Types for processes: from CONFER to FOCUS

Davide Sangiorgi

Focus Team, INRIA/University of Bologna

GGJJ 2011

From CONFER...

Esprit 6454 CONFER and WG 21836 CONFER II, 1992-2000

(CONcurrency and Functions: Evaluation and Reduction)

- JJ coordinator

  origins: a meeting in Sophia (JJ, GG, Boudol, Curien, ...)

- ‘create the basis for uniting functional and concurrent computation’
- really exciting projects

Sites, CONFER I :

INRIA-Rocquencourt (**JJ**)
INRIA-Sophia Antipolis (G. Boudol, **GG**)
University of Edinburgh (R. Milner)
Imperial College (S. Abramsky)
Ecole Normale Supérieure (P.-L. Curien)
CWI (J.-W. Klop)
SICS (J. Parrow)
ECRC (B. Thomsen)
Università di Pisa (U. Montanari)

Sites, CONFER II:

Università di Bologna (A. Asperti)
University of Sussex (M. Hennessy)
University of Edinburgh (S. Abramsky)
Università di Pisa (U. Montanari)
INRIA-Rocquencourt (J.-J. Lévy)
KTH (J. Parrow)
France Telecom (J.-F. Monin)
University of Cambridge (R. Milner)
CWI, Amsterdam (J.-W. Klop)
Ecole Normale Supérieure (P.-L. Curien)
ICL (Bent Thomsen)
INRIA-Sophia (G. Boudol)
Oxford University (D. Walker)

Sample of CONFER activities

- paths, optimal reductions, and optimal machines in lambda

  eg, the relationship among: [Asperti, Danos, Laneve, Regnier]
  Levy's family redexes (legal paths)
  Lamping-Gonthier's graph reduction algorithm (consistent paths)
  Girard's geometry of interaction (regular paths)

- game semantics and full abstraction
- Join and Pict languages
- theory of π-calculus (algebraic, behavioural, denotational, ...)
- action structures (anchestor of bigraphs)
- **types for processes**
- ....

Types for processes

Before CONFER, in Edinburgh, working on the π-calculus

[with R. Milner]

- **The expressive power**
  - **amazing**
  - **several expected behavioural equalities could not be proved**

    $\Rightarrow$ develpment of the behavioural and algebraic theory

    discouragingly, some equalities false in the available theory

Ordinary bisimulation (μ = an input, output or silent action):

$P \approx Q$ implies

– if $P \stackrel{\mu}{\Longrightarrow} P'$ then $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \approx Q'$

– if $Q \stackrel{\mu}{\Longrightarrow} Q'$ then $P \stackrel{\mu}{\Longrightarrow} P'$ and $P' \approx Q'$

- Data structures
- Chemical semantics

Example of failure of equality: inlining

Two factorial functions:

$$\mathrm{Fact} \stackrel{\text{def}}{=} !f(x,r).\ \texttt{if}\ x = 0\ \texttt{then}\ \overline{r}\,1\ \texttt{else}\ (\nu r')\,(\overline{f}\langle x-1, r'\rangle.\, r'(m).\,\overline{r}\langle x * m\rangle)$$

$$\begin{aligned}\mathrm{FactEff} \stackrel{\text{def}}{=}\ & !f(x,r).\\ & \texttt{if}\ x = 0\ \texttt{then}\ \overline{r}\,1\\ & \texttt{else}\ \ \texttt{if}\ x = 1\ \texttt{then}\ \overline{r}\,1\\ & \qquad \texttt{else}\ \ \texttt{if}\ x = 2\ \texttt{then}\ \overline{r}\,2\\ & \qquad\quad \texttt{else}\ \nu r'\,(\overline{f}\langle x-1, r'\rangle \mid r'(m).\,\overline{r}\langle x * m\rangle)\end{aligned}$$

