Pi-calculus

types, bestiary

Francesco Zappa Nardelli

INRIA Rocquencourt, MOSCOVA research team

francesco.zappa_nardelli@inria.fr

MPRI Concurrency course with:

Pierre-Louis Curien (PPS), Roberto Amadio (PPS), Catuscia Palamidessi (INRIA Futurs)

Plan (first part of the lecture)

Objective:

reason about concurrent systems using types.

Plan:

1. *Types to prevent run-tme errors*:
   simply-typed pi-calculus, soundness, subtyping;

2. *Types to reason about processes*:
   typed equivalences, a labelled characterisation.

Types and sequential languages

In sequential languages, types are *"widely"* used:

- to detect simple programming errors at compilation time;
- to perform optimisations in compilers;
- to aid the structure and design of systems;
- to compile modules separately;
- to reason about programs;
- ahem, etc...

Data types and pi-calculus

In pi-calculus, the only values are *names*. We now extend pi-calculus with *base values* of type `int` and `bool`, and with *tuples*.

Unfortunately (?!) this allows writing terms which make no sense, as

$$\overline{x}\langle \mathtt{true} \rangle.P \;\Big|\Big|\; x(y).\overline{z}\langle y+1 \rangle$$

or (even worse)

$$\overline{x}\langle \mathtt{true} \rangle.P \;\Big|\Big|\; x(y).\overline{y}\langle 4 \rangle \;.$$

These terms raise *runtime errors*, a concept you should be familiar with.

Preventing runtime errors

We know that 3 : `int` and `true` : `bool`.

Names are values (they denote channels). Question: in the term

$$P \equiv \overline{x}\langle 3 \rangle . P'$$

which type can we assign to x?

Idea: state that x is a channel that can transport values of type `int`. Formally

$$x : \mathsf{ch}(\mathtt{int}) \ .$$

A complete type system can be developed along these lines...

Simply-typed pi-calculus: syntax and reduction semantics

Types:

$$T \;::=\; \mathsf{ch}(T) \;\mid\; T \times T \;\mid\; \texttt{unit} \;\mid\; \texttt{int} \;\mid\; \texttt{bool}$$

Terms (messages and processes):

$$M \;::=\; x \;\mid\; (M, M) \;\mid\; () \;\mid\; 1, 2, \ldots \;\mid\; \texttt{true} \;\mid\; \texttt{false}$$

$$P \;::=\; \mathbf{0} \;\mid\; x(y : T).P \;\mid\; \overline{x}\langle M\rangle.P \;\mid\; P \mid\mid P \;\mid\; (\boldsymbol{\nu} x : T)P \;\mid\; \mathsf{match}\ z\ \mathsf{with}\ (x : T_1, y : T_2)\ \mathsf{in}\ P \;\mid\; !P$$

Notation: we write $w(x, y).P$ for $w(z : T_1 \times T_2).\mathsf{match}\ z\ \mathsf{with}\ (x : T_1, y : T_2)\ \mathsf{in}\ P$.

Simply-typed pi-calculus: the type system

Type environment: $\Gamma \ ::= \ \emptyset \ \mid \ \Gamma, x{:}T.$

Type judgements:

- $\Gamma \vdash M : T$ value M has type T under the type assignement for names Γ;
- $\Gamma \vdash P$ process P respects the type assignement for names Γ.

Simply-typed pi-calculus: the type rules (excerpt)

Messages:

$$3 : \texttt{int} \qquad \frac{\Gamma(x) = T}{\Gamma \vdash x : T} \qquad \frac{\Gamma \vdash M_1 : T_1 \quad \Gamma \vdash M_2 : T_2}{\Gamma \vdash (M_1, M_2) : T_1 \times T_2}$$

Processes:

$$\Gamma \vdash \mathbf{0} \qquad \frac{\Gamma \vdash P_1 \quad \Gamma \vdash P_2}{\Gamma \vdash P_1 \mid\mid P_2} \qquad \frac{\Gamma, x{:}T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} x : T)P}$$

$$\frac{\Gamma \vdash x : \mathsf{ch}(T) \quad \Gamma, y{:}T \vdash P}{\Gamma \vdash x(y : T).P} \qquad \frac{\Gamma \vdash x : \mathsf{ch}(T) \quad \Gamma \vdash M : T \quad \Gamma \vdash P}{\Gamma \vdash \overline{x}\langle M \rangle.P}$$

Soundness

The soundness of the type system can be proved along the lines of Wright and Felleisen's *syntactic approach to type soundness*.

- extend the syntax with the `wrong` process, and add reduction rules to capture runtime errors:

$$\frac{\text{where } x \text{ is not a name}}{\overline{x}\langle M\rangle.P \xrightarrow{\tau} \texttt{wrong}} \qquad \frac{\text{where } x \text{ is not a name}}{x(y{:}T).P \xrightarrow{\tau} \texttt{wrong}}$$

- prove that if $\Gamma \vdash P$, with Γ closed, and $P \rightarrow\!\!\triangleright^* P'$, then P' does not have `wrong` as a subterm.

