$



$C(Q)$

- We would like "equal" processes P and Q to "behave the same" under any context $C(\cdot)$.
- A context is a process with a hole.
  When the hole is filled in with a process P, we obtain another process (usually noted $C(P)$ or $C[P]$).

Congruence Property

$$P \equiv Q \text{ implies that } C(P) \equiv C(Q)$$

Trace Equivalence

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

Trace Set for $s \in Proc$

$$Traces(s) = \{w \in Act^* \mid \exists s' \in Proc.\ s \xrightarrow{w} s'\}$$

Let $s \in Proc$ and $t \in Proc$.

Trace Equivalence

We say that s and t are trace equivalent ($s \equiv_t t$) if and only if

$$Traces(s) = Traces(t)$$

Trace Equivalence

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

Trace Set for $s \in Proc$

$$Traces(s) = \{w \in Act^* \mid \exists s' \in Proc.\ s \xrightarrow{w} s'\}$$

Let $s \in Proc$ and $t \in Proc$.

Trace Equivalence

We say that s and t are trace equivalent ($s \equiv_t t$) if and only if

$$Traces(s) = Traces(t)$$

Black-Box Experiments

Main Idea

Two processes are behaviorally equivalent if and only if an external observer cannot tell them apart.

Strong Bisimilarity

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

Strong Bisimulation

A binary relation $\mathcal{R} \subseteq Proc \times Proc$ is a strong bisimulation iff whenever $(s,t) \in \mathcal{R}$ then for each $a \in Act$:

- if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s',t') \in \mathcal{R}$
- if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s',t') \in \mathcal{R}$.

Strong Bisimilarity

Processes $p_1, p_2 \in Proc$ are strongly bisimilar ($p_1 \sim p_2$) if and only if there exists a strong bisimulation $\mathcal{R}$ such that $(p_1, p_2) \in \mathcal{R}$.

$$\sim \; = \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a strong bisimulation}\}$$

Strong Bisimilarity

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

Strong Bisimulation

A binary relation $\mathcal{R} \subseteq Proc \times Proc$ is a strong bisimulation iff whenever $(s, t) \in \mathcal{R}$ then for each $a \in Act$:

- if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s', t') \in \mathcal{R}$
- if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s', t') \in \mathcal{R}$.

Strong Bisimilarity

Processes $p_1, p_2 \in Proc$ are strongly bisimilar ($p_1 \sim p_2$) if and only if there exists a strong bisimulation $\mathcal{R}$ such that $(p_1, p_2) \in \mathcal{R}$.

$$\sim \; = \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a strong bisimulation}\}$$

Basic Properties of Strong Bisimilarity

Theorem

$\sim$ *is an equivalence (reflexive, symmetric and transitive)*

Theorem

$\sim$ *is the largest strong bisimulation*

Theorem

$s \sim t$ *if and only if for each* $a \in Act$*:*

- *if* $s \xrightarrow{a} s'$ *then* $t \xrightarrow{a} t'$ *for some* t' *such that* $s' \sim t'$
- *if* $t \xrightarrow{a} t'$ *then* $s \xrightarrow{a} s'$ *for some* s' *such that* $s' \sim t'$.

Basic Properties of Strong Bisimilarity

Theorem

$\sim$ *is an equivalence (reflexive, symmetric and transitive)*

Theorem

$\sim$ *is the largest strong bisimulation*

Theorem

$s \sim t$ *if and only if for each* $a \in Act$*:*

- *if* $s \xrightarrow{a} s'$ *then* $t \xrightarrow{a} t'$ *for some* t' *such that* $s' \sim t'$
- *if* $t \xrightarrow{a} t'$ *then* $s \xrightarrow{a} s'$ *for some* s' *such that* $s' \sim t'$.

Basic Properties of Strong Bisimilarity

Theorem

$\sim$ *is an equivalence (reflexive, symmetric and transitive)*

Theorem

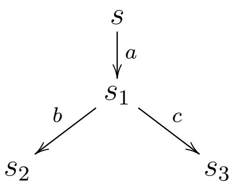
$\sim$ *is the largest strong bisimulation*

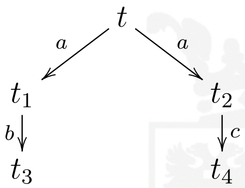
Theorem

$s \sim t$ *if and only if for each* $a \in Act$*:*

- *if* $s \xrightarrow{a} s'$ *then* $t \xrightarrow{a} t'$ *for some* t' *such that* $s' \sim t'$
- *if* $t \xrightarrow{a} t'$ *then* $s \xrightarrow{a} s'$ *for some* s' *such that* $s' \sim t'$.

How to Show Nonbisimilarity?

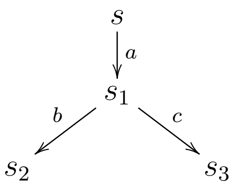


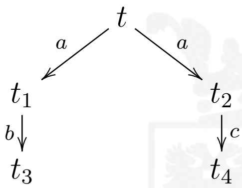


To prove that $s \not\sim t$:

- Enumerate all binary relations and show that none of them at the same time contains (s, t) and is a strong bisimulation. (Expensive: $2^{|Proc|^2}$ relations.)
- Make certain observations which will enable to disqualify many bisimulation candidates in one step.
- Use game characterization of strong bisimilarity.

How to Show Nonbisimilarity?

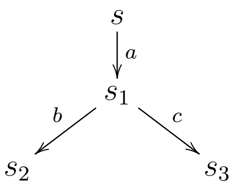


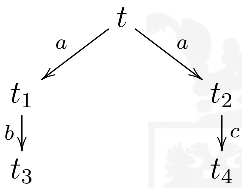


To prove that $s \not\sim t$:

- Enumerate all binary relations and show that none of them at the same time contains (s, t) and is a strong bisimulation. (Expensive: $2^{|Proc|^2}$ relations.)
- Make certain observations which will enable to disqualify many bisimulation candidates in one step.
- Use game characterization of strong bisimilarity.

How to Show Nonbisimilarity?





To prove that $s \not\sim t$:

- Enumerate all binary relations and show that none of them at the same time contains (s, t) and is a strong bisimulation. (Expensive: $2^{|Proc|^2}$ relations.)
- Make certain observations which will enable to disqualify many bisimulation candidates in one step.
- Use game characterization of strong bisimilarity.

How to Show Nonbisimilarity?

$$
\begin{tikzcd}
 & s \arrow[d, "a"] & \\
 & s_1 \arrow[dl, "b"'] \arrow[dr, "c"] & \\
s_2 & & s_3
\end{tikzcd}
\qquad
\begin{tikzcd}
 & t \arrow[dl, "a"'] \arrow[dr, "a"] & \\
t_1 \arrow[d, "b"'] & & t_2 \arrow[d, "c"] \\
t_3 & & t_4
\end{tikzcd}
$$

To prove that $s \not\sim t$:

- Enumerate all binary relations and show that none of them at the same time contains (s, t) and is a strong bisimulation. (Expensive: $2^{|Proc|^2}$ relations.)
- Make certain observations which will enable to disqualify many bisimulation candidates in one step.
- Use game characterization of strong bisimilarity.

