INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Permissive nominal terms

Gilles Dowek — Murdoch J. Gabbay — Dominic P. Mulligan

N° 6682

October 2008

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--6682--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SACLAY - ÎLE-DE-FRANCE

Permissive nominal terms

Gilles Dowek* , Murdoch J. Gabbay† , Dominic P. Mulligan†

Thème SYM — Systèmes symboliques
Équipe-Projet TypiCal

Rapport de recherche n° 6682 — October 2008 — 15 pages

Abstract: We present a simplified version of nominal terms with improved properties. Nominal terms are themselves a version of first-order terms, adapted to provide primitive support for names, binding, capturing substitution, and alpha-conversion. Nominal terms lack certain properties of first-order terms; it is always possible to 'choose a fresh variable symbol' for a first-order term and it is always possible to 'alpha-convert a bound variable symbol to a fresh symbol'. This is not the case for nominal terms. Permissive nominal terms preserve the flavour and the basic theory of nominal terms, including two levels of variable symbol, freshness, and permutation — but they recover the 'always fresh' and 'always alpha-rename' properties of first- and higher-order syntax, and they simplify the theory by eliding freshness contexts and by supporting a notion of term-unifier based on substitution alone, rather than the nominal terms' substitution and freshness conditions. No expressivity is lost moving to the permissive case.

Key-words: Unification algorithms, nominal techniques, permissive nominal terms

http://www.lix.polytechnique.fr/~dowek

http://www.gabbay.org.uk

http://www.macs.hw.ac.uk/~dpm8

* École polytechnique, 91128 Palaiseau Cedex, France
† School of Mathematics and Computer Science, Heriot-Watt University, Edinburgh, UK

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

Les termes nominaux permissifs

Résumé : On propose une amélioration et une simplification de la notion de terme nominal. Les termes nominaux étendent les termes du premier ordre avec des notions primitives de nom, de liaison, de substitution autorisant les captures et d'alpha-conversion. Hélas, un certain nombre de propriétés des termes du premier ordre ne sont pas vérifiées par ces termes nominaux : il est toujours possible de choisir une variable indépendante d'un terme du premier ordre et il est toujours possible d'alpha-convertir une variable liée dans un terme en un symbole indpendant de ce terme, mais ces propriétés ne sont pas vérifiées par les termes nominaux. Les termes nominaux permissifs gardent les idées fondamentales des termes nominaux, en particulier l'idée de distinguer deux niveaux de variables, ou les notions de variable indépendante d'un terme et de permutation, mais ils restaurent la possibilité de choisir une variable indépendante d'un terme et d'alpha-convertir une variable liée dans un terme en un symbole indépendant de ce terme, comme pour avec les termes du premier ordre ou d'ordre supérieur. Leur théorie se simplifie du fait de l'introduction de contextes d'indpendance, en particulier la notion d'unificateur pour les termes nominaux permissifs repose exclusivement sur la notion de substitution, alors qu'elle reposait, pour les termes nominaux, sur les notions de substitution et de conditions d'indépendance. L'expressivité de ces termes nominaux permissifs est toutefois la même que celles des termes nominaux originaux.

Mots-clés : Algorithme d'unification, technique nominale, terme nominal permissif

1 Introduction

Unification of first-order terms is a fundamental building-block of the theory of logic and computation. It combines expressivity with excellent mathematical properties. Yet some applications are closed to first-order unification, at least at the first glance, because they involve *name-abstraction* (binding) — sometimes, we just *want* in-built theoretical support for reasoning up to α-equivalence. Famously, getting this to work is not at all as easy as, and may be mathematically a far deeper issue than, is often supposed.

One solution is *higher-order unification*, which is unification up to $\alpha\beta\eta$-equivalence [Dow01]. However, higher-order terms are *very* expressive, and their unification is complex; it can be undecidable, decidable but without most general unifiers, or decidable and with most general unifiers subject to restrictions [Mil91]. What is more, the native notion of substitution — capture-avoiding substitution — does not precisely match the instantiation (capturing substitution) which often features in informal practice. For example we can set ourselves the informal unification problem

> "How can we make $\lambda x.\lambda y.(y\ \text{-}_1)$ equal with $\lambda x.\lambda x.(x\ \text{-}_2)$?"

with the intention being that the 'holes' -_1 and -_2 be substituted in a capturing manner.

