Concurrency theory

types to reason about processes: subtyping, receptiveness from process calculi to programming languages some slides to navigate through the literature

Francesco Zappa Nardelli

INRIA Rocquencourt, MOSCOVA research team

francesco.zappa_nardelli@inria.fr

together with

Frank Valencia (INRIA Futurs) Catuscia Palamidessi (INRIA Futurs) Roberto Amadio (PPS)

Simple types for pi-calculus

Objective: avoid run-time errors

$$\overline{x}\langle \texttt{true} \rangle.P \;\Big\|\; x(y).\overline{y}\langle 4 \rangle \;\twoheadrightarrow\; P \;\Big\|\; \overline{\texttt{true}}\langle 4 \rangle \;\twoheadrightarrow\; \mathit{error}$$

Ideas: associate a type to each channel: $x : \mathsf{ch}(\mathsf{bool})$. The free names of each process are stored in an environment Γ, that represents a *contract* between the process and the environment about the use of the channels. Processes willing to interact *must agree* on the contract.

$\overline{x}\langle \texttt{true} \rangle.P$	$\Gamma_1 = x : \mathsf{ch}(\mathsf{bool})$
$x(y).\overline{y}\langle 4 \rangle$	$\Gamma_2 = y : \mathsf{ch}(\texttt{int});\ x : \mathsf{ch}(\mathsf{ch}(\texttt{int}))$

Γ_1 and Γ_2 disagree on the type of x: the process above is not well typed.

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

$\Gamma \vdash M : T$ value M has type T under the type assignement for names Γ;

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

$\Gamma \vdash P$ process P respects the type assignement for names

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

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

Soundness

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 $P \rightarrowtriangle^* P'$, then P' does not have `wrong` as a subterm.

Types to reason about processes

Specification and an implementation of the factorial function:

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

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

Idea: the channel f should be *write-only* for the environment.

Subtyping

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

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

Example: a context that interacts with the term

$$x : \mathsf{ch}(\mathsf{o}(T)) \vdash (\boldsymbol{\nu} f : \mathsf{ch}(T))\ \overline{x}\langle f \rangle.\mathsf{Imp}$$

and respects the environment $x : \mathsf{ch}(\mathsf{o}(T))$ can only write at f.

Problem: f has type $\mathsf{ch}(T)$, not $\mathsf{o}(T)$.

However, in some sense $\mathsf{ch}(T)$ and $\mathsf{o}(T)$ are *compatible*.

Subtyping

We say that a type T_1 is a *subtype* of a type T_2, denoted $T_1 <: T_2$ if it is safe to use a value of type T_1 in all contexts that expect a value of type T_2. For instance:

- $\texttt{int} <:$ float: the value 3 can safely be used in the context $\texttt{sqr}(-)$.
- float $\not<:$ int: the value 3.14 cannot be used in the context $- \texttt{ mod } 3$ because `mod` is not defined over floats.
- $\{a{:}\texttt{int}; b{:}\texttt{int}\} <: \{a : \texttt{int}\}$: a context accepting records containing the label a will just ignore the existence of the label b.
- $\mathsf{ch}(T) <: \mathsf{o}(T)$: a context expecting a channel of type $\mathsf{o}(T)$ will just ignore the read capability available.

The subtyping relation, formally

– *is a preorder*

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

– *capabilities can be forgotten*

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

– 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. $a : \mathsf{ch}(\mathtt{int}), b : \mathsf{ch}(\mathtt{real}) \vdash \overline{a}\langle 5\rangle \parallel a(x).\overline{b}\langle x\rangle$, assuming $\mathtt{int} <: \mathtt{real}$;
2. $x : \mathsf{ch}(\mathsf{o}(T)) \vdash (\boldsymbol{\nu} y{:}\mathsf{ch}(T))(\overline{x}\langle y\rangle.!y(z{:}T))$
3. $x : \mathsf{ch}(\mathsf{o}(T)), z : \mathsf{ch}(\mathsf{i}(T)) \vdash (\boldsymbol{\nu} y{:}\mathsf{ch}(T))(\overline{x}\langle y\rangle \parallel \overline{z}\langle y\rangle)$
4. $b : \mathsf{ch}(S), x : \mathsf{ch}(\mathsf{i}(S)), a : \mathsf{ch}(\mathsf{o}(\mathsf{i}(S))) \vdash \overline{a}\langle x\rangle \parallel x(y).y(z) \parallel a(x).\overline{x}\langle b\rangle$

