Proof of the subject reduction property for a π-calculus in COQ

Loïc Henry-Gréard

N° 3698

May 1999

THÈME 1

ISSN 0249-6399

Proof of the subject reduction property for a π-calculus in COQ

Loïc Henry-Gréard

Thème 1 — Réseaux et systèmes
Projet MEIJE

Rapport de recherche n° 3698 — May 1999 — 18 pages

Abstract: This paper presents a method for coding π-calculus in the COQ proof assistant, in order to use this environment to formalize properties of the *pi*-calculus. This method consists in making a syntactic discrimination between free names (then called parameters) and bound names (then called variables) of the processes, so that implicit renamings of bound names are avoided in the substitution operation. This technique has been used by J.McKinna and R.Pollack in an extensive study of PTS [5]. We use this coding here to prove subject reduction property for a type system of a monadic π-calculus.

Key-words: π-calculus, COQ, subject reduction

Preuve de la préservation des types pour un π-calcul codé en COQ

Résumé : Ce rapport présente une formalisation du π-calcul dans le système d'aide à la preuve COQ, point de depart vers des preuves de propriétés du π-calcul. La méthode employée pour coder les noms consiste en la séparation syntaxique des noms libres (paramètres) et des noms liés (variables), en vue d'éviter les renommages implicites dans les substitutions, technique proposée et intensivement utilisee par J. McKinna et R. Pollack [5] pour la formalisation des PTS. La preuve effectuée ici est celle de la propriété de conservation des types (*subject reduction*) d'un π-calcul monadique typé.

Mots-clés : π-calcul, COQ, préservation des types

1 Introduction

1.1 A typed monadic π-calculus

The π-calculus has been introduced in 1989 by Robin Milner, Joachim Parrow and David Walker [8], as an extension of the CCS process algebra, and extends CCS's process synchronization expressivity with mobility of names, in order to have a complete formalization of concurrent processes. A precise description of π-calculus fundations can be found in [8]. A natural extension of the monadic π-calculus consists in considering name tuples as the values that can be passed between processes , instead of just names : the *polyadic* π-calculus. This version of π-calculus, described by R.Milner in [7], allows the introduction of a first notion of discipline on the use of names, since a reduction can be incorrect because of an arity mismatch. An other constraint on the use of names is introduced by D. Sangiorgi [9] [10] and concerns the *directionality* of names, that is, their ability to be used as emitters, receivers or both (or none). This name usage discipline is implemented by associating *types* to names, and by defining a predicate that states that the name that appears in a given process complies with the capacity they are given by an environment.

The proofs presented here as an application of the coding of the π-calculus is in this context the subject reduction property, which states that this correction predicate is preserved by the various kinds of reductions in a labelled transition semantics.

1.2 Coding name abstraction

The main difficulty in coding process algebras is in representing the binding of names in terms. In the π-calculus (of which the syntax is described below), name binding exists in two varieties:

Receivers: In the process $p(x).Q$, the name x is bound in the continuation process Q, waiting to be substituted by an other name before Q can be executed itself (Q is said to be "guarded" by $p(x)$). This abstraction is similar to λ-calculus abstraction (except for the guarding role of the input).

Scope restriction: In the process $(\nu x : t)Q$, the name x is only bound by a scope constraint. Its occurences in Q are not to be substituted, and the binder can be *extruded*, that is extended to processes that would come to receive x. This construction is what gives to the π-calculus all its expressivity. In fact, if the calculus is restricted to finite terms, that is, terms without the bang (!) operator, the scope restriction can be statically replaced by renamings of restricted names by fresh names, preserving the semantics:

$$a(x).P \mid (\nu x : t)(\bar{a}[x].Q \mid (\nu x : t')R_x)$$

being replaced by:

$$a(x).P \mid a[b].Q \mid R_c$$

The name restriction is needed to express that $!(\nu x : t)\bar{a}[x]$ always sends a fresh name to processes that listen to channel a.

The coding of these abstractions must meet several criterias: to allow a simple definition of the substitution operation avoiding variables captures, and thus implicit renamings, allow a natural coding of the semantics relation, and avoid to have to consider α-conversion in proofs, that is to allow the manipulation of terms that will represent their whole α-equivalence class.

A direct coding of names by a set with decidable equality, a solution used by T. Melham using HOL [6], implies a difficult definition of the substitution operation, even just for substituting names for names. $(\nu x)P\{y/z\}$ is simply defined for $x = z$ and for $x \neq z$, $y \neq x$, but implies a renaming for $x \neq z$, $y = x$.

In D. Hirschkoff [3], the coding uses de Bruijn indices, that is, replace bound names by pointers to their binder, coded by the number of binders one has to cross to find a given name's binder. This method allows to be sure to manipulate processes modulo α-equivalence, since two α-equivalent terms are not syntactically discernable in this coding, but formalizing the semantics relation becomes complicated, because manipulation of these indices do not match our intuitive representation of terms.

The method used here is a transposition to the π-calculus syntax and semantics to the one used in [5] to PTS. The idea is to distinguish by two different constructors, the free and bound names in a process.

1.3 COQ

COQ ([2]) is a proof assistant based on the Calculus of Inductive Constructions (CIC). It allows interactive manipulation of CIC terms representing proofs of assertions, and to the building of such terms by mean of tactics that reflect intuitive mathematical reasoning. COQ is also able to extract algorithm from constructive proofs of their specification. This feature will not be used here.

Outline of the paper. The proofs of the subject reduction property is very simple, after many technical lemmas are defined, giving COQ's user the basic tools to manipulate terms as one would naturally do on a paper and pencil proof. Part of these lemmas are given in section 4, as well as some steps of the proof of subject reducton property, for each kind of transition.

Before that, the reference semantics for the π-calculus (section 2.2), and the encoding method for this semantic (section 2.3) will be presented. A formal argument for the compliance of the coding with respect to the reference semantics will also be given. The type system will be introduced in section 3.

2 Semantics

This section presents the typed, monadic π-calculus, beginning by reference semantics, as used for informal reasoning, before the coding used in COQ for mechanized reasoning.