Bisimilarity is a Congruence for CCS

Theorem

Let P and Q be CCS processes such that $P \sim Q$. Then

- $\alpha.P \sim \alpha.Q$ *for each action* $\alpha \in Act$
- $P + R \sim Q + R$ *and* $R + P \sim R + Q$ *for each CCS process* R
- $P \mid R \sim Q \mid R$ *and* $R \mid P \sim R \mid Q$ *for each CCS process* R
- $(\nu a)\, P \sim (\nu a)\, Q$ *for any* a.

Other Properties of Strong Bisimilarity

Following Properties Hold for any CCS Processes P, Q and R

- $P + Q \sim Q + P$
- $P \parallel Q \sim Q \parallel P$
- $P + \mathit{Nil} \sim P$
- $P \parallel \mathit{Nil} \sim P$
- $(P + Q) + R \sim P + (Q + R)$
- $(P \parallel Q) \parallel R \sim P \parallel (Q \parallel R)$

Example – Buffer

Buffer of Capacity 1

$$B^1_0 \stackrel{\text{def}}{=} in.B^1_1$$
$$B^1_1 \stackrel{\text{def}}{=} \overline{out}.B^1_0$$

Buffer of Capacity n

$$B^n_0 \stackrel{\text{def}}{=} in.B^n_1$$
$$B^n_i \stackrel{\text{def}}{=} in.B^n_{i+1} + \overline{out}.B^n_{i-1} \quad \text{for } 0 < i < n$$
$$B^n_n \stackrel{\text{def}}{=} \overline{out}.B^n_{n-1}$$

Example: $B^2_0 \sim B^1_0 \parallel B^1_0$

Example – Buffer

Buffer of Capacity 1

$$B^1_0 \stackrel{\text{def}}{=} in.B^1_1$$
$$B^1_1 \stackrel{\text{def}}{=} \overline{out}.B^1_0$$

Buffer of Capacity n

$$B^n_0 \stackrel{\text{def}}{=} in.B^n_1$$
$$B^n_i \stackrel{\text{def}}{=} in.B^n_{i+1} + \overline{out}.B^n_{i-1} \quad \text{for } 0 < i < n$$
$$B^n_n \stackrel{\text{def}}{=} \overline{out}.B^n_{n-1}$$

Example: $B^2_0 \sim B^1_0 \parallel B^1_0$

Example – Buffer

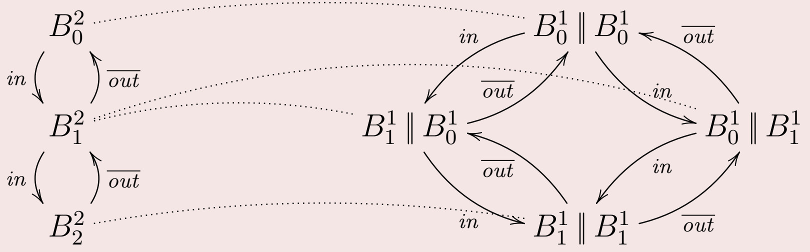
Buffer of Capacity 1

$$B_0^1 \stackrel{\text{def}}{=} in.B_1^1$$
$$B_1^1 \stackrel{\text{def}}{=} \overline{out}.B_0^1$$

Buffer of Capacity n

$$B_0^n \stackrel{\text{def}}{=} in.B_1^n$$
$$B_i^n \stackrel{\text{def}}{=} in.B_{i+1}^n + \overline{out}.B_{i-1}^n \quad \text{for } 0 < i < n$$
$$B_n^n \stackrel{\text{def}}{=} \overline{out}.B_{n-1}^n$$

Example: $B_0^2 \sim B_0^1 \parallel B_0^1$



Example – Buffer

Theorem

For all natural numbers n: $B_0^n \sim \underbrace{B_0^1 \parallel B_0^1 \parallel \cdots \parallel B_0^1}_{n \text{ times}}$

Proof.

The co-inductive proof method: to show bisimilarity, show an appropriate strong bisimulation.

Construct the following binary relation where $i_1, i_2, \ldots, i_n \in \{0, 1\}$.

$$\mathcal{R} = \{ \big(B_i^n,\ B_{i_1}^1 \parallel B_{i_2}^1 \parallel \cdots \parallel B_{i_n}^1 \big) \mid \sum_{j=1}^{n} i_j = i \}$$

- $\big(B_0^n,\ B_0^1 \parallel B_0^1 \parallel \cdots \parallel B_0^1 \big) \in \mathcal{R}$
- $\mathcal{R}$ is strong bisimulation

$\square$

Example – Buffer

Theorem

For all natural numbers n*:* $B_0^n \sim \underbrace{B_0^1 \parallel B_0^1 \parallel \cdots \parallel B_0^1}_{n \text{ times}}$

Proof.

The co-inductive proof method: to show bisimilarity, show an appropriate strong bisimulation.

Construct the following binary relation where $i_1, i_2, \ldots, i_n \in \{0, 1\}$.

$$\mathcal{R} = \{\big(B_i^n,\ B_{i_1}^1 \parallel B_{i_2}^1 \parallel \cdots \parallel B_{i_n}^1\big) \mid \sum_{j=1}^{n} i_j = i\}$$

- $\big(B_0^n,\ B_0^1 \parallel B_0^1 \parallel \cdots \parallel B_0^1\big) \in \mathcal{R}$
- $\mathcal{R}$ is strong bisimulation

$\square$

Strong Bisimilarity – Summary

Properties of $\sim$

- an equivalence relation
- the largest strong bisimulation
- a congruence
- enough to prove some natural rules like
  - $P \parallel Q \sim Q \parallel P$
  - $P \parallel \mathit{Nil} \sim P$
  - $(P \parallel Q) \parallel R \sim Q \parallel (P \parallel R)$
  - $\ldots$

Question

Should we look any further???

Strong Bisimilarity – Summary

Properties of $\sim$

- an equivalence relation
- the largest strong bisimulation
- a congruence
- enough to prove some natural rules like
  - $P \parallel Q \sim Q \parallel P$
  - $P \parallel Nil \sim P$
  - $(P \parallel Q) \parallel R \sim Q \parallel (P \parallel R)$
  - $\dots$

Question

Should we look any further???

Problems with Internal Actions

Question

Does $a.\tau.Nil \sim a.Nil$ hold? NO!

