The behaviour of typed processes

Davide Sangiorgi

INRIA Sophia Antipolis
2004 route des Lucioles, BP 93
06902 Sophia Antipolis CEDEX

Email: `davide.sangiorgi@inria.fr`
`http://www-sop.inria.fr/mimosa/personnel/Davide.Sangiorgi/`

December 8, 2000

CONTENTS

Types in languages for process mobility

- A rich spectrum.
- Usual advantages (static detection of programming errors, compiler optimisations, design of large systems, etc.).
- Also used to guarantee behavioural properties (interference, security, deadlock, etc.).
- Examples: π-calculus, Spi-calculus, Ambients, Facile, etc.

In these typed languages, standard behavioural equivalences (bisimilarity, trace equivalence, etc.) are over-discriminanting

$$\boxed{\text{What is the behaviour of typed processes?}}$$

In this talk: we focus on π-*calculus* and *bisimilarity*

CONTENTS

The untyped π-calculus

$a, b, \ldots, x, y, z, \ldots$			*Names*
			Values
v	$::=$	a	names
			Processes
P	$::=$	$P \mid P$	parallel
	$\mid$	$(\boldsymbol{\nu} a)\ P$	restriction
	$\mid$	$v(x).\,P$	input
	$\mid$	$\overline{v_1}\, v_2$	output (asynchronous)
	$\mid$	$!P$	replication ($= P \mid P \ldots$)

- Examples of communications:

$$a(y).\,Q \mid \overline{a}\, b \quad \xrightarrow{\tau} \quad Q\{^b\!/\!_y\}$$

$$a(y).\,Q \mid (\boldsymbol{\nu} b)\ (\overline{a}\, b \mid P) \quad \xrightarrow{\tau} \quad (\boldsymbol{\nu} b)\ (Q\{^b\!/\!_y\} \mid P)$$

(b fresh for Q)

A typed π-calculus (tuples)

			Values
v	$::=$	x	name
	$\mid$	$\langle v_1 \ldots v_n \rangle$	tuple of values
			Processes
P	$::=$	$v(x).\, P$	input
	$\mid$	$\overline{v_1}\, v_2$	output
	$\mid$	with $(x_1 \ldots x_n) = v$ do P	tuple destructor
	$\mid$	$\ldots$	$\ldots$
			Types
T	$::=$	$\sharp\, T$	communication color="red typ
	$\mid$	$\langle T_1 \ldots T_n \rangle$	tuple types
	$\mid$	$\ldots$	(possibly) base types, recursive types
			Type environments
Γ	$::=$	$\Gamma, x : T$	
	$\mid$	$\emptyset$	

Additional operational rules:

$$\text{with } (x_1 \ldots x_n) = \langle v_1 \ldots v_n \rangle \text{ do } P \xrightarrow{\tau} P\{{}^{v_1 \ldots v_n}\!/\!{}_{x_1 \ldots x_n}\}$$

We can then define as an **abbreviation**:

$$y(x_1 \ldots x_n).\, P \stackrel{\text{def}}{=} y(z).\ \text{with } (x_1 \ldots x_n) = z \text{ do } P$$

Type judgements

$$\Gamma \vdash v : T \qquad\qquad \Gamma \vdash P$$

Meaning:

"value v has type T under the type assignment for names Γ",

"process P respects the type assignment for names Γ".

A sample of typing rules

values:

$$\frac{\Gamma(v) = T}{\Gamma \vdash v \;:\; T} \qquad\qquad \frac{\Gamma \vdash v_i \;:\; T_i}{\Gamma \vdash \langle v_1 .. v_n \rangle \;:\; \langle T_1 .. T_n \rangle}$$

processes:

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad\qquad \frac{\Gamma \vdash v \;:\; \langle T_1 .. T_n \rangle \qquad \Gamma, x_1 : T_1 .. x_n : T_n \vdash P}{\Gamma \vdash \text{ with } (x_1 .. x_n) = v \text{ do } P}$$

$$\frac{\Gamma \vdash w \;:\; \sharp T \qquad \Gamma \vdash v \;:\; T}{\Gamma \vdash \overline{w}\, v} \qquad\qquad \frac{\Gamma \vdash w \;:\; \sharp T \qquad \Gamma, y : T \vdash P}{\Gamma \vdash w(y).\, P}$$

Types can be enriched in many ways:

- **Familiar type constructs** (in place of, or in addition to, products), ex: union, records, variants, basic values, functions, linearity, polymorphism, etc.

- Type constructs **specific to processes**, ex: process-passing, receptiveness, deadlock-freedom, sessions, etc.