2.1 Syntax of a typed, monadic π-calculus

2.1.1 Reference syntax

The formalization work has been done for a classical monadic π-calculus, extended with an explicit type declaration for restricted names:

$$P \quad ::= \quad \bar{n}m.P \mid n(m).P \mid (\nu n : t)P \mid (P \mid Q) \mid !P \mid P + Q \mid [n = m]P \mid 0$$

where P, Q,... represent processes, n, m,... an infinite set of *names*, supposed enumerable. t, t', ...represent *types*, associated with names. The various constructors that are introduced in this monadic version of π-calculus are those of [8]. The INPUT operator $n(m).P$ expresses an agent that is waiting on a chanel named n, a piece of data that will be subsituted for m in its continuation P. The OUTPUT operator $\bar{n}m.P$ allows to transmit as data the name m on the chanel named n. The $(\nu n : t)P$, RESTRICTION operator, restricts the visibility of the name n to its body P. $!P$ denotes an process with infinite behavior (cf. [7]). We call Pr_I the set of processes syntactically defined this way.

2.1.2 Syntax in the COQ proof assistant

The system used here is an alternative to the use of de Bruijn indices [3]. The names are coded by a different constructor depending on the fact that they are in a bound position or in a free position in the process. Free names are called *parameters*, while bound names are called *variables*. We note Pr_P the set of terms of the π-calculus syntactically defined this way. The concrete type for Pr_P in COQ is called `proc`.

In the rest of the report, COQ expressions are translated in more commonly used mathematical notations. For exemple, a set of inference rules that caracterize a predicate will denode an inductively defined predicate in COQ. Parts of COQ code will be typed with a `typewriter-like font`.

Parameters are noted p, q, r, variables x, y, z, names n, m and processes P, Q, R. According to notations in [5], parameters and variables set are respectively noted `PP` and `VV`. The definition of names in COQ follows:

```
Inductive name : Set :=
  pname : PP -> name |
  vname : VV -> name.
```

A predicate:

```
fresh: PP -> proc -> Prop
```

is also defined, noted $p \notin \mathrm{P}(P)$, which states that a given parameter p does not appear in a given process P. Similarly, $x \notin \mathrm{V}(P)$ denotes the fact that the variable x does not appear in process P.

Decidability of the equality for the sets `PP` and `VV` is given by two axioms `PP_decidable` and `VV_decidable`, and the facts that one can always find fresh parameters satisifying a number of conditions (`PP` is an infinite set), by axioms in the file `exists.v`.

The definition of processes in COQ follows:

```
Inductive proc : Set :=
  nil : proc |
  inp : name -> VV -> proc -> proc |
  out : name -> name -> proc -> proc |
  par : proc -> proc -> proc |
  res : VV -> type -> proc -> proc |
  ban : proc -> proc |
  sum : proc -> proc -> proc |
  mat : name -> name -> proc -> proc.
```

For example, the process $(\nu x : t)(\bar{p}x \mid x(y))$ will be represented by the term:

```
(res x t (par (out (pname p) (vname x) nil)
              (inp (vname x) y nil)))
```

given of course that `x` and `y` are declared to have types `VV` and `p` to have type `PP`.

The main technical justification for this distinction between variables and parameters is that there is no need for renamings when a parameter is substituted for another in a process. A parameter cannot be captured, because binder can only point to variables, which have a different constructor.

We'll see however that translating the reference semantics leads to substituting variables for names, which in principle entails captures (for example in $(p(x).\bar{q}r)\{x/q\})$. This difficulty is linked to the OPEN/CLOSE system of the semantics, that *creates an abstraction* when the scope of a name is extruded. It does not appear in [5].

2.2 Reference Semantics

The terms are considered modulo α-equivalence, the binding operators being INPUT and RESTRICTION. The transition relation is considered as a relation between $Pr_I/\simeq_\alpha$ and $Pr_I/\simeq_\alpha$, parameterized by names.

The formalization is based on a structural operational semantics described in fig. 1, that defines a labeled transitions system. The symmetric couterparts of rules PARl, SUMl, COMl and CLOSEl have not been represented. The choice of a labeled transitions systems allows us to take into consideration not only internal reductions, but also the interaction of processes with their environment. We will have different subject reduction properties for each kind of transition in order to reflect the relationship between conditions on the typing environment and conditions on the context in which the process is running.

$$\text{INPUT} \quad \frac{}{x(y).X \xrightarrow{xz} X\{z/y\}}$$

$$\text{OUTPUT} \quad \frac{}{\bar{x}y.X \xrightarrow{\bar{x}y} X}$$

$$\text{COMl} \quad \frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}y} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\text{OPEN} \quad \frac{P \xrightarrow{\bar{x}y} P'}{(\nu y : t)P \xrightarrow{\bar{x}(y:t)} P'} \qquad x \neq y$$

$$\text{CLOSEl} \quad \frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}(y:t)} Q'}{P|Q \xrightarrow{\tau} (\nu y : t)(P'|Q')} \qquad y \notin \mathrm{fv}(P)$$

$$\text{RES} \quad \frac{P \xrightarrow{\mu} P'}{(\nu x : t)P \xrightarrow{\mu} (\nu x : t)P'} \qquad x \notin \mathrm{n}(\mu)$$

$$\text{PARl} \quad \frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q} \qquad \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\text{BANG} \quad \frac{!P|P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

$$\text{PLUSl} \quad \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}$$

$$\text{MATCH} \quad \frac{P \xrightarrow{\mu} P'}{[x = x]P \xrightarrow{\mu} P'}$$

Figure 1: *Semantics for a monadic π-calculus*

The actions characterizing the transitions are the same than in [8]:

$$\mu \quad ::= \quad xy \mid \bar{x}y \mid \bar{x}(y:t) \mid \tau$$

that is, respectively, *input*, *output*, *bound output* and the silent action. The *bound output* action represents the emission of a name that is new to its receiver, and thus the action must carry the type of the exported name, as well as the name that extends its scope by scope extrusion.