Problem

Strong bisimilarity does not abstract away from τ actions.

Example: SmUni $\not\sim$ Spec

$$
\begin{tikzcd}
\text{SmUni} \arrow[d, "\overline{pub}"] & \not\sim & \text{Spec} \arrow[loop below, "\overline{pub}"] \\
(\nu\, coin, coffee)(CM \parallel CS_1) \arrow[d, "\tau"] & & \\
(\nu\, coin, coffee)(CM_1 \parallel CS_2) \arrow[d, "\tau"] & & \\
(\nu\, coin, coffee)(CM \parallel CS) \arrow[uu, bend right=60, "\overline{pub}"'] & &
\end{tikzcd}
$$

Problems with Internal Actions

Question

Does $a.\tau.Nil \sim a.Nil$ hold? NO!

Problem

Strong bisimilarity does not abstract away from τ actions.

Example: SmUni $\not\sim$ Spec

$$
\begin{tikzcd}
\text{SmUni} \arrow[d, "\overline{pub}"] & \not\sim & \text{Spec} \arrow[loop below, "\overline{pub}"] \\
(\nu\, coin, coffee)(CM \parallel CS_1) \arrow[d, "\tau"] & & \\
(\nu\, coin, coffee)(CM_1 \parallel CS_2) \arrow[d, "\tau"] & & \\
(\nu\, coin, coffee)(CM \parallel CS) \arrow[uu, bend right, "\overline{pub}"'] & &
\end{tikzcd}
$$

Problems with Internal Actions

Question

Does $a.\tau.Nil \sim a.Nil$ hold? NO!

Problem

Strong bisimilarity does not abstract away from τ actions.

Example: SmUni $\not\sim$ Spec

$$
\begin{tikzcd}
\mathrm{SmUni} \arrow[d, "\overline{pub}"] & \not\sim & \mathrm{Spec} \arrow[loop below, "\overline{pub}"] \\
(\nu\, coin, coffee)(CM \parallel CS_1) \arrow[d, "\tau"] & & \\
(\nu\, coin, coffee)(CM_1 \parallel CS_2) \arrow[d, "\tau"] & & \\
(\nu\, coin, coffee)(CM \parallel CS) \arrow[uu, bend right=60, "\overline{pub}"'] & &
\end{tikzcd}
$$

Problems with Internal Actions

Question

Does $a.\tau.Nil \sim a.Nil$ hold? NO!

Problem

Strong bisimilarity does not abstract away from τ actions.

Example: SmUni $\not\sim$ Spec

$$\begin{tikzcd}
\mathsf{SmUni} \arrow[d, "\overline{pub}"] & \not\sim & \mathsf{Spec} \arrow[loop below, "\overline{pub}"] \\
(\nu\, coin, coffee)(CM \parallel CS_1) \arrow[d, "\tau"] & & \\
(\nu\, coin, coffee)(CM_1 \parallel CS_2) \arrow[d, "\tau"] & & \\
(\nu\, coin, coffee)(CM \parallel CS) \arrow[uu, bend right, "\overline{pub}"'] & &
\end{tikzcd}$$

Weak Transition Relation

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

Definition of Weak Transition Relation

Below, $\circ$ stands for function composition.

$$\overset{a}{\Longrightarrow} = \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^* & \text{if } a \neq \tau \\ (\xrightarrow{\tau})^* & \text{if } a = \tau \end{cases}$$

What does $s \overset{a}{\Longrightarrow} t$ informally mean?

- If $a \neq \tau$ then $s \overset{a}{\Longrightarrow} t$ means that
  from s we can get to t by doing zero or more τ actions,
  followed by the action a, followed by zero or more τ actions.
- If $a = \tau$ then $s \overset{\tau}{\Longrightarrow} t$ means that
  from s we can get to t by doing zero or more τ actions.

Weak Transition Relation

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

Definition of Weak Transition Relation

Below, $\circ$ stands for function composition.

$$\stackrel{a}{\Longrightarrow} \; = \; \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^* & \text{if } a \neq \tau \\ (\xrightarrow{\tau})^* & \text{if } a = \tau \end{cases}$$

What does $s \stackrel{a}{\Longrightarrow} t$ informally mean?

- If $a \neq \tau$ then $s \stackrel{a}{\Longrightarrow} t$ means that
  from s we can get to t by doing zero or more τ actions, followed by the action a, followed by zero or more τ actions.
- If $a = \tau$ then $s \stackrel{\tau}{\Longrightarrow} t$ means that
  from s we can get to t by doing zero or more τ actions.

Weak Bisimilarity

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

Weak Bisimulation

A binary relation $\mathcal{R} \subseteq Proc \times Proc$ is a weak bisimulation iff whenever $(s, t) \in \mathcal{R}$ then for each $a \in Act$ (including τ):

- if $s \xrightarrow{a} s'$ then $t \stackrel{a}{\Longrightarrow} t'$ for some t' such that $(s', t') \in \mathcal{R}$
- if $t \xrightarrow{a} t'$ then $s \stackrel{a}{\Longrightarrow} s'$ for some s' such that $(s', t') \in \mathcal{R}$.

Weak Bisimilarity

Two processes $p_1, p_2 \in Proc$ are weakly bisimilar ($p_1 \approx p_2$) if and only if there exists a weak bisimulation $\mathcal{R}$ such that $(p_1, p_2) \in \mathcal{R}$.

$$\approx \; = \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a weak bisimulation}\}$$

Weak Bisimilarity

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

Weak Bisimulation

A binary relation $\mathcal{R} \subseteq Proc \times Proc$ is a weak bisimulation iff whenever $(s, t) \in \mathcal{R}$ then for each $a \in Act$ (including τ):

- if $s \xrightarrow{a} s'$ then $t \overset{a}{\Longrightarrow} t'$ for some t' such that $(s', t') \in \mathcal{R}$
- if $t \xrightarrow{a} t'$ then $s \overset{a}{\Longrightarrow} s'$ for some s' such that $(s', t') \in \mathcal{R}$.

Weak Bisimilarity

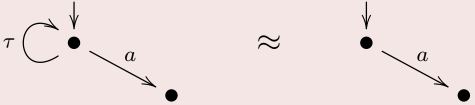
Two processes $p_1, p_2 \in Proc$ are weakly bisimilar ($p_1 \approx p_2$) if and only if there exists a weak bisimulation $\mathcal{R}$ such that $(p_1, p_2) \in \mathcal{R}$.

$$\approx \; = \; \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a weak bisimulation}\}$$

Weak Bisimilarity – Properties

Properties of $\approx$

- an equivalence relation
- the largest weak bisimulation
- validates lots of natural laws, e.g.
  - $a.\tau.P \approx a.P$
  - $P + \tau.P \approx \tau.P$
  - $a.(P + \tau.Q) \approx a.(P + \tau.Q) + a.Q$
  - $P + Q \approx Q + P \qquad P \parallel Q \approx Q \parallel P \qquad P + Nil \approx P \qquad \ldots$
- strong bisimilarity is included in weak bisimilarity ($\sim \subseteq \approx$)
- abstracts from τ loops



Is Weak Bisimilarity a Congruence?

