Type-based Communication Correctness in Multi-agent Systems

Part I: Correctness, Communication, and Process Calculi

Jorge A. Pérez

Bernoulli Institute for Mathematics, Computer Science, and AI
University of Groningen, The Netherlands

www.jperez.nl

20th European Agent Systems Summer School (EASSS 2018)

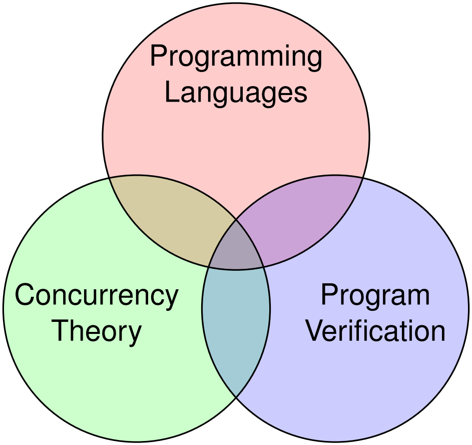
Outline

Overview of the Tutorial

An introduction to **type systems** (in particular, **session types**), a technique for enforcing correct multi-agent, communicating systems.

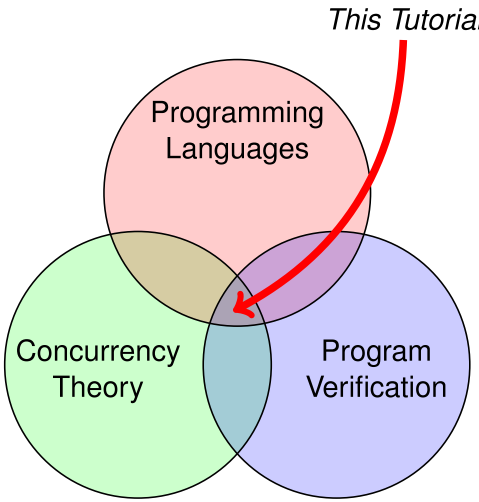
Overview of the Tutorial

An introduction to **type systems** (in particular, **session types**), a technique for enforcing correct multi-agent, communicating systems.



Overview of the Tutorial

An introduction to **type systems** (in particular, **session types**), a technique for enforcing correct multi-agent, communicating systems.



Overview of the Tutorial

An introduction to **type systems** (in particular, **session types**), a technique for enforcing correct multi-agent, communicating systems.

Part I: Correctness, Communication, and Process Calculi

- What is software correctness?
- Concurrency and communication
- Formal models of concurrency, in particular process calculi
- The π-calculus, the paradigmatic calculus of concurrency

Part II: Type Systems for Concurrency & Logical Foundations

- Type systems for concurrency: behavioral types
- Session types for the π-calculus
- Logical foundations for message-passing concurrency
- Further topics

About Me

Since April 2014, assistant professor at University of Groningen:

- Research within the “Fundamental Computing” group
  Part-time affiliation at CWI, Amsterdam (since September 2016)
- Current group: 3 PhDs (two PhDs and one PostDoc soon to arrive)
- Collaborations with UK, Italy, France, Portugal, Serbia, Colombia
- **Looking forward to new collaborations!**

About Me

Since April 2014, assistant professor at University of Groningen:

- Research within the “Fundamental Computing” group
  Part-time affiliation at CWI, Amsterdam (since September 2016)
- Current group: 3 PhDs (two PhDs and one PostDoc soon to arrive)
- Collaborations with UK, Italy, France, Portugal, Serbia, Colombia
- **Looking forward to new collaborations!**

Prior to Groningen:

- Postdoctoral Researcher (Lisbon, Portugal, 2010-2014)
  Work on: Logical Foundations of Concurrent Programming
- PhD in Informatics (Bologna, Italy, 2007-2010)
  Thesis: Expressiveness Results for Higher-Order Concurrency
- Engineering Degree in Computer Science (Cali, Colombia, 2005)
  Thesis: A Library for Soft Constraint Satisfaction Problems

Outline

Software is Everywhere!