Remarks on i/o types

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

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

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

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

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 typed 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: a context $C[-]$ is a Δ/Γ-context if $\Delta \vdash C$ is a valid type judgement when the hole is typed as

$$\frac{\Omega \text{ extends } \Gamma}{\Omega \vdash -}$$

Main property: if C is a Δ/Γ-context and $\Gamma \vdash P$ then $\Delta \vdash C[P]$.

Semantic consequences of i/o types

Example: the processes

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

are 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.

Back to the factorial function:

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

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

Which is the type that I omitted after the restriction of f?.

Semantic consequences of i/o types, ctd.

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

In the untyped calculus $P \not\cong Q$: a context that tells them apart is

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

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.

Challenge: a labelled equivalence

Problem: only a subset of the actions of the tested process is observable; in general the typings of the observer and of the tested process do not coincide.

Example: in an initial type environment $\Gamma = a : \mathsf{ch}(\mathsf{o}(T))$, we have

$$(\boldsymbol{\nu} b : \mathsf{ch}(T))\overline{a}\langle b\rangle.P \xrightarrow{(\boldsymbol{\nu} b:\mathsf{ch}(T))\overline{a}\langle b\rangle} P \ .$$

The final typing for P is $\Gamma, b : \mathsf{ch}(T)$, while for the observer it is $\Gamma, b : \mathsf{o}(T)$.

Solution: separate the observer and the process points of vies on the types of the names.

The typings should however be typewise compatible (that is, the types of the same name should have a common subtype).

Challenge: a labelled equivalence, ctd.

Problem: aliasing! Going back to the example on slide 15:

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

It holds $P \cong_\Gamma Q$ for $\Gamma = a : \mathsf{ch}(\mathsf{o}(\mathtt{unit}))$. Consider this interaction with a tester:

$$\begin{aligned} a(p).a(q).R \parallel P &\rightarrow^* R\{^x/_p\}\{^y/_q\} \parallel P' \\ \underbrace{a(p).a(q).R}_{\text{the tester}} \parallel Q &\rightarrow^* R\{^x/_p\}\{^x/_q\} \parallel Q' \end{aligned}$$

At the beginning, the tester/observer is the same but after two reductions the observers are different. *In the bisimulation game the observer should be unique.*

Aliasing, ctd.

In other terms, equivalent terms may realise different sequence of transitions:

$$(\boldsymbol{\nu} x, y)(\overline{a}\langle x\rangle \;||\; \overline{a}\langle y\rangle \;||\; !x().R \;||\; !y().R) \xrightarrow{(\boldsymbol{\nu} x)\overline{a}\langle x\rangle} \xrightarrow{(\boldsymbol{\nu} y)\overline{a}\langle y\rangle} \xrightarrow{y(v)} \ldots$$

must be matched by the sequence

$$(\boldsymbol{\nu} x)(\overline{a}\langle x\rangle \;||\; \overline{a}\langle x\rangle \;||\; !x().R) \xrightarrow{(\boldsymbol{\nu} x)\overline{a}\langle x\rangle} \xrightarrow{\overline{a}\langle x\rangle} \xrightarrow{x(v)} \ldots$$

Solution: separate the observer's view on the identity of names from their real identity.

Digression

Aliasing is not related to types. The same problem arises if we consider a dialect of pi-calculus *without matching*.

In fact, the transition $P \xrightarrow{\overline{x}\langle y\rangle} P'$ can be read as:

1. P is interacting with a context ready to receive a name over x;

2. there exists a context that can test that P is ready to send the name y over x.