Theorem

Let P and Q be CCS processes such that $P \approx Q$. Then

- $\alpha.P \approx \alpha.Q$ *for each action* $\alpha \in Act$
- $P \mid R \approx Q \mid R$ *and* $R \mid P \approx R \mid Q$ *for each CCS process* R
- $(\nu a)\, P \approx (\nu a)Q$ *for each set of labels* L.

What about choice?

$\tau.a.Nil \approx a.Nil$ but $\tau.a.Nil + b.Nil \not\approx a.Nil + b.Nil$

Conclusion

Weak bisimilarity is not a congruence for CCS.

Is Weak Bisimilarity a Congruence?

Theorem

Let P and Q be CCS processes such that $P \approx Q$. Then

- $\alpha.P \approx \alpha.Q$ *for each action* $\alpha \in Act$
- $P \mid R \approx Q \mid R$ *and* $R \mid P \approx R \mid Q$ *for each CCS process* R
- $(\nu a)\, P \approx (\nu a)Q$ *for each set of labels* L.

What about choice?

$\tau.a.Nil \approx a.Nil$ but $\tau.a.Nil + b.Nil \not\approx a.Nil + b.Nil$

Conclusion

Weak bisimilarity is not a congruence for CCS.

Is Weak Bisimilarity a Congruence?

Theorem

Let P and Q be CCS processes such that $P \approx Q$. Then

- $\alpha.P \approx \alpha.Q$ *for each action* $\alpha \in Act$
- $P \mid R \approx Q \mid R$ *and* $R \mid P \approx R \mid Q$ *for each CCS process* R
- $(\nu a)\, P \approx (\nu a) Q$ *for each set of labels* L.

What about choice?

$\tau.a.Nil \approx a.Nil$ but $\tau.a.Nil + b.Nil \not\approx a.Nil + b.Nil$

Conclusion

Weak bisimilarity is not a congruence for CCS.

Outline

A Calculus of Mobile Processes

Arguably, the π-calculus is the paradigmatic concurrent calculus

- Proposed by Milner, Parrow, and Walker in 1992.
  Developed significantly by Sangiorgi.

Interactive systems with dynamic connectivity (topology).
A dual role:

- A model of networked computation:
  Exchanged messages which contain links referring to communication channels themselves
- A basic model of computation:
  Interaction as the primitive notion of concurrent computing
  (Just as the λ-calculus for functional computing)

The π-calculus, in This Talk

- The theory of the π-calculus is richer than that of CCS. In some aspects, however, it is also more involved.
- We will overview this theory, contrasting it with CCS
- Hence, we present the π-calculus without going too much into technical details

Mobility as dynamic connectivity (1)

Towards the meaning of 'mobility':

- What kind of entity moves? In what space does it move?

Many possibilities—the two most relevant are:

1. Processes move, in the virtual space of linked processes
2. Links move, in the virtual space of linked processes

Observe that

- A process' location is given by the links it has to other processes (think of your contacts in your mobile phone)
- Hence, the movement of a process can be represented by the movement of its links

Mobility as dynamic connectivity (1)

Towards the meaning of 'mobility':

- What kind of entity moves? In what space does it move?

Many possibilities—the two most relevant are:

1. Processes move, in the virtual space of linked processes
2. Links move, in the virtual space of linked processes

Observe that

- A process' location is given by the links it has to other processes (think of your contacts in your mobile phone)
- Hence, the movement of a process can be represented by the movement of its links

Mobility as dynamic connectivity (2)

1. Processes move, in the virtual space of linked processes
2. Links move, in the virtual space of linked processes

The π-calculus commits to mobility in the sense of (2)...

- Economy, flexibility, and simplicity (at least wrt CCS)

...while models of higher-order concurrency stick to (1):

- Inspired in the λ-calculus
- It might be difficult/inconvenient to "normalize" all concurrency phenomena in the sense of (2)

We will argue that (1) and (2) need not be mutually exclusive

Mobility as dynamic connectivity (2)

1. Processes move, in the virtual space of linked processes
2. Links move, in the virtual space of linked processes

The π-calculus commits to mobility in the sense of (2)...

- Economy, flexibility, and simplicity (at least wrt CCS)

...while models of higher-order concurrency stick to (1):

- Inspired in the λ-calculus
- It might be difficult/inconvenient to "normalize" all concurrency phenomena in the sense of (2)

We will argue that (1) and (2) need not be mutually exclusive

Mobility as dynamic connectivity (2)

1. Processes move, in the virtual space of linked processes
2. Links move, in the virtual space of linked processes

The π-calculus commits to mobility in the sense of (2)...

- Economy, flexibility, and simplicity (at least wrt CCS)

...while models of higher-order concurrency stick to (1):

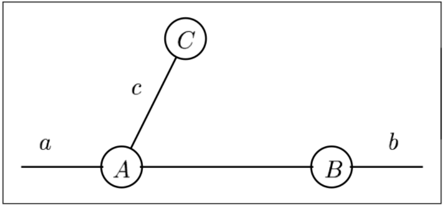
- Inspired in the λ-calculus
- It might be difficult/inconvenient to "normalize" all concurrency phenomena in the sense of (2)

We will argue that (1) and (2) need not be mutually exclusive

Dynamic connectivity in CCS is limited (1)

What's the main difference of the π-calculus wrt CCS?
Dynamic connectivity.

Suppose a CCS process $S \stackrel{\text{def}}{=} (\nu c)(A \parallel C) \parallel B$.
Name a is free in A, while b is free in B. Graphically:

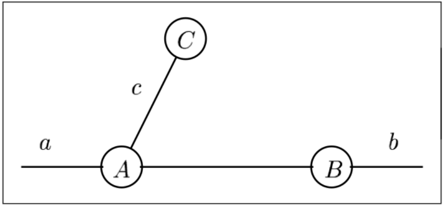


(1)

Dynamic connectivity in CCS is limited (1)

What's the main difference of the π-calculus wrt CCS?
Dynamic connectivity.

Suppose a CCS process $S \stackrel{\text{def}}{=} (\nu c)(A \parallel C) \parallel B$.
Name a is free in A, while b is free in B. Graphically:

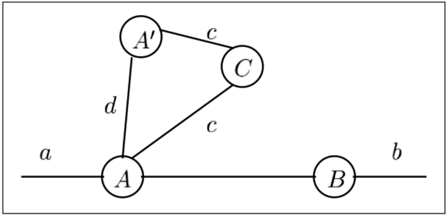


(1)

Dynamic connectivity in CCS is limited (2)

Suppose a CCS process $S \stackrel{\text{def}}{=} (\nu c)(A \parallel C) \parallel B$.
Name a is free in A, while b is free in B.