The side conditions of the inference rules deal with the scope extrusion problem. The $x \neq y$ condition on the OPEN rule denotes that a process cannot emit on a private channel, which makes for example the process $(\nu x : t)\bar{x}[y]$, equivalent to 0 in terms of behavior (bisimilarity). The condition on the CLOSE rules indicates that a process that receives an exported name receives a name that is new to it. This condition is intended to avoid a capture and can lead to a renaming. The side condition for the rule RES states that a name restricted to a process being "invisible for the outside", it cannot emit, nor receive, nor use this name as a communication channel. The side condition for the rule PAR avoids capture of a name in Q : a *bound output* action acts as a binder of the exported name on the resulting process. The PAR rule corresponds to a structural augmentation of the resulting process, where the names in Q can be binded.

2.3 Parameterized semantics

We build the labeled transitions relation in COQ from the reference semantics described above. The actions are the same, the names appearing in the actions are parameters:

```
Inductive act : Set :=
  aout : PP -> PP -> act |
  ainp : PP -> PP -> act |
  about : PP -> PP -> type -> act |
  tau : act.
```

We keep, as in [5], as an invariant the property `closed` of fully parameterized processes on both sides of the reduction arrow. This predicate checks that the process does not contain any variable in place of free names (as for example x in $p(y).P_y|\bar{x}q$). These processes represent the "well-formed" processes.

We build an inductive property in COQ, each constructor being described by an inference rule that is a variant of an rule of the reference semantics.

```
sem : proc -> act -> proc -> Prop
```

scope extrusion The OPEN rule replaces the variable whose binder has disappeared by a fresh parameter. When the scope restriction is created again (CLOSE rule), these parameters are replaced by the bound variable, checking that no capture is possible, that is, that the variable is "new" (side condition). Because of this side condition, we cannot have the property

$$\xrightarrow{\mu} \circ \simeq_\alpha = \xrightarrow{\mu}$$

any more, because for example:

$$\begin{array}{rcl} (\nu x : t)\bar{p}[x] \mid p(y).\bar{y}[q] \mid (\nu x : t)P & \xrightarrow{\tau} & (\nu y : t)(0 \mid \bar{y}[q] \mid (\nu x : t)P) \\ & \simeq_\alpha & (\nu x : t)(0 \mid \bar{x}[q] \mid (\nu x : t)P) \end{array}$$

while we cannot obtain:

$$(\nu x : t)\bar{p}[x] \mid p(y).\bar{y}[q] \mid (\nu x : t)P \xrightarrow{\tau} (\nu x : t)(0 \mid \bar{x}[q] \mid (\nu x : t)P)$$

restrictions The semantics of the π-calculus express by the RES rule the same thing than the ξ rule of the λ-calculus, that [5] deals with. It is based on a coding of abstraction by the set of possible subsitutions of the bound variable, explicit in [1]. We must note that this rule is

INPUT
$$\frac{}{p(x).P \xrightarrow{pq} P\{q/x\}}$$

OUTPUT
$$\frac{}{\bar{p}q.P \xrightarrow{\bar{p}q} P}$$

COMl
$$\frac{P \xrightarrow{pq} P' \qquad Q \xrightarrow{\bar{p}q} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

OPEN
$$\frac{P\{q/x\} \xrightarrow{\bar{p}q} P'}{(\nu x : t)P \xrightarrow{\bar{p}(q:t)} P'} \qquad \begin{array}{l} q \notin \mathrm{P}(P) \\ p \neq q \end{array}$$

CLOSEl
$$\frac{P \xrightarrow{pq} P' \qquad\qquad Q \xrightarrow{\bar{p}(r:t)} Q'}{P|Q \xrightarrow{\tau} (\nu x : t)(P'\{x/q\}|Q'\{x/r\})} \qquad \begin{array}{l} q \notin \mathrm{P}(P) \\ x \notin \mathrm{V}(P') \cup \mathrm{V}(Q') \end{array}$$

RES
$$\frac{\forall q \quad P\{q/x\} \xrightarrow{\mu} P'\{q/y\}}{(\nu x : t)P \xrightarrow{\mu} (\nu y : t)P'}$$

PARl
$$\frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q} \qquad \begin{array}{l} \mu = \bar{p}(q : t) \Rightarrow \\ q \notin \mathrm{P}(Q) \end{array}$$

BANG
$$\frac{!P|P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

PLUSl
$$\frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}$$

MATCH
$$\frac{P \xrightarrow{\mu} P'}{[p = p]P \xrightarrow{\mu} P'}$$

Figure 2: *parameterized monadic π-calculus*

the same than the P$\tilde{\text{R}}$1-BIND from [5], and that a transition relation, that is extensionally equivalent, can be obtained with a weaker induction principle by using the inference rule with a side condition instead of quantifying in the rule's premise:

$$\frac{P\{p/x\} \xrightarrow{\mu} P'\{p/y\}}{(\nu x : t)P \xrightarrow{\mu} (\nu y : t)P'} \quad p \notin \mathrm{P}(P) \wedge p \notin \mathrm{P}(\mu)$$

The formalization for the proof of this equivalence, done in [5] by induction on the length of the terms, has not been done in this study.

2.4 Adequacy of the encoding

The encoding of the π-calculus in COQ is justified by formalizing the correspondance between the reference semantics of the calculus and our inductive predicate. The goal of this formalization is to justify the choice of the side-conditions and quantifications on parameters that have been added in the inference rules. We denote by `NN` the set of names used in the informal notation (figure 1). The `PP`, `VV` and `NN` sets are supposed to be enumerable, and we choose b_P : `NN` $\rightarrow$ `PP` a bijection from the set of names to the set of parameters. Let $b_I = b_P^{-1}$.

We can now build a one-one mapping between the α-equivalence classes of the processes described by the usual syntax (Pr_I) and the processes described by the parameterized syntax (Pr_P). Let $T_P : Pr_I \rightarrow Pr_P$ be this correspondance. The μ actions involved in the semantics relation of figure 1 can also be translated in the parameterized actions of figure 2. This translation is noted $b_P(\mu)$.

To establish the translation function T_P, we first define a series $(x_n)_{n\in\mathbf{N}}$ of pairwise distinct elements of `VV`.