Nominal unification was developed to extend first-order unification just enough that we can directly represent unification problems like that above, while preserving as much as possible of the flavour of first order unification. The slogan is 'ϵ away from informal practice'. Full details of nominal unification, including an extended essay on its advantages and disadvantages relative to higher-order unification and also with respect to de Bruijn encodings of binding in first-order terms [dB72], can be found in the original journal paper [UPG04] and subsequent published discussions [Pit02, BU07, Che05]. We do not intend to further reflect on that discussion but it may be worth noting that nominal terms have been made the basis of rewriting, logic-programming, and algebraic frameworks [FG07, CU03, GM07, Mat07, CP07] with good properties. This provides some pragmatic evidence that nominal terms can be put to good mathematical use.

This paper proposes a (deceptively innocuous) change to nominal terms which we call *permissive nominal terms*. These preserve the flavour of nominal terms but they are also significantly different in many ways; we discuss how and why shortly. First, we briefly reprise nominal terms so that we can compare them easily with the permissive version. We need some by now standard definitions [UPG04]:

Definition 1 Fix a countably infinite set $\mathbb{A}$ of **atoms**. We use a **permutative convention** that $a, b, c, \ldots$ range *permutatively* over atoms. That is, for example 'a and b' means 'a pair of any two *distinct* atoms'.

Fix a countably infinite set of **unknowns**. $X, Y, Z, \ldots$ will range permutatively over unknowns.

Fix a set of **term-formers**. $\mathsf{f}, \mathsf{g}, \mathsf{h}$ will range permutatively over term-formers.

We assume that atoms, unknowns, and term-formers are all disjoint.

Definition 2 Call a bijection on atoms π a **permutation** when π is finitely-supported. That is, for all but finitely many atoms it is the case that $\pi(a) = a$.

$\pi, \pi', \tau, \ldots$ will range over permutations.

Nominal terms [UPG04] are inductively defined by:

$$r, s, \ldots ::= a \mid \pi \cdot X \mid [a]r \mid \mathsf{f}(r, \ldots, r).$$

Atoms represent *variable symbols*, term-formers represent *functions*, unknowns represent *meta-variables*, and abstraction $[a]r$ represents *binding*. For example:

- If app is a term-former then the nominal term $\mathsf{app}(a, b)$ can represent the λ-calculus expression xy (x applied to y).
- If lam is a term-former then the nominal term $\mathsf{app}(\mathsf{lam}([a]a), b)$ can represent the λ-calculus expression $(\lambda x.x)y$.

$\mathsf{app}(\mathsf{lam}([a]a), b)$ and b are not equal nominal terms, any more than $(\lambda x.x)y$ and y are identical λ-calculus expressions.[1]

Nominal terms have a sophisticated theory of α-equivalence $=_\alpha$. For example

$$b\#X \vdash [a]X =_\alpha [b](b\ a) \cdot X$$

is valid. $b\#X$ is a *freshness side-condition*; $b\#X$ represents the informal judgement that $y \notin fv(r)$. $(b\ a)$ is a *permutation* meaning 'map a to b and b to a'. The equality above represents the informal judgement[2]

'if $y \notin fv(t)$ then $\lambda x.t$ is α-equivalent with $\lambda y.[y/x]t$'.

The implicit quantification over all t is reflected in the use of X, which can be instantiated (a capturing substitution). If we set X to be a — the freshness side-condition $b\#a$ is indeed valid — then we obtain $[a]a =_\alpha [b]b$, which represents the correct informal judgement

'$\lambda x.x$ is α-equivalent with $\lambda y.y$'.