CONTENTS

Channel types ($\sharp\, T$) are refined by adding:

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

$\Rightarrow$ basis for **subtyping** (also, some very basic **security** guarantees, cf: Klaim)

Example: $x : \mathsf{o}\,\mathsf{o}\, T \vdash (\boldsymbol{\nu} y : \sharp\, T)\,(\overline{x}\, y \mid\, !y(z).\mathbf{0})$ (true, since $\sharp\, T < \mathsf{o}\, T$)

i —	$\Rightarrow$	covariance
o —	$\Rightarrow$	contravariance
$\sharp$ —	$\Rightarrow$	invariance

The typed π-calculus in most this talk: monadic, with i/o types

i/o subtyping validates the subtyping rule for functions:

Following Milner,

$$[\![\lambda x.\, M]\!]_p \stackrel{\mathsf{def}}{=} (\boldsymbol{\nu} x)\, (\overline{p}\, x \mid !x(z,y).\, [\![M]\!]_y)$$

If $\lambda x.\, M : S \rightarrow T$, then $p : \mathsf{o}\, \mathsf{o}\, \langle [\![S]\!], \mathsf{o}\, [\![T]\!] \rangle$

which gives this translation on arrow types:

$$[\![S \rightarrow T]\!] \stackrel{\mathsf{def}}{=} \mathsf{o}\, \langle [\![S]\!], \mathsf{o}\, [\![T]\!] \rangle$$

where $[\![S]\!]$ is in contravariant position, $[\![T]\!]$ is in covariant position.

CONTENTS

Process equality (informally)

Two processes are **equal** if no contexts of the language can tell them apart.
Examples: barbed equivalence, may testing, ...

Notation:

$$P \approx_\Gamma Q \qquad \begin{cases} \Gamma \vdash P, Q\ ; \\ \text{if the language is untyped, } \Gamma \text{ is trivial} \end{cases}$$

The testing contexts must respect the types in Γ

Types: **contracts** between a process and its environment
$\Rightarrow$ reduces the number of "legal" contexts.
$\Rightarrow$ more process equalities.

Semantic consequences of i/o types

Abbr.: $\overline{x}$ and $x. P$
when value at x not important
(think it is of `unit` type)

Example 1

$$
\begin{array}{lcl}
P & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} x)\,(\overline{a}\,x \mid \overline{x}) \\
Q & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} x)\,\overline{a}\,x
\end{array}
$$

In untyped π-calculus: $P \not\approx_\Gamma Q$

With i/o types: $P \approx_\Gamma Q$, for $\Gamma \stackrel{\text{def}}{=} a : \sharp\,\mathtt{o}\,\mathtt{unit}$

Example 2

$$
\begin{array}{lcl}
P & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} x\,,\, y)\,(\overline{a}\,x \;\mid \overline{a}\,y \;\mid !x.\,R \mid !y.\,R) \\
Q & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} x)\,(\overline{a}\,x \;\mid \overline{a}\,x \;\mid !x.\,R)
\end{array}
$$

In untyped π-calculus: $P \not\approx_\Gamma Q$

A context that distinguishes them:

$$[\cdot] \mid a(z_1).\,a(z_2).\,(z_1.\,\overline{c} \mid \overline{z_2})$$

With i/o types: $P \approx_\Gamma Q$.

if $\Gamma \vdash a \;:\; \sharp\,\mathtt{o}\,\mathtt{unit}$

CONTENTS

Process equality, continued

Due to the universal quantification on contexts in the definition of equality:

– proofs tend to be hard

– the meaning of process equality is not clear

- We wish to express the behaviour of processes by means of the **transitions** the processes can make (cf: bisimilarity, trace equivalence, etc.)

- Examples of transitions:

$$
\begin{array}{rcl}
a(x).\,P & \xrightarrow{a\,v} & P\{{}^{v}\!/\!_{x}\} \\
\overline{a}\,v \mid P & \xrightarrow{\overline{a}\,v} & P \\
a(x).\,P \mid \overline{a}\,v \mid Q & \xrightarrow{\tau} & P\{{}^{v}\!/\!_{x}\} \mid Q \\
(\boldsymbol{\nu} b : T)\,(\overline{a}\,b \mid P) & \xrightarrow{(\boldsymbol{\nu} b : T)\,\overline{a}\,b} & P
\end{array}
$$

Notation: we sometimes omit type annotations in restrictions

Example: bisimilarity ($\simeq$)

$P \simeq Q$ implies

– if $P \stackrel{\mu}{\Longrightarrow} P'$ then there exists Q' s.t. $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \simeq Q'$ (for $\mu \neq \tau$);

– $\ldots$

$\Rightarrow$ the behaviour of a process is, essentially, the synchronisation trees it generates

(cf.: categorical characterisations of bisimilarity: Abramsky, Joyal, Nielsen, Winskel, Fiore, Cattani, Power , Rutten, Turi, Honsell, Lenisa, Montanari, Pistore, ...)