Soundness, ctd.

Lemma Suppose that $\Gamma \vdash P$, $\Gamma(x) = T$, $\Gamma \vdash v : T$. Then $\Gamma \vdash P\{^v\!/_x\}$.

Proof. Induction on the derivation of $\Gamma \vdash P$.

Theorem Suppose $\Gamma \vdash P$, and $P \xrightarrow{\alpha} P'$.

1. If $\alpha = \tau$ then $\Gamma \vdash P'$.
2. If $\alpha = a(v)$ then there is T such that $\Gamma \vdash a : \mathsf{ch}(T)$ and if $\Gamma \vdash v : T$ then $\Gamma \vdash P'$.
3. If $\alpha = (\boldsymbol{\nu}\tilde{x} : \tilde{S})\overline{a}\langle v\rangle$ then there is T such that $\Gamma \vdash a : \mathsf{ch}(T)$, $\Gamma, \tilde{x} : \tilde{S} \vdash v : T$, $\Gamma, \tilde{x} : \tilde{S} \vdash P'$, and each component of $\tilde{S}$ is a link type.

Proof. At the blackboard.

Subtyping

Idea: refine the type of channels $\mathsf{ch}(T)$ into

$\mathsf{i}(T)$	input (read) capability
$\mathsf{o}(T)$	output (write) capability

This form a basis for *subtyping*.

Example: the term

$$x : \mathsf{o}(\mathsf{o}(T)) \vdash (\boldsymbol{\nu} y : \mathsf{ch}(T))\ \overline{x}\langle y\rangle.!y(z : T)$$

is well-typed because $\mathsf{ch}(T) <: \mathsf{o}(T)$. Effect: well-typed contexts *cannot interfere* with the existing input, because they can only *write* at channel y.

The subtyping relation, formally

– *is a preorder*

$$T <: T \qquad\qquad \frac{T_1 <: T_2 \quad T_2 <: T_3}{T_1 <: T_3}$$

– *capabilities can be forgotten*

$$\mathsf{ch}(T) <: \mathsf{i}(T) \qquad\qquad \mathsf{ch}(T) <: \mathsf{o}(T)$$

– i *is a covariant type constructor,* o *is contravariant,* ch *is invariant*

$$\frac{T_1 <: T_2}{\mathsf{i}(T_1) <: \mathsf{i}(T_2)} \qquad \frac{T_2 <: T_1}{\mathsf{o}(T_1) <: \mathsf{o}(T_2)} \qquad \frac{T_2 <: T_1 \quad T_1 <: T_2}{\mathsf{ch}(T_1) <: \mathsf{ch}(T_2)}$$

Subtyping, ctd.

Intuition: if $x : \mathsf{o}(T)$ then it is safe to send along x values of of a subtype of T. Dually, if $x : \mathsf{i}(T)$ then it is safe to assume to assume that values received along x belong to a supertype of T.

Type rules must be updated as follows:

$$\frac{\Gamma \vdash x : \mathsf{i}(T) \quad \Gamma, y{:}T \vdash P}{\Gamma \vdash x(y : T).P} \qquad \frac{\Gamma \vdash x : \mathsf{o}(T) \quad \Gamma \vdash M : T \quad \Gamma \vdash P}{\Gamma \vdash \overline{x}\langle M \rangle.P}$$

$$\frac{\Gamma \vdash M : T_1 \quad T_1 <: T_2}{\Gamma \vdash M : T_2}$$

Exercises

Show that:

1. $a : \mathsf{ch}(\mathtt{int}), b : \mathsf{ch}(\mathtt{real}) \vdash \overline{a}\langle 5\rangle \;\|\; a(x).\bar{b}\langle x\rangle$, assuming $\mathtt{int} <: \mathtt{real}$;

2. $x : \mathsf{o}(\mathsf{o}(T)) \vdash (\boldsymbol{\nu} y : \mathsf{ch}(T))(\overline{x}\langle y\rangle.!y(z))$

3. $x : \mathsf{o}(\mathsf{o}(T)), z : \mathsf{o}(\mathsf{i}(T)) \vdash (\boldsymbol{\nu} y : \mathsf{ch}(T))(\overline{x}\langle y\rangle \;\|\; \overline{z}\langle y\rangle)$

4. $b : \mathsf{ch}(S), x : \mathsf{ch}(\mathsf{i}(S)), a : \mathsf{ch}(\mathsf{o}(\mathsf{i}(S))) \vdash \overline{a}\langle x\rangle \;\|\; x(y).y(z) \;\|\; a(x).\overline{x}\langle b\rangle$

Remarks on i/o types

– *different processes may have different visibility of a name*:

$$(\boldsymbol{\nu} x : \mathsf{ch}(T))\ \overline{y}\langle x\rangle.\overline{z}\langle x\rangle.P \parallel y(a : \mathsf{i}(T)).Q \parallel z(b : \mathsf{o}(T)).R \quad \longrightarrow\!\!\!\!\!\!\triangleright \longrightarrow\!\!\!\!\!\!\triangleright$$
$$(\boldsymbol{\nu} x : \mathsf{ch}(T))\ (P \parallel Q\{^x/_a\} \parallel R\{^x/_b\})$$

Q can only read from x, R can only write to x.