A problem:

- Freshness contexts are not fixed, and this causes complexity in proofs (we always have to prove things about a term-in-context, rather than just about a term) and complicates the theory of equality between terms ($[a]X =_\alpha [b](b\ a) \cdot X$ is true if $b\#X$ is in the freshness context, and false otherwise). It would be simpler to fix freshness information once and for all at the beginning of a paper, then never change it — or the notion of α-equality between terms — again.
- Nominal terms lack some 'minor' properties of first- and higher-order terms; we can *always* pick a fresh variable symbol for a term, and we can *always* α-rename a bound variable symbol in a term, to be fresh. For example, if we have a λ-term expression $\lambda x.t$ then we can always choose a fresh y, and we can always α-rename to $\lambda y.[y/x]t$.

[1] For fully general theories of computation and equality on nominal terms, see nominal rewriting and nominal algebra [FG07, GM07, Mat07, Gab08].

[2] We use examples from the λ-calculus only as a paradigmatic instance of the general theory which nominal terms model. Note that $[y/x]$ is a *renaming*, not a permutation (permutations are bijective, renamings are not). The case for prefering permutations to develop a theory of α-equivalence has been made in detail in the literature, for example in [Che05].

> In nominal terms this is not so. $b\#X$ is neither true nor false; it is an assumption about what X may be instantiated to. We cannot obtain a b such that $b\#X$ unless we make a freshness assumption that $b\#X$. Likewise, we cannot immediately α-convert a in $[a]X$ until we obtain some $b\#X$ from the freshness context; in the empty freshness context we are stuck.[3]

Our solution; a sketch of permissive nominal terms:

We fix freshness conditions once and for all (Definition 3) then elide them as a *permissions sorting condition* on unknowns (Definitions 4 and 5). Thus unknowns take the form of X^S where S a set of atoms is the *permissions sort* of X. Permissions sorts are not cofinite sets of atoms, they are sets of 'every other atom' in a suitable sense (Definition 3). A substitution (a map instantiating unknowns) is only admitted if it instantiates each X^S to a term whose free variable symbols are in S (Definition 17). As a result we can always choose a fresh atom for a term, we can always α-convert, and α-equivalence is inherent rather than depending on a freshness context (Definition 9, Lemma 10, and Corollary 11). Furthermore the notion of solution to a unification problem is based on substitutions (Definition 29) — the corresponding notion for nominal terms is based on substitution-and-freshness [UPG04, Definition 3.1] and is more complex.

In this paper we lay down definitions and basic properties, going as far as to verify that if a unifier exists then a most general (*principal*) unifier exists, which is also the key result of [UPG04]. There seems no reason why a theory of rewriting should not also be developed similarly to [FG07] and the authors have begun to apply these permissive ideas in some extended case studies, using permissive nominal terms as the basis of novel logics and λ-calculi, taking full advantage of their greater closeness to traditional syntax. The indications are promising.

A disadvantage we see to permissive nominal terms is that *permissions sorts* use the set *comb* of 'every other atom' (Definition 3), and this is incompatible with the finite-support property of nominal sets [GP99, Definition 3.1]. This is not a paradox; in this paper we build permissive nominal terms in the meta-level, where we can have any subsets of atoms that we like. However it is unfortunate because, unlike nominal terms, permissive nominal terms can be directly qotiented by α-equivalence just as for first- and higher-order terms, and so it would be satisfying and perhaps useful to apply the Gabbay-Pitts model of abstract syntax up to α-equivalence [GP01]. The second author developed an infinitary generalisation of nominal sets [Gab07] which can serve to model permissive nominal terms, and also in fact it is possible to encode permissive nominal terms in (ungeneralised) nominal sets with a very slight effort; these considerations may be included in a journal version of this paper. In this conference paper we seek to make the definitions available, and make a case that this different putting-together of the ideas behind nominal terms works surprisingly well and merits further investigation.

Related work

[3] We encountered this before and dealt with it in two ways. In [GM07, Figure 2, axiom (**fr**)] and in [GM08, e.g. Lemma 25 and Theorem 33] we 'freshly extend' the freshness context. Alternatively we can extend nominal terms syntax with an explicit fresh names construct as in for example [GL08, FG05].