Suppose now that $A \stackrel{\text{def}}{=} a.(\nu d)(A \parallel A') + c.A''$. Graphically:



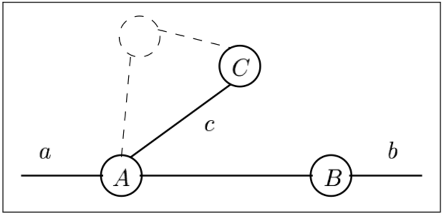
(2)

Dynamic connectivity in CCS is limited (3)

Suppose a CCS process $S \stackrel{\text{def}}{=} (\nu x)(A \parallel C) \parallel B$.
Name a is free in A, while b is free in B.

Suppose now that $A \stackrel{\text{def}}{=} a.(\nu d)(A \parallel A') + c.A''$.

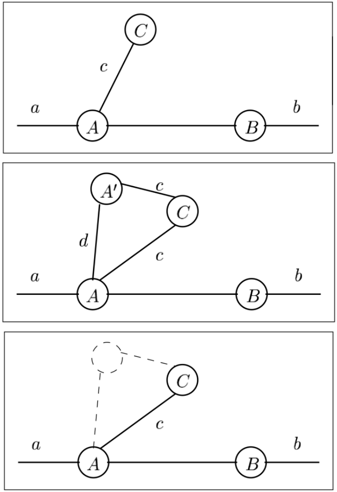
Finally, suppose that $A' = c.\mathbf{0}$. Process A' then dies. Graphically:



(3)

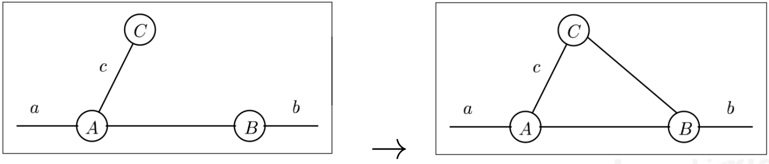
Dynamic connectivity in CCS is limited (4)

In CCS, links can proliferate and die:



Dynamic connectivity in CCS is limited (5)

However, new links between existing processes cannot be created. A transition such as

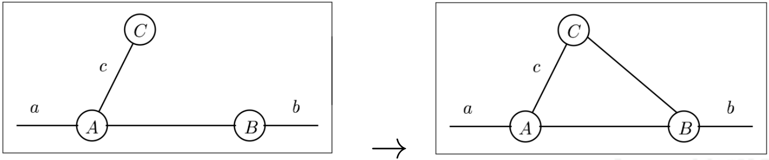


is not possible in CCS.

Dynamic connectivity refers precisely to this kind of transitions. The π-calculus goes beyond CCS by allowing dynamic communication topologies.

Dynamic connectivity in CCS is limited (5)

However, new links between existing processes cannot be created.
A transition such as



is not possible in CCS.

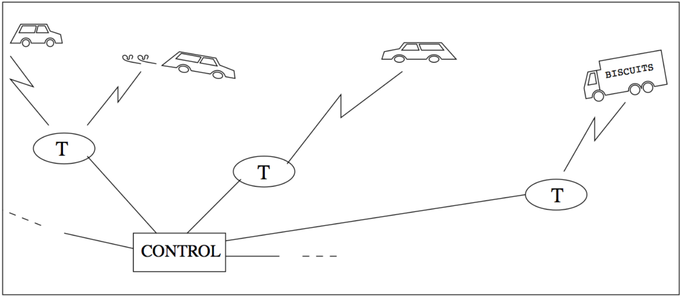
Dynamic connectivity refers precisely to this kind of transitions.
The π-calculus goes beyond CCS by allowing dynamic communication topologies.

Mobile phones and cars (1)

A simple (yet probably outdated) application of mobility.

- Vehicles on the move; each connected to a transmitter T
- Transmitters have fixed connections to a central control
- A vehicle can be switched to another transmitter
- Virtual movement of links between cars and transmitters

Before:

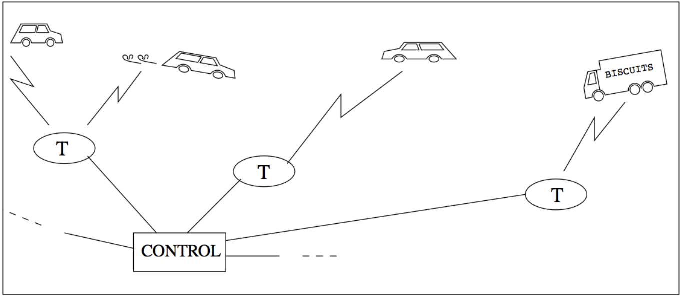


Mobile phones and cars (1)

A simple (yet probably outdated) application of mobility.

- Vehicles on the move; each connected to a transmitter T
- Transmitters have fixed connections to a central control
- A vehicle can be switched to another transmitter
- Virtual movement of links between cars and transmitters

Before:

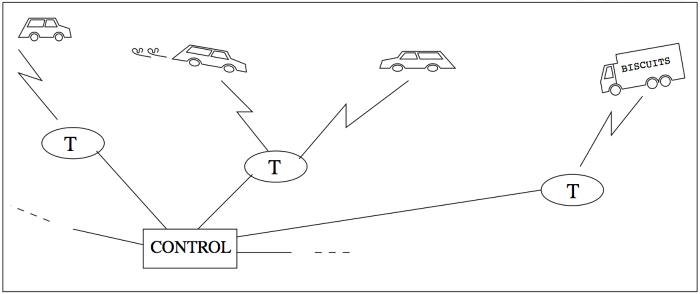


Mobile phones and cars (2)

A simple (yet probably outdated) application of mobility.