Interpretation 2. requires that the context can test if the name sent over x is y or not (eg, matching is required). *Completeness of the standard bisimulation wrt contextual equivalence is lost if matching is omitted.*

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

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

   are not equivalent in the untyped calculus. Propose a i/o typing such that $P \simeq_\Gamma Q$. If you like challenges, then prove this by exhibiting an appropriate typed bisimulation.

Receptiveness

A local environment:

$$(\boldsymbol{\nu} x)(!x(y).R \;||\; Q)$$

(Q and R have only the output capability on x).

The name x is stateless, and its input end is always available: x is *uniformly receptive*.

Uniform receptiveness occurs when modelling functions, objects, RPC protocols, etc. It is a common idiom in programming languages based on pi-calculus (Pict, Join, Blue): `def` $x(y) = R$ `in` Q. Variant: *linear receptiveness*: x is used at most once.

It is possible to impose receptiveness using *syntactical constraints*, or *a type system*.

Some properties of receptiveness

If x is *uniformly receptive* in Γ, then it holds that

$$\overline{y}\langle x\rangle.P \approx_\Gamma (\boldsymbol{\nu} z)\overline{y}\langle z\rangle.(!z(a).\overline{x}\langle a\rangle \parallel P)$$

for z fresh.

(an output of a global name becomes an output of a fresh name)

Not true in the ordinary pi-calculus because of contexts like

$$y(u).\overline{u}\langle c\rangle.\overline{p}\langle\rangle \parallel -$$

But this context is not receptive on u.

Some properties of receptiveness, ctd.

If x is *uniformly receptive* in Γ, then it holds that

$$\overline{x}\langle v\rangle.P \approx_\Gamma \overline{x}\langle v\rangle \parallel P$$

(makes a synchronous communication into an asynchronous one).

If $P \rightarrowtriangle P'$ by means of a communication at x, then

$$P \approx_\Gamma P'$$

(insensitiveness to internal reductions)

Remark: all names uniformly receptive $\Rightarrow$ confluence.

Factorial function again

$$S1 \quad = \quad !f(x{:}\mathsf{int}, y{:}\mathsf{int}, r{:}\mathsf{lrec}(\mathsf{int})).\quad \texttt{if}\ x = 0\ \texttt{then}\ \overline{r}\langle y\rangle \\ \texttt{else}\ (\boldsymbol{\nu} r')(\overline{f}\langle x-1, x*y, r'\rangle.r'(m).\overline{r}\langle m\rangle)$$

$$S2 \quad = \quad !f(x{:}\mathsf{int}, y{:}\mathsf{int}, r{:}\mathsf{lrec}(\mathsf{int})).\quad \texttt{if}\ x = 0\ \texttt{then}\ \overline{r}\langle y\rangle \\ \texttt{else}\ \overline{f}\langle x-1, x*y, r\rangle$$

Exercise: Give a (non-well typed) context that shows that S_1 and S_2 are not equivalent.

However, taking into account the receptiveness of f, r and r', S_1 and S_2 are behaviourally equivalent (*tail-call optimisation*).

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.

From process languages to programming languages

Implemementations of the pi-calculus semantics include:

- Pict : statically typed programming language based on the pi-calculus (mostly local computation)

- Nomadic Pict: communication is local to each runtime + migration between runtimes

Implementing in a distributed setting the pi-calculus semantics of channels requires solving the *distributed consensus*.

The join calculus

The join-idea: receptors and channels defined at the same time.

In informal syntax:

```
let C1() | C2() = P1
or  C2          = P2
```

defines channels `C1`, `C2`, and reactions `P1` and `P2`.

Consequence: all receptors are known *statically*.

- Synchronizations are solved locally;
- Static resolution of many problems (automata for synchronization, implicit polymorphic typing, etc.)