Software Should be Correct

- A huge part of software today is deployed in critical infrastructures (Critical as in ‘millions of human lives involved’)
- We depend on programs and systems behaving as intended
- While correct software is a “silent hero”, defective/unreliable/costly software (rightly) receives a lot of attention
- Not a new issue! Academic interest at least from the late 1960s (“Software Crisis” is a term coined in 1968!)
- Increasing awareness about software correctness, and its many ramifications (financial, social, ethical, etc)

Lots of Incorrect Software

WIKIPEDIA
The Free Encyclopedia

Article | Talk

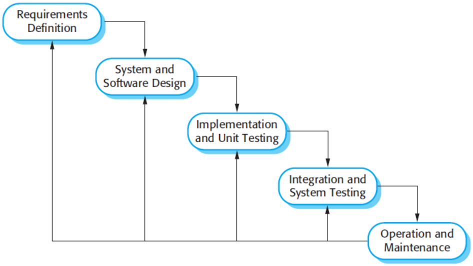
List of software bugs

From Wikipedia, the free encyclopedia

`http://en.wikipedia.org/wiki/List_of_software_bugs`

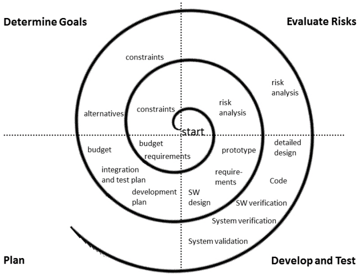
Software Correctness in the Real World

In software engineering processes, correctness means testing:



Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

The waterfall model



The spiral model

The Humble Programmer (1/2)

Edsger Wybe Dijkstra
ACM Turing Winner, 1972

- Testing can be effective to show the presence of bugs, but is hopelessly inadequate for showing their absence.

The Humble Programmer (1/2)

Edsger Wybe Dijkstra
ACM Turing Winner, 1972

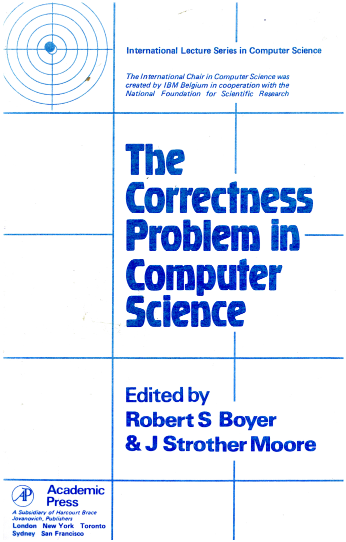
- Testing can be effective to show the presence of bugs, but is hopelessly inadequate for showing their absence.
- The only effective way to raise the confidence level of a program is to give a convincing proof of its correctness.
- But one should not first make the program and then prove its correctness: the requirement of providing the proof would only increase the programmer's burden.
- The programmer should let correctness proof and program grow hand in hand.

The Humble Programmer (2/2)

- The only mental tool by means of which a very finite piece of reasoning can cover a myriad cases is “abstraction”.
- The effective exploitation of his powers of abstraction is one of the most vital activities of a competent programmer.
- The purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise.

Edsger Wybe Dijkstra
ACM Turing Winner, 1972

Guess the Year!


International Lecture Series in Computer Science

The International Chair in Computer Science was created by IBM Belgium in cooperation with the National Foundation for Scientific Research

The Correctness Problem in Computer Science

Edited by Robert S Boyer & J Strother Moore

Academic Press

A Subsidiary of Harcourt Brace Jovanovich, Publishers

London New York Toronto Sydney San Francisco

From the preface:

- The cost of “bugs” in critical software is great; consequently the problem of software correctness is being investigated intensively worldwide.
- Programming is a mathematical activity. That is, a programmer ought to be able to prove, in the mathematical sense, that his programs are correct.
- Programming requires of programmers the precise thinking, creative leaps, and attention to detail normally required of mathematicians.

When is Software Correct?