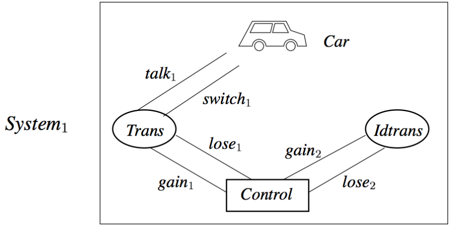
- Vehicles on the move; each connected to a transmitter T
- Transmitters have fixed connections to a central control
- A vehicle can be switched to another transmitter
- Virtual movement of links between cars and transmitters

After:

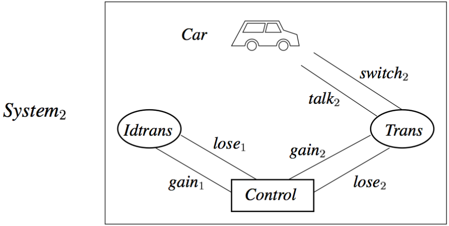


Mobile phones and cars (3)

The handover protocol in the π-calculus, schematically:

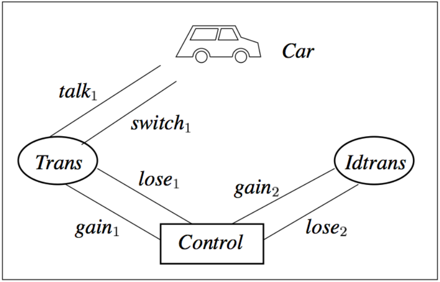


$\longrightarrow$



Mobile phones and cars (4)

System$_1$

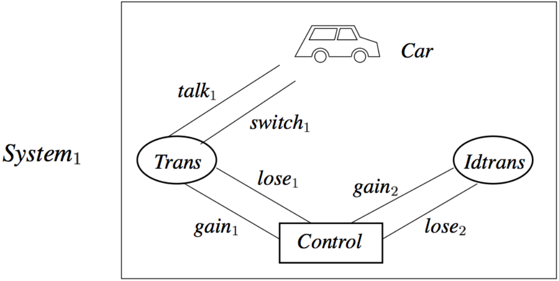


Main novelty: Communications may transmit names as messages

$$Trans\langle t,s,g,l\rangle \stackrel{\text{def}}{=} t.Trans\langle t,s,g,l\rangle \ + \ l(t,s).\overline{s}\langle t,s\rangle.Idtrans\langle g,l\rangle$$

$$Idtrans\langle g,l\rangle \stackrel{\text{def}}{=} g(t,s).Trans\langle t,s,g,l\rangle$$

Mobile phones and cars (5)



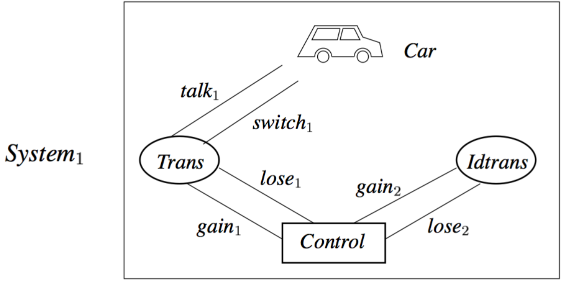
$Control$ issues a new pair of links to be communicated to Car:

$$Control_1 \stackrel{\text{def}}{=} \overline{l_1}\langle t_2, s_2\rangle.\overline{g_2}\langle t_2, s_2\rangle.Control_2$$
$$Control_2 \stackrel{\text{def}}{=} \overline{l_2}\langle t_1, s_1\rangle.\overline{g_1}\langle t_1, s_1\rangle.Control_1$$

Car can either talk or switch to another transmitter (if requested):

$$Car\langle t, s\rangle \stackrel{\text{def}}{=} \bar{t}.Car\langle t, s\rangle + s(t, s).Car\langle t, s\rangle$$

Mobile phones and cars (6)



The system is the restricted composition of the previous processes:

$$\begin{aligned} System_1 &\stackrel{\text{def}}{=} (\nu t_1, s_1, g_1, l_1, t_2, s_2, g_2, l_2) \\ &\quad (Car\langle t_1, s_1\rangle \parallel Trans_1 \parallel Idtrans_2 \parallel Control_1) \end{aligned}$$

where we have use the abbreviations ($i = 1, 2$)

$$Trans_i \stackrel{\text{def}}{=} Trans\langle t_i, s_i, g_i, l_i\rangle \qquad Idtrans_i \stackrel{\text{def}}{=} Idtrans\langle g_i, l_i\rangle$$

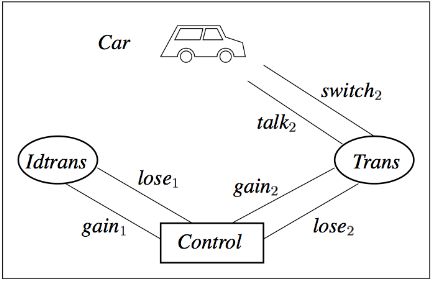
Mobile phones and cars (7)

The semantics of the π-calculus will allows to infer that $System_1$ evolves to $System_2$ where

$$System_2 \stackrel{\text{def}}{=} (\nu t_1, s_1, g_1, l_1, t_2, s_2, g_2, l_2)$$
$$(Car\langle t_2, s_2\rangle \parallel Trans_2 \parallel Idtrans_1 \parallel Control_2)$$

(The process obtained from $System_1$ by exchanging the indexes.)

System$_2$



The π-calculus, more formally

We now formally introduce the π-calculus. Some highlights:

- The major novelty is name communication
- Dynamic connectivity formalized as scope extrusion
- A structural congruence relation

The π-calculus, more formally

We use $x, y, z, \ldots$ to range over $\mathcal{N}$, an infinite set of names.

The action prefixes of the π-calculus generalize the actions of CCS:

$$\begin{array}{lll} \alpha & ::= & \overline{x}\langle y \rangle & \text{send name } y \text{ along } x \\ & & x(y) & \text{receive a name along } x \\ & & \tau & \text{unobservable action} \end{array}$$

Brackets in $\overline{x}\langle y \rangle$ represent a tuple of values.

- Above, monadic communication: exactly one name is sent.
- In polyadic communication more than one value may be sent.

Process expressions of the π-calculus

P, Q	$::=$	$\mathbf{0}$	Inactive process
	$\mid$	$\alpha.P$	Prefix
	$\mid$	$P + P$	Sum
	$\mid$	$P \parallel Q$	Parallel composition
	$\mid$	$(\nu y)P$	Name Restriction
	$\mid$	$A\langle y_1, \ldots, y_n\rangle$	Identifier

We assume each identifier A is equipped with a recursive definition $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$, where $i \neq j$ implies $x_i \neq x_j$.

- Restriction and input actions are name binders:
  In $(\nu y)P$ and $x(y).P$ name y is bound with scope P.
- In contrast, in $\overline{x}\langle y\rangle$ name y is free.

Structural Congruence

A few intuitions:

- The syntax of processes is too concrete: syntactically different things that represent the same behavior. Examples:

$$a(x).\bar{b}\langle x\rangle \quad \text{and} \quad a(y).\bar{b}\langle y\rangle$$
$$P \parallel Q \quad \text{and} \quad Q \parallel P$$

  [We often omit trailing $\mathbf{0}$s, and write $\bar{b}\langle y\rangle$ instead of $\bar{b}\langle y\rangle.\mathbf{0}$.]

- Structural congruence identifies processes which are "obviously the same" based on their structure
- In this sense, structural congruence will be stronger (that is, will equate less process) than any behavioral equivalence.