A "serious" process language

The definition

```
let Count(n) | Tick() = Count(n+1)
or  Count(n) | Show() = Count(n) | Print(n)
```

translates to rules

$$\texttt{Count(n)} \parallel \texttt{Tick()} \triangleright \texttt{Count(n+1)}$$
$$\texttt{Count(n)} \parallel \texttt{Show()} \triangleright \texttt{Count(n)} \parallel \texttt{Print(n)}$$

The semantics performs rewriting modulo equivalence (similarly to what we did when we defined the reduction semantics using structural equivalence):

$$\texttt{Count(2)} \parallel \texttt{Tick()} \parallel \texttt{Show()} \longrightarrow \texttt{Count(3)} \parallel \texttt{Show()}$$

(the new) JoCaml: a "real" programming language

- An extension of OCaml 3.10;
- distributed asynchronous channels (synchronous channels by CPS);
- polymorphic typing à la ML;
- easy encoding of concurrent programming constructs.
  For instance, what does the code below do?

```
type 'a buffer = { get : unit -> 'a ; put : 'a -> unit ; }

let create_buff () =
  def some(v) & get() = none() & reply v to get
  or  none()  & put(v) = some(v) & reply () to put in
  spawn none() ;
  { get = get ; put = put ; }
```

A more useful buffer

A classic algortihm that represents state by a pair of lists.

```
def state(xs,ys) & put(x) =
      state(x::xs,ys) & reply () to put
 or state(xs,y::ys) & get() =
      state(xs,ys) & reply y to get
 or state(_::_ as xs,[]) & get() =
      state([], List.rev xs) & reply get() to get
```

Try it:

`http://jocaml.inria.fr`

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 transfinite 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 $Q\sigma \stackrel{\hat{\alpha}}{\Longrightarrow} \bowtie P'$.

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.

Distribution, action at distance, and mobility

The parallel composition operator of CCS and pi-calculus does not specify whether the concurrent threads are running on the same machine, or on different machines connected by a network.

Some phenomena typical of distributed systems require a finer model, that explicitly keeps track of the spatial distribution of the processes.

We will briefly sketch two models that have been proposed: *DPI* (Hennessy and Riely, 1998) and *Mobile Ambients* (Cardelli and Gordon, 1998).

The aim of this section is to get a glimpse of more complex process languages, and to rediscover the idea of "transitions in an LTS characterise the interactions a term can have with a context" in this setting.

DPI, design choices

- add explicit locations to pi-calculus processes: $\ell[\![\, P \,]\!]$;
- locations are identified by their name: $\ell[\![\, P \,]\!] \parallel \ell[\![\, Q \,]\!] \equiv \ell[\![\, P \parallel Q \,]\!]$;
- communication is local to a location:

$$\ell[\![\, \overline{x}\langle y \rangle.P \,]\!] \;\Big\|\; \ell[\![\, x(u).Q \,]\!] \;\rightarrowtriangle\; \ell[\![\, P \,]\!] \;\Big\|\; \ell[\![\, Q\{^{y}/_{u}\} \,]\!] \;;$$

- add explicit migration: $\ell[\![\, \text{goto } k.P \,]\!] \;\rightarrowtriangle\; k[\![\, P \,]\!]$.

We also include the restriction and match operators, subject to the usual pi-calculus semantics.

Behavioural equivalence for DPI

Again, we apply the standard recipe:

- define the suitable contexts:

$$C[-] \;::=\; - \;\mid\; C[-] \,\|\, \ell[\![\, P \,]\!] \;\mid\; (\boldsymbol{\nu} n) C[-] \;.$$

- define the observation:

$$M \downarrow x@\ell \text{ iff } P \equiv (\boldsymbol{\nu}\tilde{n})(\ell[\![\, x(u).P' \,]\!] \,\|\, P'') \text{ for } x, \ell \notin \tilde{n} \;.$$

Can we characterise this equivalence with a labelled bisimulation?