– *acquiring the* o *and* i *capabilities on a name is different from acquiring* ch: the term

$$(\boldsymbol{\nu} x : \mathsf{ch}(\mathtt{unit}))\ \overline{y}\langle x\rangle.\overline{z}\langle x\rangle \;\Big\|\; y(a : \mathsf{i}(\mathtt{unit})).z(b : \mathsf{o}(\mathtt{unit})).\overline{a}\langle\rangle$$

is not well-typed.

Types for reasoning

Types can be seen as *contracts* between a process and its environment: the environment *must respect* the constraints imposed by the typing discipline.

In turn, *types reduce the number of legal contexts* (and give us more process equalities).

Example: an observer whose typing is

$$\Gamma = a : \mathsf{o}(T), b : T, c : T' \qquad\qquad T \text{ and } T' \text{ unrelated}$$

- can offer an output $\overline{a}\langle b\rangle$;
- cannot offer an output $\overline{a}\langle c\rangle$, or an input at a.

A “natural” contextual equivalence, informally

Definition (informal): The processes P and Q are equivalent in Γ, denoted

$$P \cong_{\Gamma} Q$$

iff $\Gamma \vdash P, Q$ and they are equivalent in all the testing contexts that respect the types in Γ.

To formalize this equivalence we need to type contexts, at the blackboard...

Semantic consequences of i/o types

Example: the processes

$$P = (\boldsymbol{\nu} x)\overline{a}\langle x\rangle.\overline{x}\langle\rangle$$
$$Q = (\boldsymbol{\nu} x)\overline{a}\langle x\rangle.\mathbf{0}$$

and different in the untyped or simply-typed pi-calculus.

With i/o types, it holds that

$$P \cong_\Gamma Q \qquad \text{for } \Gamma = a : \mathsf{ch}(\mathsf{o}(\mathtt{unit}))$$

because the residual $\overline{x}\langle\rangle$ of P is deadlocked (the context cannot read from x).

Semantic consequences of i/o types, ctd.

Specification and an implementation of the factorial function:

$$\begin{array}{lcl} \mathsf{Spec} & = & !f(x,r).\overline{r}\langle \mathrm{fact}(x)\rangle \\ \mathsf{Imp} & = & !f(x,r).\mathtt{if}\ \ x=0\ \ \mathtt{then}\ \ \overline{r}\langle 1\rangle\ \ \mathtt{else}\ \ (\boldsymbol{\nu} r')\overline{f}\langle x-1,r'\rangle.r'(m).\overline{r}\langle x*m\rangle \end{array}$$

In general, Spec $\not\cong$ Imp. (Why?)

With i/o types, we can protect the input end of the function, obtaining

$$(\boldsymbol{\nu} f)\overline{a}\langle f\rangle.\mathsf{Spec} \ \cong_{\Gamma}\ (\boldsymbol{\nu} f)\overline{a}\langle f\rangle.\mathsf{Imp}$$

for $\Gamma = a : \mathsf{ch}(\mathsf{o}(\mathtt{int} \times \mathsf{o}(\mathtt{int})))$.

Semantic consequences of i/o types, ctd.

$$
\begin{aligned}
P &= (\boldsymbol{\nu} x, y)(\overline{a}\langle x\rangle \mid\mid \overline{a}\langle y\rangle \mid\mid !x().R \mid\mid !y().R) \\
Q &= (\boldsymbol{\nu} x)(\overline{a}\langle x\rangle \mid\mid \overline{a}\langle x\rangle \mid\mid !x().R)
\end{aligned}
$$

In the untyped calculus $P \ncong Q$: a context that tells them apart is

$$
- \mid\mid a(z_1).a(z_2).(z_1().\overline{c}\langle\rangle \mid\mid \overline{z_2}\langle\rangle) \;.
$$

With i/o types

$$
P \cong_\Gamma Q \qquad \text{for } \Gamma = a : \mathsf{ch}(\mathsf{o}(\mathtt{unit})) \;.
$$

Notation: I will often omit redundant type informations.