$\mathrm{Fact} \not\approx \mathrm{FactEff}$

Another example: tail-call optimisations

More factorial functions:

$$S1 \stackrel{\text{def}}{=} !f(x,y,r).\ \texttt{if}\ x=0\ \texttt{then}\ \overline{r}\,y\ \texttt{else}\ (\nu r')\,(\overline{f}\langle x-1, x*y, r'\rangle.\, r'(m).\,\overline{r}\,m\,)$$

$$S2 \stackrel{\text{def}}{=} !f(x,y,r).\ \texttt{if}\ x=0\ \texttt{then}\ \overline{r}\,y\ \texttt{else}\ \overline{f}\langle x-1, x*y, r\rangle$$

$S1 \not\approx S2$

Example: call-by-value λ-calculus

Milner, R. 1990. *Functions as Processes*. Research Report 1154. INRIA.

$$
\begin{aligned}
[\lambda x.\, M]p &\stackrel{\text{def}}{=} p(x,q).\,[M]q \\
[x]p &\stackrel{\text{def}}{=} \nu q\,(\overline{x}\langle q\rangle.\,!q(v).\,\overline{p}\langle v\rangle) \\
[MN]p &\stackrel{\text{def}}{=} \nu q\,([M]q \mid (!q(v).\,\nu r\,([N]r \mid r(w).\,\overline{v}\langle w,p\rangle)))
\end{aligned}
$$

An optimisation: $[x]p \stackrel{\text{def}}{=} \overline{x}\langle p\rangle$

However, with the optimisation [DS92]

$[(\lambda x.\, M)\lambda y.\, N]p \not\approx [M\{^{\lambda y.\, N}/_{x}\}]p$ (different even on traces)

(The final MSCS version of Milner's paper was written after finding the counterexample and does not present the optmisation)

Before CONFER, in Edinburgh, working on the π-calculus

[with R. Milner]

- The expressive power
- **Data structures**
  - could be modeled
  - tedious in programming
  - run-time errors
- Chemical semantics

Structured data values and run-time errors

Beyond the monadic communications of the basic π-calculus

A first step: polyadicity

Problem: Run-time errors

$$\underline{\overline{x}\langle y, z\rangle}.\, P \mid \underline{x(h)}.\, Q \longrightarrow \texttt{wrong}$$

$$\overline{x}\langle y\rangle.\, \overline{y}\langle z_1, z_2\rangle.\, P \mid x(h).\, h(k).\, Q \longrightarrow \underline{\overline{y}\langle z_1, z_2\rangle}.\, P \mid \underline{y(k)}.\, (Q\{^{y}/_{k}\}) \longrightarrow \texttt{wrong}$$

Milner's sorting (ie, name arities)

as an ad-hoc notion to avoid run-time errors

no connection between arities and classical notions of types

Types, or: taming the untyped π-calculus

- CONFER: at Inria-Rocquencourt, in JJ's Team Para, 1992
  many visitors (DS, B. Pierce, C. Talcott, R. Statman...)

[Pierce & DS]

- a discipline on π-calculus names
- use types to make this explicit

Benefits:

- transplant well-studied type systems for the λ-calculus to the pi-calculus
- consequences on behaviour

(cf: the CONFER objectives, putting together functional and concurrency traditions)

A simple discipline: input and output usages

$\mathsf{b}(T)$	a channel, carrying values of type T
$\mathsf{r}(T)$	input (read) capability
$\mathsf{w}(T)$	output (write) capability

$\Rightarrow$ basis for **subtyping**

Example: $x : \mathsf{w}(\mathsf{w}(T)) \vdash (\nu y : \mathsf{b}(T))\ (\overline{x}\, y\, .\, !y(z).\, 0)$

(true, since $\mathsf{b}(T) < \mathsf{w}(T)$)

N.B.: The recipient of y cannot **interfere** with the existing input.