Theorem In CCS and (full) π-calculus, bisimilarity and barbed equivalence coincide.

Problem: in typed calculi bisimilarity is far too strong

In Examples 1 and 2 of i/o types, regardless of the types of names:

$$(\boldsymbol{\nu} x)\,(\overline{a}\,x \mid \overline{x}) \xrightarrow{\boldsymbol{\nu} x\,\overline{a}\,x} \xrightarrow{\overline{x}}$$

$$(\boldsymbol{\nu} x)\,\overline{a}\,x \xrightarrow{\boldsymbol{\nu} x\,\overline{a}\,x}$$

$$(\boldsymbol{\nu} x, y)\,(\overline{a}\,x \mid \overline{a}\,y \mid !x.\,R \mid !y.\,R) \xrightarrow{\boldsymbol{\nu} x\,\overline{a}\,x} \xrightarrow{\boldsymbol{\nu} y\,\overline{a}\,y}$$

$$(\boldsymbol{\nu} x)\,(\overline{a}\,x \mid \overline{a}\,x \mid !x.\,R) \xrightarrow{\boldsymbol{\nu} x\,\overline{a}\,x} \xrightarrow{\overline{a}\,x}$$

N.B: the processes are not even trace equivalent!

What is the behaviour (= bisimilarity) in typed calculi?

Next few slides: based on work with Boreale (lics'98)

We shall replace the usual (very informal) interpretation of bisimulation:

Bisimilar processes **perform the same actions** and their derivatives are again bisimilar.

with

Bisimilar processes **react to the same actions offered** by the **observer** and their derivatives are again bisimilar.

Idea (first try): represent the observer as a type environment.

Example: An observer whose typing is

$$\Gamma \stackrel{\text{def}}{=} a : \mathsf{o}\, T, b : T, c : S \qquad\qquad (T \text{ and } S \text{ unrelated})$$

– can offer an output $\overline{a}\, b$

– cannot offer an output $\overline{a}\, c$, or an input at a.

We look for something like $\Gamma \triangleright P \simeq Q$, meaning

"$\Gamma \vdash P, Q$, and P and Q cannot be distinguished by a well-typed observer that respects the type constraints in Γ".

Problem 1: the typings of the observer and a tested process need not coincide

Example: Initial common typing: $\Gamma \stackrel{\text{def}}{=} a : \sharp \circ T$

Process's transition: $P \xrightarrow{(\boldsymbol{\nu} b\, :\, \sharp T)\, \overline{a}\, b} P'$

Final typing for P': $\Gamma, b : \sharp T$

Final typing for the observer: $\Gamma, b : \circ T$

$$\boxed{\text{Solution: separate the observer and the process points of views on the types of the names}}$$

[The typings should however be typewise compatible (= the types of the same name should have a common subtype)]

Problem 2: aliasing

Going back to Example 2 (page 13):

$$\begin{array}{lcl} a(p).\,a(q).\,R \mid P & \Longrightarrow & R\{^x/_p\}\{^y/_q\} \mid P' \\ \underbrace{a(p).\,a(q).\,R}_{\text{the tester}} \mid Q & \Longrightarrow & R\{^x/_p\}\{^x/_q\} \mid Q' \end{array}$$

At the beginning:	the tester is the same
At the end:	the tester processses are different

But in the bisimulation game the observer/tester should be unique!

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

- *Typed closures*:

$$[\Delta\,;\,\sigma] \qquad \begin{cases} \Delta \text{ (a typing environment) is the observer's perspective;} \\ \sigma \text{ (a function from names to names) is the ``God's-eye'' perspective.} \end{cases}$$

- A transition relation on closures (*allow relation*):

$$[\Delta\,;\,\sigma] \xrightarrow[\delta]{\mu} [\Delta'\,;\,\sigma'] \quad \begin{cases} \delta & : & \text{the observer's point of view on the action,} \\ & & \text{he/she offers (n.b.: } \delta \text{ respects the types in } \Delta). \\ \mu = \delta\sigma & : & \text{the concrete action offered} \end{cases}$$

Example:

$$\Delta \stackrel{\text{def}}{=} x : \mathsf{o}\,T, y : T, p : \mathsf{i}\,T$$

$$\sigma \stackrel{\text{def}}{=} \{p/x, b/y, p/p, \}$$

$$[\Delta\,;\,\sigma] \quad \xrightarrow[\overline{x}\,y]{\overline{p}\,b} \quad [\Delta\,;\,\sigma] \qquad\qquad [\Delta\,;\,\sigma] \quad \xrightarrow[p\,z]{p\,b} \quad [\Delta, z : T\,;\,\sigma\{b/z\}] \quad (z \text{ fresh})$$

A closure $[\Delta\,;\,\sigma]$ and a process P form a **configuration** $[\Delta\,;\,\sigma] * P$.