The correctness problem consists in checking that a software system “behaves as intended”

We’d say that a software artifact (e.g., a mobile app) is correct when

- Provides an output consistent with some given input
- Correctly (and quickly) completes its tasks
- Shows predictable and reliable behavior

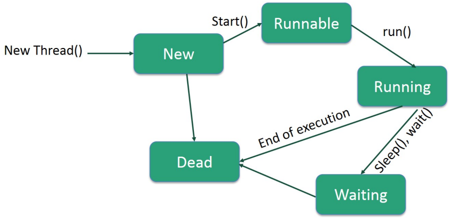
Formal Verification

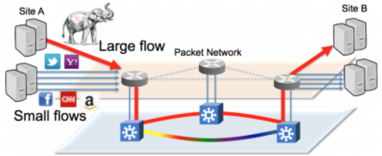
Formal verification is the problem of checking that an artifact P is correct with respect to a given mathematical specification φ.

- φ describes the system behaviors (or configurations) that the designer judges to be acceptable
- artifacts that always exhibit behaviors as described by φ are said to correctly implement φ
- φ could be a formula in some logic (model checking) or a formal specification of the expected behavior (equivalence checking)

Outline

“Things That Occur at the Same Time”





Concurrency: A Challenge

Currently, a mismatch:

Information technologies
(predominantly concurrent and interactive)
vs.
computing foundations
(mostly sequential)

Concurrency: A Challenge

Currently, a mismatch:

Information technologies
(predominantly concurrent and interactive)
vs.
computing foundations
(mostly sequential)

- Consequence: conceiving, designing, and implementing concurrent software systems is difficult and error prone.
- These errors are often costly (even catastrophic) and have societal implications

Correctness in Concurrency is Tricky

Consider two programs, P_1 and P_2:

$$P_1 : x := 1; x := x + 1$$
$$P_2 : x := 2$$

Run in isolation, P_1 and P_2 have the **same semantics**:
they replace the value of x by 2.

Correctness in Concurrency is Tricky

Consider two programs, P_1 and P_2:

$$P_1 : x := 1; x := x + 1$$
$$P_2 : x := 2$$

Run in isolation, P_1 and P_2 have the **same semantics**:
they replace the value of x by 2.

Consider now the program Q:

$$Q : x := 3$$

and run it concurrently with P_1 and P_2:

$$R_1 : P_1 \parallel Q$$
$$R_2 : P_2 \parallel Q$$

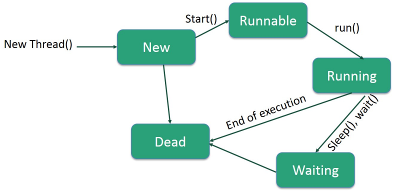
Now R_1 and R_2 have a **different semantics** (Why?)

Which Concurrency?

A rough classification of the forms of concurrency of interest in computer science:

- Shared-memory concurrency (“local concurrency”)
- Message-passing concurrency (“global concurrency”)

Shared-Memory Concurrency



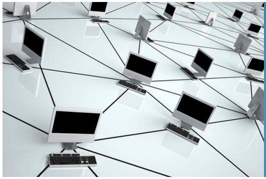
Focus on interaction that occurs in a shared state.

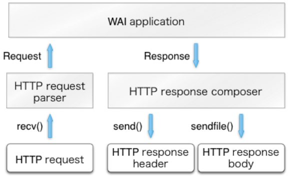
Multiple homogeneous tasks executing “nearby” with limited resources and operations.

Examples:

- Threads in a smartphone (or any mobile device)
- Multicore processors in modern tablets and laptops
- GPGPUs
- Concurrent data structures

Message-Passing Concurrency





Privileges the notion of computing as interaction, in distributed and highly dynamic scenarios.

Typical of systems built as the composition of heterogeneous components which communicate between them.

We shall refer to those components as agents or processes.

Examples:

- Online services for booking flights and hotels
- Web services, cloud computing, software-defined networks
- Government information systems

Models for Concurrency

It is hard to specify and reason about the phenomena which are typical of concurrent computation.