$\mathsf{r}(-)$	$\Rightarrow$	covariance
$\mathsf{w}(-)$	$\Rightarrow$	contravariance
$\mathsf{b}(-)$	$\Rightarrow$	invariance

Sample of rules for subtyping

Subtyping rules

$$\text{Sub-}\sharp\text{I}\ \frac{}{\sharp\, T \leq \mathsf{r}(T)} \qquad \text{Sub-}\sharp\text{O}\ \frac{}{\sharp\, T \leq \mathsf{w}(T)}$$

$$\text{Sub-II}\ \frac{S \leq T}{\mathsf{r}(S) \leq \mathsf{r}(T)} \qquad \text{Sub-OO}\ \frac{T \leq S}{\mathsf{w}(S) \leq \mathsf{w}(T)} \qquad \text{Sub-BB}\ \frac{T \leq S \qquad S \leq T}{\mathsf{b}(S) \leq \mathsf{b}(T)}$$

Typing rules

$$\text{T-InpS}\ \frac{\Gamma \vdash a : \mathsf{r}(S) \qquad \Gamma, x : S \vdash P}{\Gamma \vdash a(x).P}$$

$$\text{T-OutS}\ \frac{\Gamma \vdash a : \mathsf{w}(T) \qquad \Gamma \vdash w : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}\, w\, .P}$$

$$\text{Subsumption}\ \frac{\Gamma \vdash v : S \qquad S \leq T}{\Gamma \vdash v : T}$$

Examples typing with I/O

The factorial function

$$
\begin{array}{l}
f : \mathtt{b}(\langle \mathtt{Int}, \mathtt{w}(\mathtt{Int}) \rangle) \\
\vdash \\
!f(x, r).\quad \mathtt{if}\ x = 0 \quad \mathtt{then}\ \overline{r}\langle 1 \rangle \\
\qquad\qquad\qquad\qquad\ \ \mathtt{else}\ (\nu r')\, \overline{f}\langle x - 1, r' \rangle . r'(m). \overline{r}\langle x * m \rangle
\end{array}
$$

r/w **subtyping validates the subtyping rule for functions:**

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

If $\lambda x.\, M : S \to T$, then $p : \mathsf{w}(\mathsf{w}(\langle [S], \mathsf{w}([T])\rangle))$

which gives this translation on arrow types:

$$[S \to T] \stackrel{\text{def}}{=} \mathsf{w}(\langle [S], \mathsf{w}([T])\rangle)$$

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

The polyadic π-calculus: add the product types

Examples of other additions

- unions
- variants (cf: objects)
- polymorphism
- linearity

Behavioural effects of types

Before CONFER, in Edinburgh, working on the π-calculus

[with R. Milner]

- The expressive power
- Data structures
- **Chemical semantics**

The chemical abstract machine (CHAM)

Berry, G. and Boudol, G., *The Chemical Abstract Machine*, POPL 1990.

- Very influential paper in concurrency (1153 citations, Googlescholar)
- Revised by Milner into the **reduction semantics**
- Axioms for a **structural congruence** relation break a ‘geometrical’ vision of concurrency
- Simple **reduction rules** where redexes are subterms

Example of reduction semantics: the π-calculus

Structural congruence The smallest congruence $\equiv$ s.t.:

1. $P \equiv Q$ if P and Q are alpha convertible
2. $P \mid 0 \equiv P, \quad P \mid Q \equiv Q \mid P, \quad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
3. $\nu x\, 0 \equiv 0, \quad \nu x\, \nu y\, P \equiv \nu y\, \nu x\, P$
4. $\nu x\, (P \mid Q) \equiv (\nu x\, P) \mid Q$, if x not free in Q
5. $!P \equiv P \mid !P$