P's typing should 'agree' with $[\Delta\,;\,\sigma]$.

$$[\Delta\,;\,\sigma] * P \stackrel{\delta}{\Longrightarrow} [\Delta'\,;\,\sigma'] * P'$$

$$\stackrel{\mathsf{def}}{=}$$

$$[\Delta\,;\,\sigma] \xrightarrow[\delta]{\mu} [\Delta'\,;\,\sigma'] \quad \text{and } P \stackrel{\overline{\mu}}{\Longrightarrow} P'.$$

Typed bisimilarity:

The largest symmetric relation $\simeq$ on compatible configurations such that $H \simeq K$ implies

1. if $H \overset{\delta}{\Longrightarrow} H'$ then there is K' s.t. $K \overset{\delta}{\Longrightarrow} K'$ and $H' \simeq K'$;

2. if $H \xrightarrow{\langle \mu, \lambda \rangle} H'$ then there is K' s.t.

   – either $K \xrightarrow{\langle \mu, \lambda \rangle} K'$,
   – or $K \overset{\mu}{\Longrightarrow}\overset{\lambda}{\Longrightarrow} K'$, or $K \overset{\lambda}{\Longrightarrow}\overset{\mu}{\Longrightarrow} K'$

   and $H' \simeq K'$.

The meaning of "matching transitions" is quite different from standard bisimilarity.

Another subtle point ...

Consider a closure $[\Delta\,;\,\sigma] \stackrel{\mathsf{def}}{=} [z : T, x : \mathsf{o}\,T, y : \mathsf{i}\,T, \;;\; \{{}^{b}\!/\!_{z}, \{{}^{a}\!/\!_{x}, \{{}^{a}\!/\!_{y}]$

The observer can perform, internally, an interaction using names x and y.

Thus we also have a transition: $[\Delta\,;\,\sigma] \xrightarrow[\overline{x}\,z\,,\,x(w)]{-} \quad \ldots$

$\Rightarrow$ the second clause in the definition of typed bisimilarity

Theorem Typed bisimilarity coincides with typed barbed equivalence (as usual, on image-finite processes.)

Is image-finiteness necessary?

Polymorphism

Type systems for parametric polymorphism constrain the power of observers by preventing them from directly manipulating data values whose types are abstract.

Polymorphic types can be used to **hide** certain implementation details of processes [cf: Mitchell and Plotkin's representation of data types in the λ-calculus]

Configurations have also **type substitution**

Open: A characterisation theorem for polymorphic type on the full π-calculus.

The combination of polymorphism and aliasing has surprising effects **(information leakage)**

Pierce and Sangiorgi, 97-00, JACM

$\boxed{\text{CONTENTS}}$

The π-calculus has 2 communication rules:

1. $a(b).\,Q \mid \boldsymbol{\nu} b\,(\overline{a}\,b \mid P) \xrightarrow{\tau} \boldsymbol{\nu} b\,(Q \mid P)$ $\quad b$ fresh. **[internal mobility]**

   A private name is being communicated.

   Only α-conversion is needed

2. $a(y).\,Q \mid \overline{a}\,b \xrightarrow{\tau} Q\{b/y\}$ **[external mobility]**

   A global name is being communicated.

   Full substitutions are needed.

External mobility gives aliasing
$\Rightarrow$ control aliasing by removing (certain forms of) external mobility.