Labelled bisimulation for DPI

$$\frac{P \rightarrowtriangle P'}{P \xrightarrow{\tau} P'}$$

$$\frac{P \equiv (\boldsymbol{\nu}\tilde{n})(\ell[\![\, x(u).P' \,]\!] \parallel P'') \qquad x, \ell \notin \tilde{n}}{P \xrightarrow{x(y)@\ell} (\boldsymbol{\nu}\tilde{n})(\ell[\![\, P'\{{}^{y}/_{u}\} \,]\!] \parallel P'')}$$

$$\frac{P \equiv (\boldsymbol{\nu}\tilde{n})(\ell[\![\, \overline{x}\langle y\rangle.P' \,]\!] \parallel P'') \qquad x, y, \ell \notin \tilde{n}}{P \xrightarrow{\overline{x}\langle y\rangle@\ell} (\boldsymbol{\nu}\tilde{n})(\ell[\![\, P' \,]\!] \parallel P'')}$$

$$\frac{P \equiv (\boldsymbol{\nu}\tilde{n})(\ell[\![\, \overline{x}\langle y\rangle.P' \,]\!] \parallel P'') \qquad x, \ell \notin \tilde{n} \qquad y \in \tilde{n}}{P \xrightarrow{\overline{x}\langle (y)\rangle@\ell} (\boldsymbol{\nu}\tilde{n} \setminus y)(\ell[\![\, P' \,]\!] \parallel P'')}$$

Labelled bisimulation for DPI, ctd.

The standard bisimulation on top of the LTS below coincides with reduction barbed congruence.

Remark: the LTS is written in an *unconventional* style, which precisely characterises the interactions a term can have with a context.

Questions:

1- every label should correspond to a (minimal) interacting context: can you spell out these contexts?

2- why there are no explicit labels for the "goto" action?

Mobile Ambients, design choices

Objective: build a process language on top of the concepts of barriers (administrative domains, firewalls, ...) and of barrier crossing.

A graphical representation of the syntax and of the reduction semantics of Mobile Ambients can be found here:

`http://research.microsoft.com/Users/luca/Slides/`
`2000-11-10%20Wide%20Area%20Computation%20(Valladolid).pdf`

Mobile Ambients syntax (in ISO 10646)

Processes:

$$
\begin{array}{lcl}
P, Q, R & ::= & \mathbf{0} \\
& \mid & P_1 \parallel P_2 \\
& \mid & (\boldsymbol{\nu} n) P \\
& \mid & n[P] \\
& \mid & C.P \\
& \mid & !P
\end{array}
$$

Capabilities:

$$
\begin{array}{lcl}
C & ::= & \texttt{in\_}n \\
& \mid & \texttt{out\_}n \\
& \mid & \texttt{open\_}n
\end{array}
$$

Mobile Ambients: interaction

- Locations migrate under the control of the processes located at their inside:

$$n[\texttt{in\_}m.P \parallel Q] \parallel m[R] \rightarrow m[\,n[P \parallel Q] \parallel R\,]$$
$$m[\,n[\texttt{out\_}m.P \parallel Q] \parallel R\,] \rightarrow n[P \parallel Q] \parallel m[R]$$

- a location may be opened:

$$\texttt{open\_}n.P \;\Big\|\; n[\,Q\,] \rightarrow P \;\Big\|\; Q$$

Hint about an LTS for Mobile Ambients

Consider the term $M \equiv (\boldsymbol{\nu}\tilde{m})(k[\texttt{in\_}n.P \parallel Q] \parallel R)$ where $k \notin \tilde{m}$. It can interact with the context $n[T] \parallel -$, where T is an arbitrary process, yielding $O \equiv (\boldsymbol{\nu}\tilde{m})(n[T \parallel k[P \parallel Q]] \parallel R)$. This interaction can be captured with a transition $M \xrightarrow{k.\texttt{enter\_}n} O$.

Remark that, contrarily to what happens in CCS and pi-calculus, a bit of the interacting context is still visible in the outcome!

Along these lines (asynchrony is needed too!) it is possible to characterise reduction barbed congruence using a labelled bisimilarity.

References

James Riely, Matthew Hennessy: *Distributed Pprocesses and location failures*. Theoretical Computer Science, 2001. An extended abstract appeard in ICALP 97.

Luca Cardelli, Andrew Gordon: *Mobile Ambients*. Theoretical Computer Science, 2000. An extended abstract appeared in FOSSACS 1998.

Massimo Merro, myself: *A behavioral theory for Mobile Ambients*. Journal of ACM, 2005.

Non-interleaving models of concurrency

All the models we have considered (traces, completed traces, bisimulation semantics) identify parallelism with nondeterminstic interleaving of atomic actions.