Reduction relation ($\longrightarrow$)

$$\text{R-INTER}\quad \frac{}{(\overline{x}\, y\, .\, P_1) \mid (x(z).\, P_2) \longrightarrow P_1 \mid P_2\{y/z\}}$$

$$\text{R-PAR}\quad \frac{P_1 \longrightarrow P_1'}{P_1 \mid P_2 \longrightarrow P_1' \mid P_2} \qquad \text{R-RES}\quad \frac{P \longrightarrow P'}{\nu z\, P \longrightarrow \nu z\, P'}$$

$$\text{R-STRUCT}\quad \frac{P_1 \equiv P_2 \longrightarrow P_2' \equiv P_1'}{P_1 \longrightarrow P_1'}$$

Essentially the same reduction semantics for the Higher-Order π-calculus (just make processes values in the syntax)

Contrast that with the labeled transition semantics

inp: $a(b).P \xrightarrow{a(b)} P$

out: $\overline{a}\,b.P \xrightarrow{\overline{a}\,b} P$

parL: $$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \; \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$$

comL: $$\frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\overline{a}\,b} Q'}{P \mid Q \xrightarrow{\tau} P'\{b/x\} \mid Q'}$$

closeL: $$\frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\nu b\,\overline{a}\,b} Q'}{P \mid Q \xrightarrow{\tau} \nu b\,(P'\{b/x\} \mid Q')} \; b \notin \mathrm{fn}(P)$$

res: $$\frac{P \xrightarrow{\alpha} P'}{\nu a\,P \xrightarrow{\alpha} \nu a\,P'} \; a \notin \mathrm{n}(\alpha)$$

open: $$\frac{P \xrightarrow{\overline{a}\,b} P'}{\nu b\,P \xrightarrow{\nu b\,\overline{a}\,b} P'} \; a \neq b$$

rep: $$\frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

alpha: $$\frac{P =_\alpha Q \qquad Q \xrightarrow{\alpha} Q'}{P \xrightarrow{\alpha} Q'}$$

... or with the LTS for the Higher-Order π-calculus :

$$\text{alp: } \frac{P' \xrightarrow{\mu} Q \qquad P \text{ and } P' \text{ are } \alpha\text{-convertible}}{P \xrightarrow{\mu} Q}$$

out: $\overline{x}\langle\widetilde{K}\rangle.\,P \xrightarrow{\overline{x}\langle\widetilde{K}\rangle} P$ inp: $x(\widetilde{U}).\,P \xrightarrow{x\langle\widetilde{K}\rangle} P\{\widetilde{K}/\widetilde{U}\},$ if $\widetilde{K} : \widetilde{U}$

$$\text{sum: } \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \qquad \text{par: } \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\; bn(\mu) \cap fn(Q) = \emptyset$$

$$\text{com: } \frac{P \xrightarrow{(\nu\widetilde{y})\overline{x}\langle\widetilde{K}\rangle} P' \quad Q \xrightarrow{x\langle\widetilde{K}\rangle} Q'}{P \mid Q \xrightarrow{\tau} \nu\widetilde{y}\,(P' \mid Q')}\; \widetilde{y} \cap fn(Q) = \emptyset$$

$$\text{match: } \frac{P \xrightarrow{\mu} P'}{[x = x]P \xrightarrow{\mu} P'} \qquad \text{const: } \frac{P\{\widetilde{K}/\widetilde{U}\} \xrightarrow{\mu} P'}{D\langle\widetilde{K}\rangle \xrightarrow{\mu} P'},\text{ if } D \stackrel{\text{def}}{=} (\widetilde{U})P$$

$$\text{res: } \frac{P \xrightarrow{\mu} P'}{\nu x\, P \xrightarrow{\mu} \nu x\, P'}\; x \notin n(\mu) \qquad \text{open: } \frac{P \xrightarrow{(\nu\widetilde{y})\overline{z}\langle\widetilde{K}\rangle} P'}{\nu x\, P \xrightarrow{(\nu x,\widetilde{y})\overline{z}\langle\widetilde{K}\rangle} P'}\; x \in fn(\widetilde{K}) - \widetilde{y}$$

Behavioural equivalences in the CHAM setting

[Milner & DS, 1992]

$P \Downarrow_a \stackrel{\text{def}}{=} P$ is capable of performing an input or output at a

barbed congruence

(**barbed bisimulation,** $\dot{\simeq}$): $P \dot{\simeq} Q$ implies:

1. if $P \Longrightarrow P'$ then $Q \Longrightarrow Q'$ and $P' \dot{\simeq} Q'$;
2. $\forall a$, if $P \Downarrow_a$ then also $Q \Downarrow_a$.