Exercise

1. Extend the syntax, the reduction semantics, and the type rules of pi-calculus with i/o types with the nondeterministic sum operator, denoted $+$;

2. Show that the terms

$$
\begin{aligned}
P &= \bar{b}\langle x\rangle.a(y).(y() \parallel \overline{x}\langle\rangle) \\
Q &= \bar{b}\langle x\rangle.a(y).(y().\overline{x}\langle\rangle + \overline{x}\langle\rangle.y())
\end{aligned}
$$

are not equivalent in the untyped calculus. Propose a i/o typing such that $P \simeq_\Gamma Q$.

References

Milner: *The polyadic pi-calculus - a tutorial*, ECS-LFCS-91-180.

Pierce, Sangiorgi: *Typing and subtyping for mobile processes*, LICS '93.

Boreale, Sangiorgi: *Bisimulation in name-passing calculi without matching*, LICS '98.

Sangiorgi, Walker: *The pi-calculus*, CUP.

...there is a large literature on the subject. The articles above have been reported because they are explicitely mentioned in this lecture.

Navigating through the literature

Pi-calculus literature describes **zillions** of slightly different languages, semantics, equivalencies.

Some slides for not getting lost.

Barbed congruence vs. reduction-closed barbed congruence

Let *barbed equivalence*, denoted $\cong^{\bullet}$, be the largest symmetric relation that is barb preserving and reduction closed. Barbed equivalence is not preserved by context, so define *barbed congruence*, denoted $\cong^{c}$, as

$$\{(P, Q) : C[P] \cong^{\bullet} C[Q] \text{ for every context C[-].}\}$$

- Barbed congruence is *more natural* and *less discriminating* than reduction-closed barbed congruence (for pi-calculus processes).
- Completeness of bisimulation for image-finite processes holds with respect to barbed congruence, but its proof requires transifinite induction.

Late bisimulation

Change the definition of the LTS:

$$x(y).P \xrightarrow{x(y)} P \qquad \frac{P \xrightarrow{\overline{x}\langle v\rangle} P' \quad Q \xrightarrow{x(y)} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'\{^v/_y\}}$$

and extend the definition of bisimulation with the clause: if $P \approx_l Q$ and $P \xrightarrow{x(y)} P'$, then there is Q' such that $Q \stackrel{x(y)}{\Longrightarrow} Q'$ and for all v it holds $P'\{^v/_y\} \approx_l Q'\{^v/_y\}$.

- Late bisimulation differs (slightly) from (early) bisimulation. More importantly, *the label $x(y)$ does not denote an interacting context*.

Ground bisimulation

Idea: play a standard bisimulation on the late LTS. Or,

Let *ground bisimulation* be the largest symmetric relation, $\approx_g$, such that whenever $P \approx_g Q$, there is $z \notin \mathrm{fn}(P, Q)$ such that if $P \xrightarrow{\alpha} P'$ where α is $\overline{x}\langle y \rangle$ or $x(z)$ or $(\boldsymbol{\nu} z)\overline{x}\langle z \rangle$ or τ, then $Q \stackrel{\hat{\alpha}}{\Longrightarrow} \approx_g P'$.

Contrast it with bisimilarity: to establish $x(z).P \approx x(z).Q$ it is necessary to show that $P\{^v/_z\} \approx Q\{^v/_z\}$ for all v. Ground bisimulation requires to test only *a single, fresh, name*.

However, ground bisimilarity is less discriminating than bisimilarity, and it is not preserved by composition (still, it is a reasonable equivalence for sublanguages of pi-calculus).

Open bisimulation

Full bisimilarity is the closure of bisimilairty under substitutions, and is a congruence with respect to all contexts. Unfortunately, full bisimilarity is not defined co-inductively.

Question: can we give a co-inductive definition of a useful congruence?

Yes, with *open bisimulation*.

Idea: (on the restriction free calculus) let $\bowtie$ be the largest symmetric relation such that whenever $P \bowtie Q$ and σ is a substitution, $P\sigma \xrightarrow{\alpha} P'$ implies $Q\sigma \overset{\hat{\alpha}}{\Longrightarrow} \bowtie P'$.

It is possible to avoid the σ quantification by means of an appropriate LTS.

Subcalculi

Idea: In pi-calculus contexts have a great discriminating power. It may be useful to consider other languages in which contexts "*observe less*", so that we have *more equations*.

Asynchronous pi-calculus: no continuation after an output prefix.

Localized pi-calculus: given $x(y).P$, the name y is not used as subject of an input prefix in P.

Private pi-calculus: only output of new names.

Conclusion: back to programming languages

Design choice:

bake into the definition of the language specific communication primitives?

- *yes*: Pict (Pierce et al.), NomadicPict (Sewell et al.), JoCaml (Moscova), ...
- *no*: Acute (Sewell et al., Moscova), ...

Some demos

...crossing fingers...