Then we define:

$$\begin{array}{rcl}
T_P(n(m).Q,N) & = & b_P(n)(x_N).(T_P(Q,N+1)\{x_N/b_P(m)\}) \\
T_P(\bar{n}m.Q,N) & = & \overline{b_P(n)}b_P(m).T_P(Q,N) \\
T_P((\nu n : t)Q,N) & = & (\nu x_N : t)T_P(Q,N+1)\{x_N/b_p(n)\} \\
T_P(P \mid Q,N) & = & T_P(P,N) \mid T_P(Q,N) \\
 & \vdots &
\end{array}$$

Lemma 2.4.1 *For every process $P \in Pr_I$, and for every integer N and M, we have the following:*

$$T_P(P,M) \simeq_\alpha T_P(P,N)$$

In order to define the inverse function of T_P, $T_I : Pr_P \rightarrow Pr_I$, we take a series $(p_N)_{N\in M}$ of pairwise distinct elements of `PP`. As the terms are syntactically defined, the set $Pars(P) = \{N \in \mathbf{N} \mid p_N \in \mathbf{fn}(P)\}$ is always finite. We note $N_0^P = \max Pars(P) + 1$, and $(n_N$ the series of $(b_I(p_N))$. We define the function T_I as follows:

$$\begin{array}{rcl}
T_I\,(p(x).Q,N) & = & b_I(p)(n_N).T_I(Q\{p_N/x\},N+1) \\
T_I\,(\bar{p}q.Q,N) & = & \overline{b_I(p)}b_I(q).T_I(Q,N) \\
T_I\,((\nu x : t)Q,N) & = & (\nu n_N : t)T_I(Q\{p_N/x\},N+1) \\
T_I\,(P \mid Q,N) & = & T_I(P,N) \mid T_I(Q,N) \\
 & \vdots &
\end{array}$$

Property 2.4.1 *If P, P' are two processes of Pr_I, and μ is an action, and if we have the $P \xrightarrow{\mu} P'$ transition in the usual semantics, then we can build a transition $Q \xrightarrow{b_P(\mu)} Q'$ in the parameterized semantics such that $Q \simeq_\alpha T_P(P,0)$ and $Q' \simeq_\alpha T_P(P',0)$.*

The proof is done by induction on the derivation of the transition in the semantics of figure 1. For the INPUT rule we have:

$$n(m).Q \xrightarrow{nk} Q\{k/m\}$$

Where $T_P(n(m).Q, 0) = b_P(n)(x_0).(T_P(Q,1)\{x_0/b_P(m)\})$, which, according to the semantics coded in COQ, reduces to:

$$b_P(n)(x_0).(T_P(Q,1)\{x_0/b_P(m)\}) \xrightarrow{b_P(n)b_P(k)} T_P(Q,1)\{x_0/b_P(m)\}\{b_P(k)/x_0\}.$$

Since x_0 does not appear in $T_P(Q,1)$, we have the right term of this derivation syntactically equal to $T_P(Q,1)\{b_P(k)/b_P(m)\}$, so with lemma 2.4.1, this term is also α-equivalent to the expected translation.

For the RES rule : if $P \xrightarrow{\mu} P'$, then by the induction hypothesis, there exists Q and $Q' \in Pr_P$ such that $Q \xrightarrow{b_P(\mu)} Q'$. Let m be the name whose scope is to be restricted to P, and $p = b_P(m)$, we have from the hypotheses $p \notin \mathrm{P}(b_P(\mu))$, so $Q\{q/p\} \xrightarrow{b_P(\mu)} Q'\{q/p\}$ for every parameter q. But $Q\{q/p\} = Q\{x_N/p\}\{q/x_N\}$ for every integer N big enough to avoid captures. The same reasoning applies to Q'. We can then apply the RES rule of figure 2:

$$\frac{\forall q \quad Q\{x_N/p\}\{q/x_N\} \xrightarrow{b_P(\mu)} Q\{x_N/p\}\{q/x_N\}}{(\nu x_N : t)Q\{x_N/p\} \xrightarrow{b_P(\mu)} (\nu x_N : t)Q'\{x_N/p\}}$$

However we have $T((\nu m : t)P, N) = (\nu x_N : t)T(P, N+1)\{x_N/p\}$, so as $Q \simeq_\alpha T(P, N+1)$, we have processes in the same α-class than their expected translations.

For the OPEN rule, we have from the hypotheses : $\dfrac{P \xrightarrow{\bar{n}m} P'}{(\nu m : t)P \xrightarrow{\bar{n}(m:t)} P'} \quad n \neq m.$ Let $p = b_P(n)$, and $q = b_P(m)$. From the induction hypothesis, we also have $Q \xrightarrow{\bar{p}q} Q'$ with Q and Q' α-convertible with the translations of P and P'. The hypothesis $p \neq q$ holds because b_P is a bijection. Taking N big enough to avoid captures, we have $Q\{x_N/q\}\{q/x_N\} \xrightarrow{\bar{p}q} Q'$. The condition $q \notin \mathrm{P}(Q\{x_N/q\})$ being always respected, we can apply the transition rule and obtain that $(\nu x_N : t)Q\{x_N/q\} \xrightarrow{\bar{p}(q:t)} Q'$, which allows us to conclude. For the CLOSE rule: if we have:

$$\frac{P \xrightarrow{nm} P' \qquad Q \xrightarrow{\bar{n}(m)} Q'}{P \mid Q \xrightarrow{\tau} (\nu m : t)(P' \mid Q')}$$

We note $p = b_P(n)$ and $q = b_P(m)$. By the induction hypothesis $R_P \xrightarrow{pq} R_{P'}$ and $R_Q \xrightarrow{\bar{p}(q)} R_{Q'}$. The hypothesis $q \notin \mathrm{P}(Q_P)$ holds because $m \notin \mathtt{fv}(P)$. Let N be big enough for the hypothesis $x_N \notin \mathrm{V}(R_{P'}) \cup \mathrm{V}(R_{Q'})$ to hold. We have, according to the CLOSE rule:

$$R_P \mid R_Q \xrightarrow{\tau} (\nu x_N : t)(R_{P'}\{x_N/q\} \mid R_{Q'}\{x_N/q\}).$$

We can then verify that both members of the transition are α-equivalent to the expected translations.