This is because, unlike sequential programs, concurrent systems are:

- Interactive and reactive
- Infinite (as opposed to terminating)
- Hard to predict (as opposed to deterministic)

Consequence: Models and techniques for designing, building, and verifying sequential systems are inadequate for concurrent systems.

Models for Concurrency

We require models tailored to concurrent computing.

Models for Concurrency

We require models tailored to concurrent computing.

In principle, we would like models which are at least

- general, based on a few key principles
- expressive enough to represent relevant phenomena

Models for Concurrency

We require models tailored to concurrent computing.

In principle, we would like models which are at least

- general, based on a few key principles
- expressive enough to represent relevant phenomena

Moreover, these models should be also precise and reliable.
To this end, we will find it reasonable to require models which

- are formal: based upon solid mathematical foundations
- are endowed with reasoning techniques which allow us to discern about certain **aspects of interest**

Aspects of Interest

Models of concurrency may aim at capturing different aspects. Some examples:

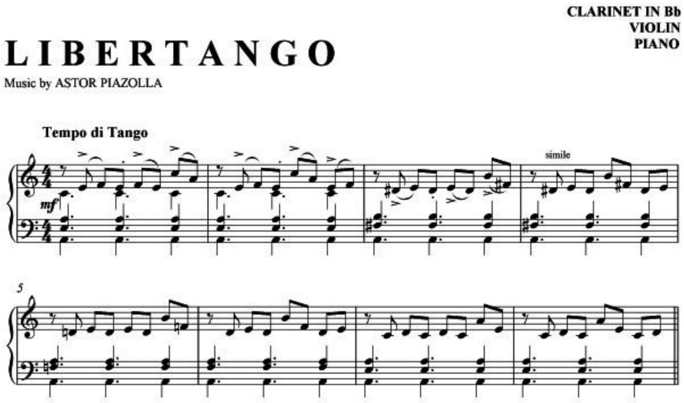
- communication discipline: point-to-point, broadcast
- synchronization mechanisms: synchronous, asynchronous
- message passing, shared variables
- timed (discrete, continuous) or untimed
- deterministic, or non deterministic

Following an abstraction principle, the intended models will focus only on a few aspects/concerns, ignoring the rest.

Models of Concurrency: Ideal

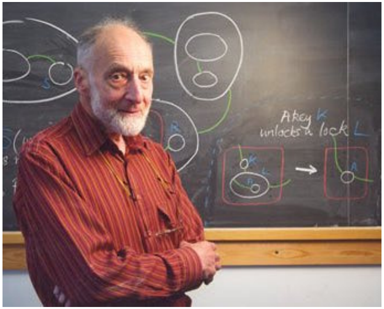
CLARINET IN Bb
VIOLIN
PIANO
LIBERTANGO
Music by ASTOR PIAZOLLA
Tempo di Tango
mf
simile
5

Models of Concurrency: Ideal



- **Formal**: abstract, rigorous, unambiguous
- **Common language**, widely understandable to various experts
- **Very expressive** (e.g. flexible time, multiparty synchronizations)

Elements of Interaction (1/2)

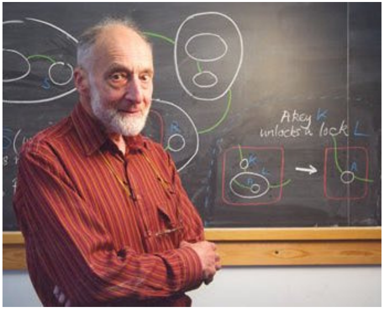


Robin Milner
ACM Turing Winner, 1991

- Concurrency requires a fresh approach, not just an extension of the constructions which explain sequential computing.
- Constructions for concurrency may be understood mathematically, just as sequential computing may be understood in terms of functions.
- I reject the idea that there can be a unique model or formalism for all aspects of something as large as concurrent computation.