Structural Congruence, $\equiv$

P and Q structurally congruent, written $P \equiv Q$, if we can transform one into the other by using the following equations:

1. α-conversion: change of bound names
2. Laws for parallel composition:

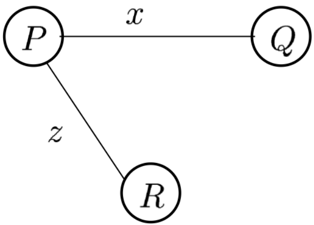
$$\begin{array}{rcl} P \parallel \mathbf{0} & \equiv & P \\ P \parallel Q & \equiv & Q \parallel P \\ P \parallel (Q \parallel R) & \equiv & (P \parallel Q) \parallel R \end{array}$$

3. Law for recursive definitions: $A\langle\tilde{y}\rangle \equiv P\{\tilde{y}/\tilde{x}\}$ if $A(\tilde{x}) \stackrel{\text{def}}{=} P$
4. Laws for restriction:

$$\begin{array}{rcll} (\nu x)(P \parallel Q) & \equiv & P \parallel (\nu x)Q & \text{if } x \notin \mathsf{fn}(P) \\ (\nu x)\mathbf{0} & \equiv & \mathbf{0} & \\ (\nu x)(P + Q) & \equiv & P + (\nu x)Q & \text{if } x \notin \mathsf{fn}(P) \\ (\nu x)(\nu y)\, P & \equiv & (\nu y)(\nu x)\, P & \end{array}$$

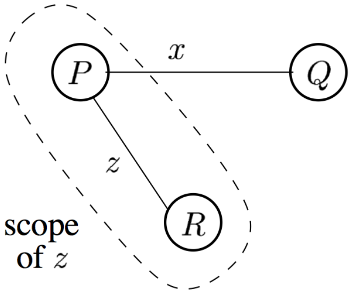
Scope Extrusion (1)

A process $P \parallel Q \parallel R$.
Name x is free in P and Q, while z is free in Q and R:



Scope Extrusion (2)

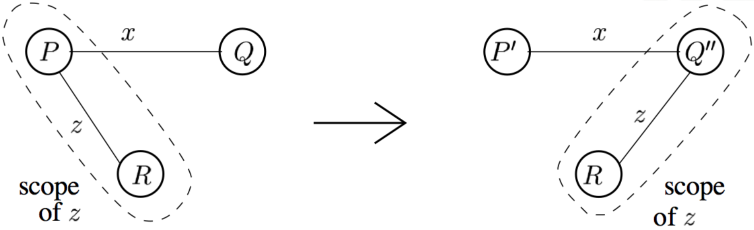
Suppose that z is restricted to P and R, while x is free in P and Q. That is, we have the process $(\nu z)(P \parallel R) \parallel Q$:



What happens if P wishes to send z to Q?

Scope Extrusion (3)

Suppose $P = \overline{x}\langle z\rangle.P'$, with $z \notin \mathsf{fn}(P')$.
Suppose also $Q = x(y).Q'$, with $z \notin \mathsf{fn}(Q')$.



where $Q'' = Q'\{z/y\}$. We have graphically described the reduction

$$(\nu z)(P \parallel R) \parallel Q \longrightarrow P' \parallel (\nu z)(R \parallel Q'')$$

The above describes a movement of a way of accessing R (rather than a movement of R).

Some Simple Examples

We present some simple examples of scope extrusion.
We exploit three (informal) postulates for this:

1. A law for inferring interactions:

$$a(x).P \parallel \overline{a}\langle b\rangle.Q \xrightarrow{\tau} P\{b/x\} \parallel Q$$

2. Restrictions respect silent transitions:

$$P \xrightarrow{\tau} Q \text{ implies } (\nu x)P \xrightarrow{\tau} (\nu x)Q$$

3. Structurally congruent processes should have the same behavior.

A Simple Example

We use str. congruence to infer an interaction for the process

$$a(x).\overline{c}\langle x\rangle \parallel (\nu b)\overline{a}\langle b\rangle$$

Since $b \notin \mathsf{fn}(a(x).\overline{c}\langle x\rangle)$, we have

$$a(x).\overline{c}\langle x\rangle \parallel (\nu b)\overline{a}\langle b\rangle \equiv (\nu b)(a(x).\overline{c}\langle x\rangle \parallel \overline{a}\langle b\rangle)$$

We can infer that

$$(\nu b)(a(x).\overline{c}\langle x\rangle \parallel \overline{a}\langle b\rangle) \xrightarrow{\tau} (\nu b)(\overline{c}\langle b\rangle \parallel \mathbf{0})$$

because $a(x).\overline{c}\langle x\rangle \parallel \overline{a}\langle b\rangle \xrightarrow{\tau} \overline{c}\langle b\rangle \parallel \mathbf{0}$ is a valid interaction.

Removing $\mathbf{0}$, in general we have, for any $b \notin fn(P)$:

$$a(x).P \parallel (\nu b)\overline{a}\langle b\rangle.Q \xrightarrow{\tau} (\nu b)(P \parallel Q\{b/x\})$$

and the scope of b has moved from the right to the left.

A Simple Example

We use str. congruence to infer an interaction for the process

$$a(x).\overline{c}\langle x\rangle \parallel (\nu b)\overline{a}\langle b\rangle$$

Since $b \notin \mathsf{fn}(a(x).\overline{c}\langle x\rangle)$, we have

$$a(x).\overline{c}\langle x\rangle \parallel (\nu b)\overline{a}\langle b\rangle \equiv (\nu b)(a(x).\overline{c}\langle x\rangle \parallel \overline{a}\langle b\rangle)$$

We can infer that

$$(\nu b)(a(x).\overline{c}\langle x\rangle \parallel \overline{a}\langle b\rangle) \xrightarrow{\tau} (\nu b)(\overline{c}\langle b\rangle \parallel \mathbf{0})$$

because $a(x).\overline{c}\langle x\rangle \parallel \overline{a}\langle b\rangle \xrightarrow{\tau} \overline{c}\langle b\rangle \parallel \mathbf{0}$ is a valid interaction.

Removing $\mathbf{0}$, in general we have, for any $b \notin fn(P)$:

$$a(x).P \parallel (\nu b)\overline{a}\langle b\rangle.Q \xrightarrow{\tau} (\nu b)(P \parallel Q\{b/x\})$$

and the scope of b has moved from the right to the left.