Barbed congruence $\stackrel{\text{def}}{=}$ the induced congruence.

- coinductive
- contextually defined, and therefore language dependent

Theorem In CCS and π-calculus, barbed congruence coincides with the ordinary bisimulation congruences

Types and the CHAM approach

Bisimilarity of processes

"match" each other **reductions**, in all contexts

rather than : match each other actions

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

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

The surrounding context must respect the types in Γ

Example: inlining on the factorial function

$$\mathrm{Fact} \stackrel{\text{def}}{=} !f(x,r).\ \ \texttt{if}\ x=0\ \ \texttt{then}\ \overline{r}\,1\ \ \texttt{else}\ (\nu r')\,(\overline{f}\langle x-1, r'\rangle.\, r'(m).\,\overline{r}\langle x*m\rangle)$$

$$\begin{aligned}\mathrm{FactEff} \stackrel{\text{def}}{=}\ & !f(x,r).\\ & \texttt{if}\ x=0\ \texttt{then}\ \overline{r}\,1\\ & \texttt{else}\ \ \texttt{if}\ x=1\ \texttt{then}\ \overline{r}\,1\\ & \qquad\texttt{else}\ \ \texttt{if}\ x=2\ \texttt{then}\ \overline{r}\,2\\ & \qquad\qquad\texttt{else}\ \nu r'\,(\overline{f}\langle x-1, r'\rangle \mid r'(m).\,\overline{r}\langle x*m\rangle)\end{aligned}$$

In the polyadic π-calculus:

$\mathrm{Fact} \not\approx_{\Gamma} \mathrm{FactEff}$ (for all legal Γ)

With r/w types:

$\mathrm{Fact} \approx_{f\leftarrow \mathrm{w}} \mathrm{FactEff}$ ($\approx_{f\leftarrow \mathrm{w}} \stackrel{\text{def}}{=}$ observers have only the output capability on f)

Example, tail-call on the factorial

$$S1 \stackrel{\text{def}}{=} !f(x,y,r).\quad \texttt{if } x = 0 \quad \texttt{then } \overline{r}\, y \quad \texttt{else } (\nu r')\,(\overline{f}\langle x-1, x*y, r'\rangle.\, r'(m).\, \overline{r}\, m\,)$$

$$S2 \stackrel{\text{def}}{=} !f(x,y,r).\quad \texttt{if } x = 0 \quad \texttt{then } \overline{r}\, y \quad \texttt{else } \overline{f}\langle x-1, x*y, r\rangle$$

$S1 \approx_{\Gamma} S2$ (with r/w types + linearity)

Another example: validity of the optimisation in the encoding of call-by-value λ-calculus

Types: other developments

- to control access rights (i.e., how information available in a certain site or process can be used by others),
- to control process mobility (i.e., in systems where processes and sites move, obtaining guarantees about where a process can be run),
- to control the usage of resources,
- to obtain systems of processes that are deadlock-free or livelock-free,
- to underpin new proof techniques,
- to improve the efficiency of verification algorithms,
- to guarantee integrity and confidentiality of data (as well as other security properties)
- “behavioural types” to specify interfaces and possible interactions among processes.
- session types

CHAM + Join (GG + JJ) : a functional semantics of processes

1. **CHAM** (reduction style)
2. **the Join calculus** (function application)

$$\texttt{def}\ spooler\langle printer\rangle \ \&\ job\langle file\rangle \ \triangleright\ \overline{printer}\langle file\rangle$$

Also:

- type inference á la ML
- a bridge towards Petri Nets
- a clear meaning of state
- distributed implementation

[Fournet, Gonthier, JJ, Maranget, Remy, Le Fessant, ...]