Property 2.4.2 *Whenever we have a transition $P \xrightarrow{\mu} P'$ of the parameterized semantics, there exist Q and Q' processes in Pr_I such that*

$$Q \xrightarrow{b_I(\mu)} Q')$$

is a valid transition according to the rules of figure 1 and $Q \simeq_\alpha T_I(P, N_0^P)$ and $Q' \simeq_\alpha T_I(P', N_0^{P'})$.

The proof is again done by induction, with for example the rule INPUT: Let $n = b_I(p)$, $k = b_I(q)$ and $N = \max N_0^{p(x).P} N_0^{P\{q/x\}}$. We have:

$$T_I(p(x).P, N) = n(n_N).T_I(P\{p_N/x\}, N+1)$$

Which reduces according to the rules of figure 2 to:

$$n(n_N).T_I(P\{p_N/x\}, N+1) \xrightarrow{\bar{n}k} T_I(P\{p_N/x\}, N+1)\{k/n_N\}.$$

But $T_I(P\{p_N/x\}, N+1)\{k/n_N\} = T_I(P\{p_N/x\}\{q/p_N\}, N+1)$, and also $p_N \notin \mathrm{P}(P)$, so we get:

$$T_I(p(x).P, N) \xrightarrow{\bar{n}k} T_I(P\{q/x\}, N+1)$$

For the RES rule: $\dfrac{\forall q \quad P\{q/x\} \xrightarrow{\mu} P'\{q/y\}}{(\nu x : t)P \xrightarrow{\mu} (\nu y : t)P'}$ By the induction hypothesis, for every parameter q, we have:

$$T_I(P\{q/x\}, N) \xrightarrow{b_I(\mu)} T_I(P'\{q/y\}, N),$$

so as $p_N \notin \mathrm{P}(P)$, for a q_0 such that $q_0 \notin \mathbf{n}(\mu)$ we have:

$$T_I(P\{p_N/x\}\{q_0/p_N\}, N) \xrightarrow{b_I(\mu)} T_I(P'\{p_N/y\}\{q_0/p_N\}, N).$$

If $n = b_I(q_0)$, then the hypothesis $n \notin \mathbf{n}(b_I(\mu))$ holds, so we have:

$$(\nu n : t)T_I(P\{p_N/x\}, N)\{n/n_N\} \xrightarrow{b_I(\mu)} (\nu n : t)T_I(P'\{p_N/y\}, N)\{n/n_N\}.$$

The left and right members of this relation are α-equivalent to the expected translations.

3 The type system

This section presents the specifications of the type system. The type system can be implemented in several ways, allowing or not to build for every process a suitable typing environment.

3.1 Types

The type system used in our monadic π-calculus carries for every name two informations:

- A directionnality information noted $\langle t \rangle$, and called the *capacity* of t. This information expresses the capacity of the name to be used as a channel for a communication. The R (Read), W (Write), B (Both) and N (Nil) capacitie form a small lattice with a subtyping relation noted $\leq_c$:

$$\begin{tikzcd} & \mathrm{N} \arrow[dl, dash] \arrow[dr, dash] & \\ \mathrm{R} \arrow[dr, dash] & & \mathrm{W} \arrow[dl, dash] \\ & \mathrm{B} & \end{tikzcd}$$

  The names that have a R capacity can be used as receivers, like p in $p(x).P$, those of capacity W can be emitters, as p in $\bar{p}[q].P$, those of capacity B can be both, and finaly those of capacity N cannot be receivers nor emitters, but can still appear in a term, like q in $\bar{p}[q].P$.

- An information about the type of the names that can transit by a given channel This type is denoted $[t]$. For our monadic π-calculus, $[t]$ represents itself a type since the only objects that can be transmitted are names. If we extend the calculus to polyadic π-calculus for example, $[t]$ will be a set of types denoting the constraints on the set of names simultaneously transmitted by a channel (including an arity information).

Given a certain types set (data of type `Set` in COQ), and functions $\langle \_ \rangle$ and $[\_]$, we define axioms on the subtyping relation, based on the covariance of the receiving types, and countervariance of the emitting types (fig. 4). We then suppose that the subtyping relation is transitive.

Note that whatever the representation of the types IS, these cannot form an inf-semi lattice, since for example two types t_1 and t_2 such that $\langle t_1 \rangle = \langle t_2 \rangle = B$ and $[t_1] \neq [t_2]$ do not have a minimum (it is impossible to find a type that is at the same time a subtype of one and of the other).

TypNil
$$\frac{}{\Gamma \vdash 0}$$

TypInp
$$\frac{\begin{array}{c}\forall q \notin \mathrm{P}(P) \\ \langle \Gamma(p) \rangle \leq_c I \qquad \Gamma \oplus (q \mapsto [\Gamma(p)]) \vdash P\{q/x\}\end{array}}{\Gamma \vdash p(x).P}$$

TypOut
$$\frac{\langle \Gamma(p) \rangle \leq_c O \qquad \Gamma(q) \leq [\Gamma(p)] \qquad \Gamma \vdash P}{\Gamma \vdash \bar{p}[q].P}$$

TypRes
$$\frac{\begin{array}{c}\forall q \notin \mathrm{P}(P) \\ \Gamma \oplus (q \mapsto t) \vdash P\{q/x\}\end{array}}{\Gamma \vdash (\nu x : t)P}$$

TypPar
$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P|Q}$$

TypBan
$$\frac{\Gamma \vdash P}{\Gamma \vdash !P}$$

TypPlus
$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P + Q}$$

TypMatch
$$\frac{\langle \Gamma(p) \rangle = \langle \Gamma(q) \rangle = B \qquad \Gamma \vdash P}{\Gamma \vdash [p = q]P}$$

Figure 3: *Typing rules for a monadic π-calculus*

STRefl $$\frac{}{t \leq t}$$

STNil $$\frac{\langle t' \rangle = N}{t \leq t'}$$

STRead $$\frac{\langle t_1 \rangle \leq_c R \quad \langle t_2 \rangle = R \quad [t_1] \leq [t_2]}{t_1 \leq t_2}$$

STWrite $$\frac{\langle t_1 \rangle \leq_c W \quad \langle t_2 \rangle = W \quad [t_2] \leq [t_1]}{t_1 \leq t_2}$$

Figure 4: *Subtyping rules for the names of π-calculus*

3.2 Environments

The processes will be typed (fig 3) or not in a environment that gives to each of their free names (that is of their parameters), a type expressing what they can transmit, as well as the direction in which they can transmit it. This predicate is noted $\Gamma \vdash P$ ([10]). These environments, noted Γ, Δ, are total functions form the set of parameters to the set of types.