Elements of Interaction (2/2)

- Functions are an essential ingredient of the air we breathe, so to speak, when we discuss sequential programming.
- But for concurrent programming and interactive systems in general, we have nothing comparable.
- Right ideas to explain concurrency will only come from a dialectic between models from logic and mathematics and a proper distillation of practical experience.



Robin Milner
ACM Turing Winner, 1991

Formal Models of Concurrency

In this tutorial, we shall focus on so-called process calculi (aka process algebras), a class of models of concurrency.

Formal Models of Concurrency

In this tutorial, we shall focus on so-called process calculi (aka process algebras), a class of models of concurrency.

- Formal languages in which the structure of terms represents (or reflects) the structure of computational processes
- Such a structure is given by a reduced set of process constructors
- Tiny programming languages, endowed with an operational semantics which represents (concurrent) computational steps.
- Widely studied since the 80s (promoted by Milner and others).

Example:

- Given processes P and Q, process $P \parallel Q$ is the parallel execution of P and Q. That is, P and Q are combined using the $\parallel$ operator.
- If P autonomously evolves into P', an operational semantics $\longrightarrow$ may decree that $P \longrightarrow P'$ but also that $P \parallel Q \longrightarrow P' \parallel Q$.

Process Calculi: Some Features

- A compositional approach: a concurrent system specified in terms of its sub-systems and the interaction between them
- The operators allow us to represent explicitly abstraction criteria in specifications
- They are defined as minimal models, able to represent interesting behaviors using a reduced set of elements.

Process Calculi: Some Features

- A compositional approach: a concurrent system specified in terms of its sub-systems and the interaction between them
- The operators allow us to represent explicitly abstraction criteria in specifications
- They are defined as minimal models, able to represent interesting behaviors using a reduced set of elements.

Typically, process calculi are able to represent

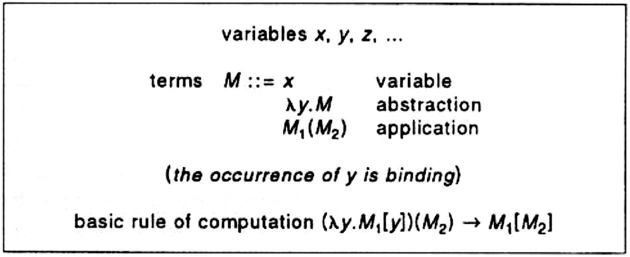
- Atomic actions (inputs, outputs) and their interaction (synchronizations)
- Explicit concurrency (e.g. the parallel composition operator $\|$)
- Choices between different alternative behaviors
- Delimited interactions (e.g., channels private to a process)
- Infinite behaviors (e.g. recursion)

Outline

Process Calculi: Purpose

- Basic models of concurrent computing (just as the λ-calculus and automata are the foundation of sequential programming)
- Formal foundations for modern programming languages and development tools
- Useful to define and study reasoning and verification techniques:
  - Simulators
  - Model and equivalence checkers
  - **Type systems**
- Two notable process calculi (by Milner and collaborators):
  - CCS: The Calculus of Communicating Systems
  - **The π-calculus**

Sequential (λ) vs Concurrent (π)

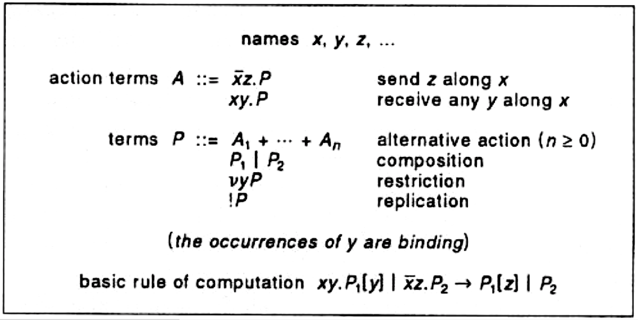


variables $x, y, z, \ldots$

terms	$M ::=$	x	variable
		$\lambda y.M$	abstraction
		$M_1(M_2)$	application

(*the occurrence of y is binding*)

basic rule of computation $(\lambda y.M_1[y])(M_2) \rightarrow M_1[M_2]$



names $x, y, z, \ldots$

action terms	$A ::=$	$\bar{x}z.P$	send z along x
		$xy.P$	receive any y along x
terms	$P ::=$	$A_1 + \cdots + A_n$	alternative action ($n \geq 0$)
		$P_1 \mid P_2$	composition
		$\nu y P$	restriction
		$!P$	replication