A Simple Example

We use str. congruence to infer an interaction for the process

$$a(x).\overline{c}\langle x\rangle \parallel (\nu b)\overline{a}\langle b\rangle$$

Since $b \notin \mathsf{fn}(a(x).\overline{c}\langle x\rangle)$, we have

$$a(x).\overline{c}\langle x\rangle \parallel (\nu b)\overline{a}\langle b\rangle \equiv (\nu b)(a(x).\overline{c}\langle x\rangle \parallel \overline{a}\langle b\rangle)$$

We can infer that

$$(\nu b)(a(x).\overline{c}\langle x\rangle \parallel \overline{a}\langle b\rangle) \xrightarrow{\tau} (\nu b)(\overline{c}\langle b\rangle \parallel \mathbf{0})$$

because $a(x).\overline{c}\langle x\rangle \parallel \overline{a}\langle b\rangle \xrightarrow{\tau} \overline{c}\langle b\rangle \parallel \mathbf{0}$ is a valid interaction.

Removing $\mathbf{0}$, in general we have, for any $b \notin fn(P)$:

$$a(x).P \parallel (\nu b)\overline{a}\langle b\rangle.Q \xrightarrow{\tau} (\nu b)(P \parallel Q\{b/x\})$$

and the scope of b has moved from the right to the left.

Another Example

Consider the process:

$$P = (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel x(u).\overline{u}\langle v\rangle \parallel \overline{x}\langle z\rangle)$$

Observe: $\mathsf{fn}(P) = \{x, v, y\}$, $\mathsf{bn}(P) = \{z, w, u\}$. Two possibilities:

1. Interaction among the first and second components:

$$\begin{array}{rcl} P & \xrightarrow{\tau} & (\nu z)(\mathbf{0} \parallel \overline{u}\langle v\rangle\{y/u\} \parallel \overline{x}\langle z\rangle) \\ & = & (\nu z)(\mathbf{0} \parallel \overline{y}\langle v\rangle \parallel \overline{x}\langle z\rangle) = P_1 \end{array}$$

   $P\{y/u\}$ is the process P in which the free occurrences of name u have been substituted with y.

2. Interaction among the second and third components:

$$\begin{array}{rcl} P & \xrightarrow{\tau} & (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel \overline{u}\langle v\rangle\{z/u\} \parallel \mathbf{0}) \\ & = & (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel \overline{z}\langle v\rangle \parallel \mathbf{0}) = P_2 \end{array}$$

While $P_1 \not\xrightarrow{\tau}$, we do have $P_2 \xrightarrow{\tau} (\nu z)(\overline{z}\langle y\rangle \parallel \mathbf{0} \parallel \mathbf{0}) \equiv (\nu z)\overline{z}\langle y\rangle$

Another Example

Consider the process:

$$P = (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel x(u).\overline{u}\langle v\rangle \parallel \overline{x}\langle z\rangle)$$

Observe: $\mathsf{fn}(P) = \{x, v, y\}$, $\mathsf{bn}(P) = \{z, w, u\}$. Two possibilities:

1. Interaction among the first and second components:

$$\begin{aligned} P &\xrightarrow{\tau} (\nu z)(\mathbf{0} \parallel \overline{u}\langle v\rangle\{y/u\} \parallel \overline{x}\langle z\rangle) \\ &= (\nu z)(\mathbf{0} \parallel \overline{y}\langle v\rangle \parallel \overline{x}\langle z\rangle) = P_1 \end{aligned}$$

   $P\{y/u\}$ is the process P in which the free occurrences of name u have been substituted with y.

2. Interaction among the second and third components:

$$\begin{aligned} P &\xrightarrow{\tau} (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel \overline{u}\langle v\rangle\{z/u\} \parallel \mathbf{0}) \\ &= (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel \overline{z}\langle v\rangle \parallel \mathbf{0}) = P_2 \end{aligned}$$

While $P_1 \not\xrightarrow{\tau}$, we do have $P_2 \xrightarrow{\tau} (\nu z)(\overline{z}\langle y\rangle \parallel \mathbf{0} \parallel \mathbf{0}) \equiv (\nu z)\overline{z}\langle y\rangle$

Another Example

Consider the process:

$$P = (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel x(u).\overline{u}\langle v\rangle \parallel \overline{x}\langle z\rangle)$$

Observe: fn$(P) = \{x, v, y\}$, bn$(P) = \{z, w, u\}$. Two possibilities:

1. Interaction among the first and second components:

$$\begin{aligned} P &\xrightarrow{\tau} (\nu z)(\mathbf{0} \parallel \overline{u}\langle v\rangle\{y/u\} \parallel \overline{x}\langle z\rangle) \\ &= (\nu z)(\mathbf{0} \parallel \overline{y}\langle v\rangle \parallel \overline{x}\langle z\rangle) = P_1 \end{aligned}$$

   $P\{y/u\}$ is the process P in which the free occurrences of name u have been substituted with y.

2. Interaction among the second and third components:

$$\begin{aligned} P &\xrightarrow{\tau} (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel \overline{u}\langle v\rangle\{z/u\} \parallel \mathbf{0}) \\ &= (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel \overline{z}\langle v\rangle \parallel \mathbf{0}) = P_2 \end{aligned}$$

While $P_1 \not\xrightarrow{\tau}$, we do have $P_2 \xrightarrow{\tau} (\nu z)(\overline{z}\langle y\rangle \parallel \mathbf{0} \parallel \mathbf{0}) \equiv (\nu z)\overline{z}\langle y\rangle$

Another Example

Consider the process:

$$P = (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel x(u).\overline{u}\langle v\rangle \parallel \overline{x}\langle z\rangle)$$

Observe: $\mathsf{fn}(P) = \{x, v, y\}$, $\mathsf{bn}(P) = \{z, w, u\}$. Two possibilities:

1. Interaction among the first and second components:

$$\begin{array}{rcl} P & \xrightarrow{\tau} & (\nu z)(\mathbf{0} \parallel \overline{u}\langle v\rangle\{y/u\} \parallel \overline{x}\langle z\rangle) \\ & = & (\nu z)(\mathbf{0} \parallel \overline{y}\langle v\rangle \parallel \overline{x}\langle z\rangle) = P_1 \end{array}$$

   $P\{y/u\}$ is the process P in which the free occurrences of name u have been substituted with y.

2. Interaction among the second and third components:

$$\begin{array}{rcl} P & \xrightarrow{\tau} & (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel \overline{u}\langle v\rangle\{z/u\} \parallel \mathbf{0}) \\ & = & (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \parallel \overline{z}\langle v\rangle \parallel \mathbf{0}) = P_2 \end{array}$$

While $P_1 \not\xrightarrow{\tau}$, we do have $P_2 \xrightarrow{\tau} (\nu z)(\overline{z}\langle y\rangle \parallel \mathbf{0} \parallel \mathbf{0}) \equiv (\nu z)\overline{z}\langle y\rangle$

Process Calculi

A Brief, Gentle Introduction

Jorge A. Pérez

university of groningen

University of Brasilia
July 20, 2015