... to FOCUS

(a joint Research Team INRIA/Univ. Bologna)

An activity in FOCUS: liveness properties via types

- **safety** properties (absence of communication errors, interferences, deadlock, ...) intensively studied since the CONFER times
  - safety expressed in the subject reduction
- **liveness** properties
  - notoriously hard in concurrency, in mobile processes even harder
  - **termination**, **lock-freedom**

Termination in concurrency

- why? (big concurrent systems are supposed to run forever...)
- same reasons as in sequential languages
- applets, imported code
- sessions in SOC
- can be combined with other analysis

Lock-freedom

A **lock-free process**: certain selected input/output actions will eventually succeed (under fair scheduling)

$$!\,\mathtt{fib}(n, r).\ \ \mathtt{if}\ n < 2\ \mathtt{then}\ \overline{r}\langle n\rangle\ \mathtt{else}\ \nu r_1\,(\,\overline{\mathtt{fib}}\langle n-1, r_1\rangle \mid \overline{\mathtt{fib}}\langle n-2, r_1\rangle \mid r_1(x).\, r_1(y).\, \overline{r}\langle x+y\rangle\,)$$

$$\mid\ \overline{\mathtt{fib}}\langle 10, \mathtt{reply}\rangle\ \mid\ \mathtt{reply}(\mathtt{x}).\, \overline{\mathtt{print}}\langle x\rangle$$

↑ ↑

the selected prefixes

A Fibonacci server

1st rec. call

2nd rec. call

computes result

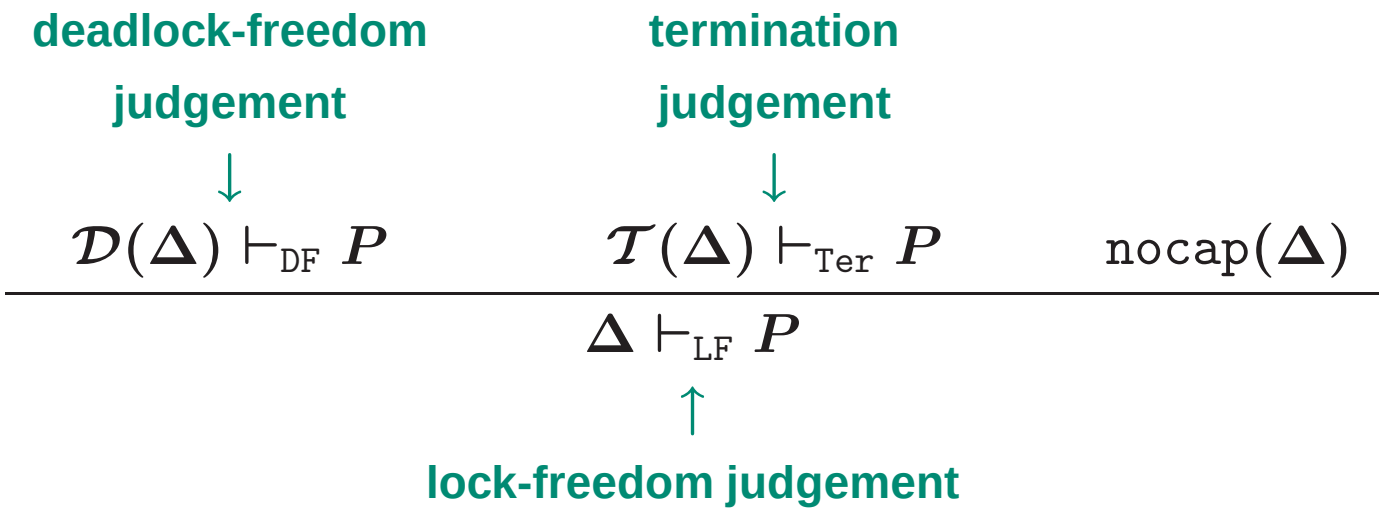
Most liveness properties can be turned into the lock-freedom property

- Access to a resource eventually granted?
- A server will eventually accept a client's request?
- If so, will the server eventually send a reply to the client?
- Can a process eventually acquire a lock?
- If so, will the process eventually release the lock?