(*the occurrences of y are binding*)

basic rule of computation $xy.P_1[y] \mid \bar{x}z.P_2 \rightarrow P_1[z] \mid P_2$

Outline

The π-calculus: A Calculus of Mobile Processes

Arguably, the paradigmatic calculus for concurrency

- 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)

Here: 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?

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 in this course 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

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

...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)

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

...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, Intuitively (1)

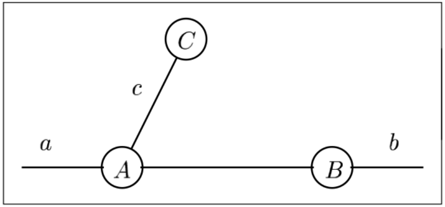
Arguably the most distinctive feature of the π-calculus as a model of concurrency is **dynamic connectivity.**

Dynamic connectivity, Intuitively (1)

Arguably the most distinctive feature of the π-calculus as a model of concurrency is **dynamic connectivity**.

To motivate this idea, let's assume a graph-like notation for communicating agents. Agents, depicted as nodes, will be connected if they share a name for communication.

Let A, B, and C be agents. A and C share a name c.
A and B have names a and b, respectively, to communicate with their environment. C and B don't share any names.

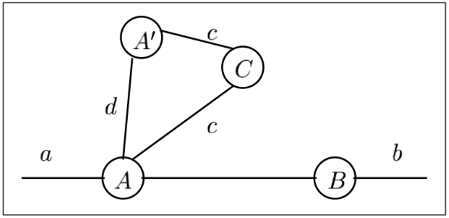


(1)

Dynamic connectivity, Intuitively (2)

Suppose now that A can evolve and split into two agents, A and A', which use a private name d to communicate. We may think of A' as a "deputy" agent for A.

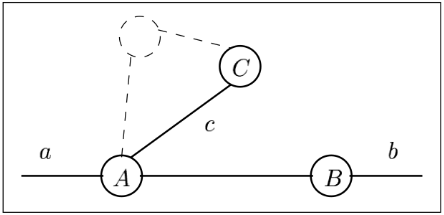
Then the system looks as follows:



(2)

Dynamic connectivity, Intuitively (3)

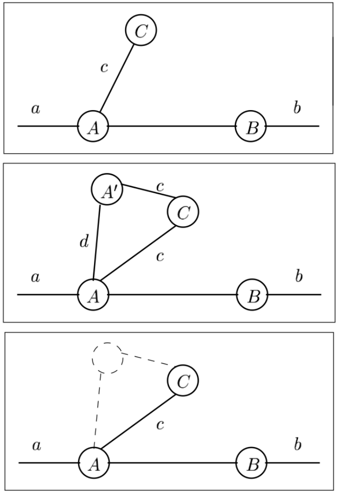
Finally, suppose that A' performs some (communication) actions to stand by for A and then dies. Graphically:



(3)

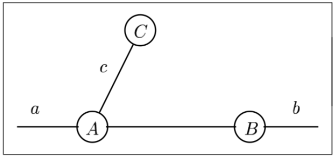
Dynamic connectivity, Intuitively ((4)

The models of concurrency that preceded the π-calculus (such as CCS) links between agents can proliferate and die:

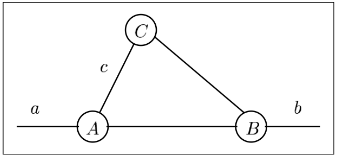


Dynamic connectivity, Intuitively (5)

However, such models are limited: new links between existing agents cannot be created. In those models, a transition such as



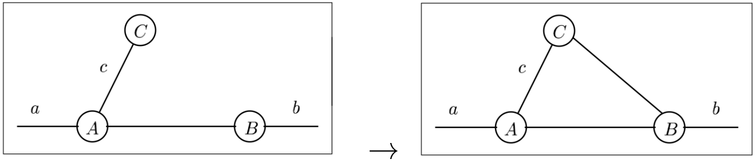
$\rightarrow$



is not possible.