We have discussed the connection with nominal terms [UPG04] and in the body of the paper we will draw out similarities and differences as we come to them. This work bears the stamp of nominal techniques; two levels of variable symbol (atoms at level 1 and unknowns at level 2), a capturing substitution (instantiation), and α-equivalence managed by permutations and freshness assertions. Like (permissive) nominal terms, higher-order patterns have many of the good properties of first-order terms [Mil91]. A significant difference is that the notion of unification is based squarely on capture-avoiding substitution rather than the (permissive) nominal term capturing substitution and this gives the system's behaviour a very different flavour. The interested reader is also referred to a recent paper [LV08] which goes some way to teasing out formal connections between these two approaches.

Hamana's β_0 unification of λ-terms with holes adds a capturing substitution [Ham01]. Level 2 variables (which are instantiated) are annotated with level 1 variable symbols that *may* appear in them; permissive nominal terms move in this direction in the sense that permissions sorts also describe which level 1 variable symbols (we call them atoms in this paper) may appear in them, though with our permissions sorts there are infinitely many that may, and infinitely many that may not. Another significant difference is that the treatment of α-equivalence in Hamana's system is not nominal (not based on permutations) and unlike our systems, Hamana's does not have most general unifiers. Similarly Qu-Prolog [NR96] adds level 2 variables, but does not manage α-conversion in nominal style and also, for better or for worse, the system is more ambitious in what it expresses and thus loses mathematical properties (unification is semi-decidable, most general unifiers need not exist).

We refer the interested reader to extended discussions in [UPG04, Section 4]. Permissive nominal terms certainly seem closer to first- and higher-order terms than nominal terms, but they have their own character and are not a special case of previous work.

2 Permissive nominal terms

Definition 3 Fix a bijection γ between natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ and $\mathbb{A}$. Define[4]

$$comb = \{\gamma(2i) \mid i \in \mathbb{N}\} \subseteq \mathbb{A}.$$

Intuitively, *comb* is a set of 'every other atom'. Define *Permissions* inductively by:

$$\frac{}{comb \in Permissions} \qquad \frac{S \in Permissions}{S \setminus \{a\} \in Permissions} \qquad \frac{S \in Permissions}{S \cup \{a\} \in Permissions}$$

Thus, *Permissions* is the collection of sets of atoms that differ finitely from *comb*. We can choose *Permissions* to be one of many other subsets of the powerset of all atoms; our choice in Definition 3 serves our needs. For Lemma 10 and Corollary 11 to work, it is important that every $S \in Permissions$ be such that $\mathbb{A} \setminus S$ is infinite.

[4] Nominal sets [GP01] famously only allow elements with finite support. However, we are working at the meta-level, where we can talk about any subset or function that we wish.

Definition 4 For each permissions set $S \in \mathit{Permissions}$ fix a countably infinite set of **unknowns** of permissions sort S. We assume unknowns of distinct permissions sort are distinct. We let X^S, Y^S, Z^S range permutatively over unknowns of sort S. That is, 'X^S and Y^S' means 'two distinct unknowns of sort S'.

Note that if $S, S' \in \mathit{Permissions}$ and $S \neq S'$ then X^S and $X^{S'}$ are distinct unknowns (with misleadingly similar names). If the superscript of an unknown is unimportant or understood then we drop it.

Definition 5 Let **permissive nominal terms** be defined by:

$$r, s, t, \ldots ::= a \mid \pi \cdot X^S \mid [a]r \mid \mathsf{f}(r, \ldots, r).$$

a denotes any atom, S any permissions set, X^S any unknown, and f any term-former. We write $\equiv$ for syntactic identity, that is, $r \equiv s$ when r and s are identical terms.

Remark 6 Note in passing that if we pick a bijection between $\mathbb{A}$ and *comb* (we can do this because both are countably infinite sets) then we can translate nominal terms in freshness contexts to the permissive framework, bijectively in a suitable formal sense. We omit details in this conference paper, but the intuition is this: we can view nominal terms as a proper subsystem of permissive nominal terms, modulo some minor technical wizardry to help things match up.