$\Rightarrow$ **locality**, calculus ALπ

Disallow inputs of x in the body P of $a(x).\,P$

[cf: Join calculus; Merro and Sangiorgi 98-00]

$$\overbrace{\boldsymbol{\nu} a \; \boxed{\begin{array}{cc} \ldots a(y) \ldots & \\ & \mid \overline{b}\,a \\ a(z)\ldots \overline{a}\,v \ \ldots & \end{array}}}^{\boldsymbol{\nu} a\,(P \mid \overline{b}\,a)} \quad \mid \quad b(x).\overbrace{\boxed{\begin{array}{c} \ldots \overline{x}\,w \ \ldots \\ \\ \ldots \overline{c}\,x \ \ldots \overline{x}\,u \end{array}}}^{Q}$$

$$\downarrow \tau$$

$$\boldsymbol{\nu} a \; (\underbrace{\boxed{\begin{array}{c} \ldots a(y) \ldots \\ \\ a(z)\ldots \overline{a}\,v \ \ldots \end{array}}}_{P} \quad \mid \quad \underbrace{\boxed{\begin{array}{c} \ldots \overline{a}\,w \ \ldots \\ \\ \ldots \overline{c}\,a \ \ldots \overline{a}\,u \end{array}}}_{Q\{a/x\}})$$

You may think of P as an **object**, named a
(better: one P's methods is named a)

- an "object-oriented" programming style
  (a bit strong as a claim, maybe; yet in the π semantics of objects [Walker, Sangiorgi-Kleist, Huttel-Kleist-Nestmann-Merro] locality is essential)

- algebraic theory

- type inference à la ML (Vasconcelos and Honda, Fournet, Laneve, Maranget, and Rémy, Dal Zilio)

From external to internal mobility

$\approx_{AL\pi} \stackrel{\text{def}}{=}$ equality (contextual) in ALπ.

- $\overline{y}\, x \quad \approx_{AL\pi} \quad \boldsymbol{\nu} z\, (\overline{y}\, z \mid z \multimap x)$

  where z is fresh

  $z \multimap x \stackrel{\text{def}}{=} !z(a).\, \overline{x}\, a$ *(a link)*

- $\overline{y}\, x \quad \approx_{AL\pi} \quad \boldsymbol{\nu} z\, (\overline{y}\, z \mid z \twoheadrightarrow x)$

  where z is fresh

  $z \twoheadrightarrow x \stackrel{\text{def}}{=} !z(a).\, \boldsymbol{\nu} b\, (\overline{x}\, b \mid b \twoheadrightarrow a)$ *(a dynamic link)* replace free output, $\overline{a}\, b\,.\,P$ with bound output, $\overline{a}(b).\,P \stackrel{\text{def}}{=} \boldsymbol{\nu} b\,(\overline{a}\, b\,.\,P)$

$\Rightarrow$ Internal π (or Private π), πI [Sangiorgi, 95]

- Using these and other laws, dangerous forms of aliasing can be eliminated.
- The resulting processes can be manipulated using more standard techniques
  Examples: standard bisimilarity,
  simplified forms of typed bisimilarity where configurations are just pairs $\Gamma * P$

Going back to Example 2 (page 13):

$$
\begin{array}{rcl}
Q & \stackrel{\mathsf{def}}{=} & (\boldsymbol{\nu} x)\,(\overline{a}\,x \mid \overline{a}\,x \mid {!x}.\,R) \\
& \approx_{AL\pi} & (\boldsymbol{\nu} x, y)\,(\overline{a}\,x \mid \overline{a}\,y \mid {!\,y \multimap x} \mid {!x}.\,R) \\
& \approx_{AL\pi} & (\boldsymbol{\nu} x, x', y)\,(\overline{a}\,x \mid \overline{a}\,y \mid {!\,y \multimap x'} \mid {!x}.\,R \mid {!x'}.\,R) \\
& \simeq & (\boldsymbol{\nu} x\,,\,y)\,(\overline{a}\,x \mid \overline{a}\,y \mid {!x}.\,R \mid {!y}.\,R) \\
& \stackrel{\mathsf{def}}{=} & P
\end{array}
$$

CONTENTS

Some references

Polyadic pi-calculus: Milner 91.

i/o **types** (or **capability types**): Pierce and Sangiorgi 93, Boreale and Sangiorgi 98, Merro 98. **Extensions**: De Nicola, Ferrari and Pugliese 98, Sewell 98, Hennessy and Riely 98-9, Gay and Hole 99, Röckl 99.

Polymorphism: Turner 98, Pierce and Sangiorgi 97-99.

Applications to **object-oriented languages**: Kleist and Sangiorgi 98-9, Dal-Zilio 98, Hüttel, Kleist,Merro, Nestmann 99.

Applications also to **untyped languages**: π-calculus without matching (Boreale and Sangiorgi 98), spi-calculus (Boreale and De Nicola and Pugliese, 99).

Future?

- Typed synchronisation trees and categorical formulations
- Richer type systems
- Axiomatisations of typed behavioural equivalences
- A typed bisimilarity parametric on the type system
- Computer-aided assistance
- Larger proofs