Example: In Γ such that $\langle\Gamma(p)\rangle = B$, $\langle\Gamma(q)\rangle = N$, $\langle\Gamma(r)\rangle = N$, the processes $\bar{p}[q]$ and $p(x).\bar{x}[r]$ are correctly typed individually, but in the first case, it takes $\langle[\Gamma(p)]\rangle = N$ because of contravariance for typing emitting names, and in the second case $\langle[\Gamma(p)]\rangle \leq_c W$ because of the typing rule. Thus, the process $\bar{p}[q] \mid p(x).\bar{x}[r]$ cannot be correctly typed in an environment that forbids emitting to q. For detailed arguments about the correction of the directionnality information in typing, with respect to an "incorrect" usage of channels in processes, see [9].

We note $\oplus$ (binary operator, left associativity), the operation such that $\Gamma \oplus (p \mapsto t)$ is the function from parameters to types, that associates t to p and $\Gamma(q)$ to every $q \neq p$.

4 Proof for type preservation

The proof for the subject reduction properties are done very simply by an induction on the derivation of the labeled transitions, after we have defined a number of technical results, the most significant of which are given in the next subsection. Of course these results are themselves proven using a number of other lemmas. Thus, proving the subject reduction property for our formalization in COQ has been an opportunity to define many technical lemmas corresponding to intuitive facts a paper and pencil proof would not investigate, that provide a "toolbox" for future proofs.

4.1 Lemmas on substitutions

These lemmas, and those of the next section, are purely technical results and their proof in COQ is often trivial, by induction on the structure of the processes in most cases, the proof being then simplified by defining a similar lemma on the set of names, to factorize the verification of the property on process that include free names (*output*, *input* and *match*).

Lemma 4.1.1 (subs par after subs var)

$$\forall P, p, q, x \quad q \notin P(P) \Longrightarrow P\{q/x\}\{p/q\} = P\{p/x\}.$$

Lemma 4.1.2 (subs var after subs par)

$$\forall P, p, q, x \quad x \notin V(P) \Longrightarrow P\{x/p\}\{q/x\} = P\{q/p\}.$$

4.2 Lemmas on parameters

Lemma 4.2.1 (fresh after subs)

$$\forall P, p, q, x \quad p \notin \mathcal{P}(P) \wedge p \neq q \Longrightarrow p \notin \mathcal{P}(P\{q/x\}).$$

Lemma 4.2.2 (fresh before subs)

$$\forall P, p, q, x \quad p \notin \mathcal{P}(P\{q/x\}) \Longrightarrow p \notin \mathcal{P}(P).$$

Lemma 4.2.3 (listening on known name)

$$\forall P, P', p, q \quad P \xrightarrow{pq} P' \Longrightarrow p \in \mathcal{P}(P).$$

Similar properties are of couse true for output and bound output acions.

Lemma 4.2.4 (fresh after trans)

$$\forall P, \mu, P' \quad P \xrightarrow{\mu} P' \Longrightarrow \forall p\, p \notin \mathcal{P}(P) \wedge p \notin \mathcal{P}(\mu) \Longrightarrow p \notin \mathcal{P}(P').$$

4.3 Subtyping lemmas

These lemmas allows us to manipulate subtyping on environments, that is, the extension of the subtyping relation on parameters, to environments that are total functions of parameters to types:

$$\Gamma \leq_e \Delta \Leftrightarrow \forall p \quad \Gamma(p) \leq \Delta(p).$$

Lemma 4.3.1 (subtyping extension)

$$\forall \Gamma, \Delta, p, t_1, t_2 \quad \Delta \leq_e \Gamma \wedge t_1 \leq t_2 \Longrightarrow \Delta \oplus (p \mapsto t_1) \leq_e \Gamma \oplus (p \mapsto t_2).$$

Lemma 4.3.2 (subtyping)

$$\forall \Gamma, P \quad \Gamma \vdash P \Longrightarrow \forall \Delta\, \Delta \leq_e \Gamma \Longrightarrow \Delta \vdash P.$$

4.4 Swapping parameters

We introduce the "swap parameters" operator on processes, environments, names and parameters : `swap_proc`, `swap_env`, `swap_name` and `swap_par`. This operation is in all cases notes $X\{p \leftrightarrow q\}$ where X is a process, an environment, a name or a parameter. This operation consists in substituting to every occurenceof p, an occurence of q, and to every occurences of q, an occurence of p. In the case of environments, $\Gamma\{p \leftrightarrow q\}$ represents the function that associates $\Gamma(q)$ to p, $\Gamma(p)$ to q, and associates $\Gamma(r)$ to every r different of p and q .

A number of properties are proved for this operator, including:

Lemma 4.4.1 (typing after swap)

$$\forall \Gamma, P \quad \Gamma \vdash P \Longrightarrow \forall p, q \quad \Gamma\{p \leftrightarrow q\} \vdash P\{p \leftrightarrow q\}.$$

4.5 Typing Lemmas

Lemma 4.5.1 (addenv unused name)

$$\forall \Gamma, P \quad \Gamma \vdash P \Longrightarrow \forall t, q \notin P(P) \quad \Gamma \oplus (q \mapsto t) \vdash P.$$

Lemma 4.5.2 (redundant addenv)

$$\forall \Gamma, P \quad \Gamma \vdash P \Longrightarrow \forall \Delta, p, t\; p \notin P(P) \wedge \Gamma =_e \Delta \oplus (p \mapsto t) \Longrightarrow \Delta \vdash P.$$

Lemma 4.5.3 (type with other subs)

$$\forall \Gamma, P, p, q, x, t \quad p \notin P(P) \wedge q \notin P(P) \wedge \Gamma \oplus (p \mapsto t) \vdash P\{p/x\} \Longrightarrow \Gamma \oplus (q \mapsto t) \vdash P\{q/x\}.$$

Proof The proof of this lemma uses parameter swapping in terms. We consider the non-trivial case where $p \neq q$. From

$$\Gamma \oplus (p \mapsto t) \vdash P\{p/x\}$$

we deduce (4.4.1)

$$(\Gamma \oplus (p \mapsto t))\{p \leftrightarrow q\} \vdash P\{p/x\}\{p \leftrightarrow q\}.$$

As we have

$$(\Gamma \oplus (p \mapsto t))\{p \leftrightarrow q\} =_e \Gamma \oplus (q \mapsto t) \oplus (p \mapsto \Gamma(q))$$

We get the conclusion since p does not appear in $P\{q/x\}$.