Dynamic connectivity, Intuitively (5)

However, such models are limited: new links between existing agents cannot be created. In those models, a transition such as



is not possible.

Dynamic connectivity refers precisely to this kind of transitions.

The π-calculus goes beyond CCS by allowing dynamic communication topologies.

The π-calculus, more formally

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

- The major novelty is communication of names
- Dynamic connectivity formalized as scope extrusion
- An operational semantics coupled with a relation of structural congruence

The π-calculus: Syntax

- We use $x, y, z, \ldots$ to range over $\mathcal{N}$, an infinite set of names.
- Communication actions are specified via action prefixes:

α	$::=$	$\overline{x}\langle y\rangle$	send name y along x
		$x(y)$	receive a name along x
		τ	unobservable action

- Syntax of processes:

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

  Each A is equipped with a definition $A(x_1, \ldots, x_n) \overset{\text{def}}{=} P$.
- In $(\nu y)\, P$ and $x(y).P$ name y is bound with scope P.

Structural Congruence: Intuitions

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

$$
\begin{array}{ccccc}
a(x).\overline{b}\langle x\rangle & \text{and} & a(y).\overline{b}\langle y\rangle & & \text{(“Receive on } a\text{, forward on } b\text{”)} \\
P \parallel Q & \text{and} & Q \parallel P & & \text{(“Run } P \text{ and } Q \text{ concurrently”)}
\end{array}
$$

  [We often omit trailing $\mathbf{0}$s, and write $\overline{b}\langle y\rangle$ instead of $\overline{b}\langle y\rangle.\mathbf{0}$.]
- Structural congruence identifies processes which are “clearly the same” based on their syntactical structure

Structural Congruence

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:

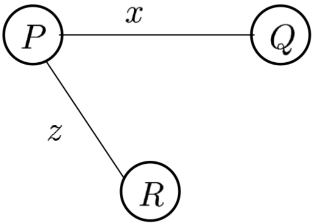
$$\begin{array}{rcl} P \parallel \mathbf{0} & \equiv & P \\ P \parallel Q & \equiv & Q \parallel P \\ P \parallel (Q \parallel R) & \equiv & (P \parallel Q) \parallel R \end{array}$$

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

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

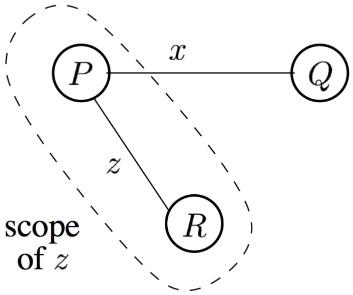
Scope Extrusion (1)

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)

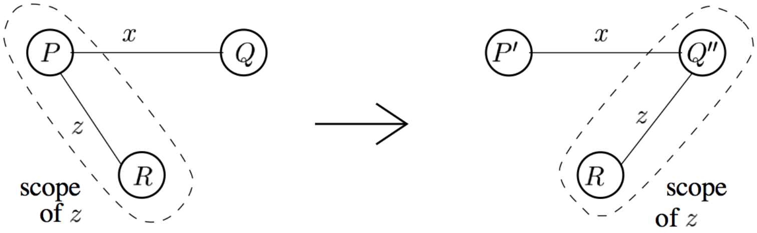
Suppose that z is restricted to P and R, while x is free in P and Q. That is, we have the process $(\nu z)(P \parallel R) \parallel Q$:



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 an operational semantics, which we will define as a reduction relation $\longmapsto$:

1. A law for inferring interactions (point-to-point communication):

$$a(x).P \parallel \overline{a}\langle b\rangle.Q \longmapsto P\{{}^{b}\!/\!{}_{x}\} \parallel Q$$

2. Restrictions respect silent transitions:

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

3. Structurally congruent processes have the same behavior

A Simple Example

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

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

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 $b \notin \mathrm{fn}(a(x).\overline{c}\langle x\rangle)$, 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

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

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

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 $b \notin \text{fn}(a(x).\overline{c}\langle x\rangle)$, 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

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

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

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

$$a(x).P \parallel (\nu b)\overline{a}\langle b\rangle.Q \longmapsto (\nu b)(P \parallel Q\{^{b}/_{x}\})$$

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

Another Example

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

Observe that fn$(P) = \{x, v, y\}$ and bn$(P) = \{z, w, u\}$.

Another Example

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

Observe that fn$(P) = \{x, v, y\}$ and bn$(P) = \{z, w, u\}$.
There are two possibilities for reduction.

Another Example

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

Observe that fn(P) $= \{x, v, y\}$ and bn(P) $= \{z, w, u\}$.
There are two possibilities for reduction.

1. Interaction among the first and second components:

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

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

Another Example

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

Observe that fn(P) $= \{x, v, y\}$ and bn(P) $= \{z, w, u\}$.
There are two possibilities for reduction.

1. Interaction among the first and second components:

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

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

2. Interaction among the second and third components:

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

Another Example

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

Observe that fn$(P) = \{x, v, y\}$ and bn$(P) = \{z, w, u\}$.
There are two possibilities for reduction.

1. Interaction among the first and second components:

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

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

2. Interaction among the second and third components:

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

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

Try it yourself

- Three agents: a printing server, a client, and a printer.
  The client wishes to print a document d.
- The client and the server share a public name b. The server and the printer share a name a.
- However, the client doesn't share names with the printer, so it cannot contact it.
- The document d cannot be transmitted along public names.

Reduction Semantics

- Intuitively, the reduction semantics focuses on the internal behavior of a process, without external intervention.
- Defined as a binary relation on processes coupled with structural congruence, denoted

$$P \longmapsto Q \qquad (\text{“}P \text{ reduces to } Q\text{”})$$

- Alternatively, one may endow processes with a semantics that captures the interaction between a process and its environment.
- To that end, we would need a different kind of semantics (based on a Labeled Transition System).

Reduction Semantics

The reduction relation, denoted $\longmapsto$, is defined by the following rules, which make our previous informal postulates formal:

$$(a(x).P + P') \parallel (\overline{a}\langle v\rangle.Q + Q') \longmapsto P\{^{v}\!/x\} \parallel Q$$

$$\frac{P \longmapsto P'}{P \parallel Q \longmapsto P' \parallel Q} \qquad \frac{P \longmapsto P'}{(\nu x)\, P \longmapsto (\nu x)\, P'} \qquad \frac{P \equiv P' \longmapsto Q' \equiv Q}{P \longmapsto Q}$$

Observe:

- Hence, $\equiv$ can occur at any point in the inference.
  It promotes behavior, by bringing together processes.
- Above, guarded choices the form $\alpha_1.P_1 + \cdots + \alpha_n.P_n$.

Outline

Taking Stock

Up to here:

- Software correctness (following Dijkstra)
- Concurrency and communication (following Milner)
- Formal models of concurrency, in particular process calculi
- The π-calculus, the paradigmatic calculus of concurrency: overview of its syntax and semantics

How to ensure that **communication-centric software systems** (specified in the π-calculus) respect their protocols?

Next:

- Type systems for concurrency: behavioral types
- Session types for the π-calculus
- Logical foundations for message-passing concurrency

Type-based Communication Correctness in Multi-agent Systems

Part I: Correctness, Communication, and Process Calculi

Jorge A. Pérez

Bernoulli Institute for Mathematics, Computer Science, and AI
University of Groningen, The Netherlands

www.jperez.nl

university of groningen

20th European Agent Systems Summer School (EASSS 2018)