Also: Information flow analysis, program slicing (dependency analysis), ...

Lock-freedom: Example of rules

[Kobayashi& DS, TOPLAS'10]



- **local** reasoning
- **implemented** in Kobayahsi's TyPiCal tool (types inferred)

Types for termination of processes

Separate functional and imperative names

- A functional name: (it is always available, unique continuation)

$$\nu a\,(!a(x).\,P \mid Q)$$

- a not in P
- only output capability on a in Q

Logical relation techniques

Theorem The simply-typed π-calculus, with all names functional, is terminating

- Other names: imperative

Term-rewriting techniques

– weight-based
– weight of a process eventually decreases under reduction

Example of hybrid system

a server using a bank service

clients using the servers s, s'

$$! \, s(\widetilde{x}) \cdots \overline{\mathrm{bank}} \cdots \mid ! \, \mathrm{bank}(\widetilde{y}) \cdots \mid ! \, s'(\widetilde{x}).\, \overline{s}\langle \widetilde{x} \rangle \mid \cdots \overline{s} \cdots \mid \cdots \overline{s'} \cdots$$

the bank

previous location for the server

(eg, migration, transparent upgrade)

– the server and the bank have a **state**

– s, s', bank are **functional**

To handle termination, we need the power of both **logical relation** (for the functional parts) and of **measure-based techniques** (for the imperative parts)

Idea of typing:

check that in $!a(x).P$:

- a **imperative** $\Rightarrow \mathtt{weigth}(a) > \mathtt{weigth}(P)$ (as before)
- a **functional** $\Rightarrow \mathtt{weigth}(a) \geq \mathtt{weigth}(P)$

- Functional names: imperative effects, but not at higher levels

  (no violations of the stratification on the imperative names)
- The measure assignment also captures divergent terms
- Termination holds if the purely functional sublanguage is terminating

  This can be established separately (eg, logical relations, but possibly other techiniques)

[Demangeon, Hirschkoff, DS ; CONCUR'10, FSEN'11]

Proof of termination

1. If P well typed, and $P \longrightarrow P'$ via an **imperative redex** then $\texttt{weight}(P') < \texttt{weight}(P)$
2. If $P \Uparrow$ then $P \Uparrow$ also if **functional names have priority**

   (a functional reduction does not affect the imperative redexes)
3. Hence P diverges in a computation where in any **two consecutive imperative reductions**

$$P \longrightarrow \ldots \longrightarrow P_i \longrightarrow \ldots \longrightarrow P_j \longrightarrow \ldots$$

we have $\texttt{weight}(P_i) > \texttt{weight}(P_j)$

4. Hence: from some point onwards, say P_k, **all reductions are functional**

   In P_k strip off all imperative prefixes (subterms $a(x).P'$, and $\overline{a}\langle v \rangle$, with a imperative)

   What remains is **a divergent functional process**

5. A **contradiction**, as the functional subcalculus is terminating (**logical relation or other techniques**)

Note: we separately apply
imperative and functional techniques for termination

Applications to other languages

A λ-calculus with references

- proving termination here using only logical relation is difficult
  our system subsumes **[Boudol 07, Amadio 09]**
- references treated using the term-rewriting techniques, akin to the imperative channels in π

Future work

- **Implicit computational complexity** in impure languages

  when there exist complexity bounds on the core functional language