Lemma 4.5.4 (subs typing)

$$\forall \Gamma, P \quad \Gamma \vdash P \Longrightarrow \forall p, q \quad \Gamma(p) \leq \Gamma(q) \Longrightarrow \Gamma \vdash P\{p/q\}$$

4.6 INPUT transitions

The property proved for INPUT transitions is the following:

$$P \xrightarrow{pq} P' \Longrightarrow \Gamma \vdash P \Longrightarrow \Gamma(q) \leq [\Gamma(p)] \Longrightarrow \Gamma \vdash P'.$$

The proof is done by induction on the transition relation. In each case, the typing predicate is inverted according to the syntax of the processes involved in a given transition rule.

4.6.1 The INPUT rule

We have $\Gamma \vdash p(x).Q$, so for every $s \notin \mathrm{P}(Q)$,

$$\Gamma \oplus (s \mapsto [\Gamma(p)]) \vdash Q\{s/x\}.$$

We also deduce from the hypothesis:

$$\Gamma \oplus (s \mapsto [\Gamma(p)])(q) \leq \Gamma \oplus (s \mapsto [\Gamma(p)])(s).$$

(To simplify the proof in COQ, we also make the assumption that s is different from q).

Thus we can substitute the paramater q for the parameter s in the typing relation (lemma 4.5.4):

$$\Gamma \oplus (s \mapsto [\Gamma(p)]) \vdash Q\{s/x\}\{q/s\}.$$

We know that s does not appear in the typed term, so we can omit to add it to the environment (lemma 4.5.2):

$$\Gamma \vdash Q\{s/x\}\{q/s\}.$$

The fact that $Q\{q/x\} = Q\{s/x\}\{q/s\}$ allows us to conclude (lemma 4.1.1).

4.6.2 The RES rule

The proof is the same for the RES rule whatever the transition kind is, so we will only describe it for the INPUT case.

If $\Gamma \vdash (\nu x : t)P$, we want to obtain for any $r \notin \mathrm{P}(P')$,

$$\Gamma \oplus (r \mapsto t) \vdash P'\{r/y\}.$$

Let $s \notin \mathrm{P}(P)$, with s different from p, q and r. $s \notin \mathrm{P}(P\{r/x\})$ (lemme 4.2.1) $\Longrightarrow s \notin \mathrm{P}(P'\{r/y\})$ (lemme 4.2.4), $\Longrightarrow s \notin \mathrm{P}(P')$ (lemme 4.2.2). So we can apply the lemma 4.5.3 and conclude if we have:

$$\Gamma \oplus (s \mapsto t) \vdash P'\{s/y\}$$

The induction hypothesis is:

$$\forall r \quad \Delta \vdash P\{r/x\} \Longrightarrow \Delta(q) \leq [\Delta(p)] \Longrightarrow \Delta \vdash P'\{r/y\}.$$

We replace in this proposition, Δ by $\Gamma \oplus (s \mapsto t)$ and r by s. The first condition is true because $\Gamma \vdash (\nu x : t)P$, the second condition is true because $s \neq p$ and $s \neq q$.

4.7 OUTPUT transitions

The result for OUTPUT transitions has two parts:

- A type preservation result:

$$P \xrightarrow{\bar{p}q} P' \Longrightarrow \Gamma \vdash P \Longrightarrow \Gamma \vdash P',$$

- A constraint on the environment that stems from the initial typing correction:

$$P \xrightarrow{\bar{p}q} P' \Longrightarrow \Gamma \vdash P \Longrightarrow \Gamma(q) \leq [\Gamma(p)].$$

4.8 Bound OUTPUT transitions

We proove as for OUTPUT the following properties:

- A type preservation property:

$$P \xrightarrow{\bar{p}(q:t)} P' \Longrightarrow \Gamma \vdash P \Longrightarrow \Gamma \oplus (q \mapsto t) \vdash P',$$

- a *bout* transition exports a type that is compatible with the typing environment:

$$P \xrightarrow{\bar{p}(q:t)} P' \Longrightarrow \Gamma \vdash P \Longrightarrow t \leq [\Gamma(p)].$$

4.8.1 The case of the OPEN rule

From the hypothesis:

$$\frac{P\{q/x\} \xrightarrow{\bar{p}q} P'}{(\nu x : t)P \xrightarrow{\bar{p}(q:t)} P'} \quad \text{for } q \notin \mathrm{P}(P) \text{ and } p \neq q. \quad \text{and also } \Gamma \vdash (\nu x : t)P.$$

Type preservation. The type preservation property directly follows from the similar property for OUTPUT transitions: $\Gamma \vdash (\nu x : t)P$, so for all r that does not appear in P, $\Gamma \oplus (r \mapsto t) \vdash P\{r/x\}$ (from the TypRes typing rule), so in particular for q. Thus by applying the subject reduction property for OUTPUT transitions to the $P\{q/x\} \xrightarrow{\bar{p}q} P'$ transition we obtain:

$$\Gamma \oplus (q \mapsto t) \vdash P'.$$

Exporting a correct type. We also use the similar property for OUTPUT:

$$(\Gamma \oplus (q \mapsto t))\,(q) \leq [(\Gamma \oplus (q \mapsto t))\,(p)]$$

As we have $p \neq q$, we can conclude that:

$$t \leq [\Gamma(p)].$$

4.8.2 PAR rule

We need our side condition on the inference rule: taking for example the PARl rule: $\dfrac{P \xrightarrow{\bar{p}(q:t)} P'}{P|Q \xrightarrow{\bar{p}(q:t)} P'|Q}$
with the supplementary condition: $q \notin \mathrm{P}(Q)$. The expected result is $\Gamma \oplus (q \mapsto t) \vdash P' \mid Q$.