We briefly describe two models where concurrency is modelled explicitly in the form of independence between actions:

- Mazurkiewicz traces;
- Event structures.

Mazurkiewicz trace languages

Definition: A *Mazurkiewicz trace language* consists of (M, L, I) where L is a set, $I \subseteq L \times L$ is a symmetric, irreflexive relation called the *independence* relation, and M is a nonempty subset of strings L^* such that

- *prefix closed*: $sa \in M \Rightarrow s \in M$ for all $s \in L^*, a \in L$;
- *I-closed*: $sabt \in M \wedge aIb \Rightarrow sbat \in M$ for all $s, t \in L^*, a, b \in L$;
- *coherent*: $sa \in M \wedge sb \in M \wedge aIb \Rightarrow sab \in M$ for all $s \in L^*, a, b \in L$.

Mazurkiewicz trace languages, ctd.

Definition: Let (M, L, I) be a Mazurkiewicz trace language. For $s, t \in M$ define $\equiv$ to be the smallest equivalence relation such that

$$sabt \equiv sbat \text{ if } aIB$$

for $sabt, sbat \in M$. Call an equivalence class $\{s\}_{\equiv}$ for $s \in M$ a *trace*. For $s, t \in M$ define

$$s < t \Leftrightarrow \exists u.\ su \equiv t\ .$$

The quotient $< / \equiv$ is a partial order on traces.

Event structures

Often there is no point in analysing the precise times of *events* in a distributed computation. What is important is how an event *causally depends* on the previous occurrences of others, and how an event *rules out* the occurrence of others.

Definition: an *event structure* is a structure $(E, \leq, \#)$ consisting of a a set E, of events which are partially ordered by $\leq$, the causal dependency relation, and a binary, symmetric, irreflexive relation $\# \subseteq E \times E$, the conflict relation, which satisfy

$$\{e' : e' \leq e\} \text{ is finite}$$

$$e\#e' \subseteq e'' \Rightarrow e\#e''$$

Two events are *concurrent* iff $\neg(e \leq e' \text{ or } e' \leq e \text{ or } e\#e')$.

Event structures, ctd.

The state of a computation is represented by a *configuration*.

A *configuration* is a subset $x \subseteq E$ which are

- *conflict-free*: $\forall e, e' \in x.\ \neg(e \# e')$ and
- *downwards-closed*: $\forall e, e'.\ e' \leq e \in x \Rightarrow e' \in x$.

If you want to know more: Nielsen, Winskel, *Models of concurrency*.

Summary

- syntax and semantics of CCS:
  - non-determinism, parallel composition: from automata to CCS
  - LTS for CCS: compositional definition of automata
  - reduction semantics for CCS
- equivalences:
  - traces, completed traces, failures, simulation, bisimulation
  - from strong to weak equivalences
  - proof techniques for bisimulation (up-to bisimulation, up-to context)
  - axiomatisation (proof of soundness and completeness in the the strong case)
  - Hennessy-Milner logic (proof of soundness and completeness)
- name passing:
  - syntax and reduction semantics of pi-calculus
  - data structures as processes

- contextual equivalence:
  - relationship between contextual equivalences and labelled equivalence (proof of sonudness and completeness for CCS in the weak case)
  - derivation of the early LTS for pi-calculus
  - proof of soundness of bisimilarity for pi-calculus, and counter-example to completeness
- asynchronous interaction
  - asynchronous contextual equivalences
  - how to build an LTS for asynchronous pi-calculus (ACS and HT)
- functions as processes:
  - cbv and cbn CPS transform
  - encoding of cbv and cbn lambda calculus in pi-calculus
- types:
  - types to avoid errors (simple types, proof of type preservation)
  - types to reason about processes: subtyping, receptiveness, types and contextual equivalence

Partiel

19/11/07 — 12h45-15h45

Salle 0C2 — Chevaleret

Personal notes and lecture notes authorised.

Thank you for your attention

Feel free to contact me if you want to know more about research in concurrency.

francesco.zappa_nardelli@inria.fr