- $\Gamma \oplus (q \mapsto t) \vdash P'$ is given directly by the induction hypothesis.
- $\Gamma \oplus (q \mapsto t) \vdash Q$ follows from the fact that $q \notin \mathrm{P}(Q)$, as well as from the hypothesis $\Gamma \vdash Q$: we modify in a typing environment the type of a name that is not used in the typed process (lemma 4.5.1).

4.9 Internal transitions

We prove the following property:

$$P \xrightarrow{\tau} P' \Longrightarrow \Gamma \vdash P \Longrightarrow \Gamma \vdash P'.$$

4.9.1 COM rule

Let's consider the COMl rule. The derivation of the transition ends with:
$$\dfrac{P \xrightarrow{pq} P' \qquad Q \xrightarrow{\bar{p}q} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$
So we have $\Gamma \vdash P \mid Q$, and we want to obtain $\Gamma \vdash P' \mid Q'$.

To proove $\Gamma \vdash P'$ from $\Gamma \vdash P$, we use the subject reduction property for the INPUT transitions. To validate the side condition $\Gamma(q) \leq [\Gamma(p)]$ of this property, we use the type correction property for the emission of a name by the transition $Q \xrightarrow{\bar{p}q} Q'$.

The fact the we can deduce $\Gamma \vdash Q'$ from $\Gamma \vdash Q$ simply stems from the subject reduction property for the transition $Q \xrightarrow{\bar{p}q} Q'$.

4.9.2 CLOSE rule

We take the CLOSEl rule as example. The derivation of the transition ends with:

$$\dfrac{P \xrightarrow{pq} P' \qquad\qquad Q \xrightarrow{\bar{p}(r:t)} Q'}{P|Q \xrightarrow{\tau} (\nu x : t)(P'\{x/q\}|Q'\{x/r\})} \quad .$$

We know on one hand that $q \notin \mathrm{P}(P)$, i.e. that P actually receives a name that is unknown to it, and on a second hand that $x \notin \mathrm{V}(P') \cup \mathrm{V}(Q')$, i.e. that there is no risk of capture of the new binding that is created by this rule.

From $Q \xrightarrow{\bar{p}(r:t)} Q'$ and $\Gamma \vdash Q$ we deduce $t \leq [\Gamma(p)]$; thus $(\Gamma \oplus (q \mapsto t))(q) \leq [\Gamma(p)]$. Also, q not appearing in P,
$$\Gamma \vdash P \Longrightarrow \Gamma \oplus (q \mapsto t) \vdash P$$
by the subject reduction property on the INPUTs:
$$\Gamma \oplus (q \mapsto t) \vdash P'.$$
we have $P' = P'\{x/q\}\{q/x\}$ since x does not appear in P'. So for every s free in $P'\{x/q\}$ (lemma 4.5.3) we have:
$$\Gamma \oplus (s \mapsto t) \vdash P'\{x/q\}\{s/x\}.$$

On the other hand, r not appearing in Q, we have $\Gamma \oplus (r \mapsto t) \vdash Q'$, by the subject reduction property on bounded emissions, so
$$\forall s \notin \mathrm{P}(Q'\{x/r\}) \quad \Gamma \oplus (s \mapsto t) \vdash Q'\{x/r\}\{s/x\}.$$
We can then conclude:
$$\Gamma \vdash (\nu x : t)(P'\{x/q\}|Q'\{x/r\}).$$

Conclusion

We have presented here a encoding of the π-calculus and used it to do a proof in the COQ theorem proover. The various lemmas that were needed to perform basic manipulations on π-terms can be reused for proofs of other properties. This coding has allowed a almost direct transcription of the usual semantics into COQ, and to code a proof that is close to the intuitive informal proof. Some initial mistakes in the transcription of the semantics have been uncovered during the proof process by encontering unsolvable or inconsistent goals, which is an argument in favor of mecanizing proofs and having formal methods to help believing in proofs on mathematical objects. But the COQ system also allows to exploit the prooving-is-computing principle and extract algorithms from proofs.

A natural extension of this work, besides extending the calculus to being able to transport name tuples instead of just name, would use this capability to generate, for example type checking algorithms for π-terms

Acknowledgmeents

This study has been done during a three month pre-doctorate internship in the INRIA Sophia-Antipolis, under the supervision of Joelle Despeyroux of the CROAP team, and Davide Sangiorgi of the MEIJE team. I would like to thank both of them for their disponibility and their advices on the π-calculus and COQ tips and tricks.

References

[1] Tom Melham Andrew Gordon. Five axioms of alpha-conversion. In J. Harrison J. von Wright, J. Grundy, editor, *TPHOL'96*, volume 1125, pages 173–190. Springer-Verlag, 196.

[2] JF. Filliâtre G.Huet P.Manoury C.Muñoz C.Murthy C.Parent C.Paulin-Mohring A. Saibi B.Werner C.Cornes, J.Courant. The coq proof assistant reference manual, 1997.

[3] Daniel Hirschkoff. A full formalisation of pi-calculus theory in the calculus of constructions, 1997.

[4] Daniel Hirschkoff. *Mise en oeuvre de preuves de bisimulation.* Phd thesis, École Nationale des Ponts et Chaussées (ENPC), January 1999. In French.

[5] James McKinna and Robert Pollack. Some Pure Type Sytems formalized. *To appear in a special issue on Formal Proof of the Journal of Automated Reasoning (JAR)*, 1999.

[6] T.F. Melham. A mechanized theory of the pi-calculus in hol. Technical Report 244, University of Cambridge Computer Laboratory, 1992.

[7] Robin Milner. The polyadic pi-calculus : a tutorial. Technical Report ECS-LFCS-91-180, LFCS, 1991.

[8] Walker Milner, Parrow. A calculus of mobile processes. Technical Report ECS-LFCS-89-85/6, LFCS, 1989.

[9] Davide Sangiorgi. Typing and subtyping for mobile processes. *Math. Struct. in Comp. Science*, 11, 1995.

[10] Davide Sangiorgi. An interpretation of typed objects into typed pi-calculus. INRIA RR-3000, 1996.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399