Definition 7 Let r be a term. Define the **free atoms** $fa(r)$ by:

$$fa(a) = \{a\} \qquad fa(\pi \cdot X^S) = \{\pi(a) \mid a \in S\}$$
$$fa([a]r) = fa(r) \setminus \{a\} \qquad fa(\mathsf{f}(r_1, \ldots, r_n)) = fa(r_1) \cup \cdots \cup fa(r_n)$$

Definition 8 Define a **permutation action** on terms by:

$$\pi \cdot a \equiv \pi(a) \qquad \pi \cdot (\pi' \cdot X^S) \equiv (\pi \circ \pi') \cdot X^S$$
$$\pi \cdot (\mathsf{f}(r_1, \ldots, r_n)) \equiv \mathsf{f}(\pi \cdot r_1, \ldots, \pi \cdot r_n) \qquad \pi \cdot [a]r \equiv [\pi(a)](\pi \cdot r)$$

Write id for the **identity** permutation such that $id(a) = a$ always. Write $(a\ b)\cdot r$ for the **swapping** permutation that swaps a and b in the term r. If $S \subseteq \mathbb{A}$ write $\pi \cdot S$ for $\{\pi(a) \mid a \in S\}$.

Definition 9 If π and π' are two permutations define $ds(\pi, \pi')$ their **difference set** by

$$ds(\pi, \pi') = \{a \in \mathbb{A} \mid \pi(a) \neq \pi'(a)\}.$$

We let α**-equivalence** $=_\alpha$ be inductively defined by:

$$\frac{r =_\alpha s}{[a]r =_\alpha [a]s}\ (=_\alpha[]\mathbf{aa}) \qquad \frac{(b\ a) \cdot r =_\alpha s \qquad (b \notin fa(r))}{[a]r =_\alpha [b]s}\ (=_\alpha[]\mathbf{ab})$$

$$\frac{}{a =_\alpha a}\ (=_\alpha\mathbf{aa}) \qquad \frac{r_1 =_\alpha s_1 \quad \cdots \quad r_n =_\alpha s_n}{\mathsf{f}(r_1, \ldots, r_n) =_\alpha \mathsf{f}(s_1, \ldots, s_n)}\ (=_\alpha\mathsf{f})$$

$$\frac{(ds(\pi, \pi') \cap S = \emptyset)}{\pi \cdot X^S =_\alpha \pi' \cdot X^S}\ (=_\alpha\mathbf{X})$$

The validity of $r =_\alpha s$ is independent of freshness assumptions, which are now built in to the permissions sorts of the unknowns in r and s. This has quite a different flavour from nominal terms' α-equivalence which is always in a context of freshness assumptions [UPG04, Figure 2]. Lemma 10 and Corollary 11 are properties that 'ordinary syntax' has, that nominal terms do not have, and that permissive nominal terms recover. Lemma 10 states 'we can always pick a fresh variable'. Corollary 11 states 'we can always α-rename'.

Lemma 10 *For any r, there exist infinitely many b such that $b \notin fa(r)$.*

Proof. By an easy induction on r. We consider only the case $r \equiv X^S$. $fa(\pi \cdot X^S) = \pi \cdot S$. There are infinitely many b such that such that $b \notin \pi \cdot S$.

Corollary 11 *For any r and a there exists infinitely many fresh b (so $b \notin fa(r)$) such that for some s, $[a]r =_\alpha [b]s$.*

Proof. By Lemma 10 and $(=_\alpha[]\mathbf{ab})$.

We conclude this section with results whose proofs are essentially identical to those of corresponding properties for nominal terms [UPG04, e.g. Lemma 2.8 and Theorem 2.11]. That is, our changes do not affect basic results about nominal terms:

Lemma 12 $\pi' \cdot (\pi \cdot r) \equiv (\pi' \circ \pi) \cdot r$

Lemma 13 $\pi \cdot fa(r) = fa(\pi \cdot r)$.

Lemma 14 *If $r =_\alpha s$ then $\pi \cdot r =_\alpha \pi \cdot s$.*

Lemma 15 *If $ds(\pi, \pi') \cap fa(r) = \emptyset$ then $\pi \cdot r =_\alpha \pi' \cdot r$.*

Lemma 16 *$=_\alpha$ is transitive, reflexive, and symmetric.*

3 Substitutions

Definition 17 A **substitution** σ is a finitely-supported mapping from unknowns to terms such that

$$\text{for every } X^S \text{ it is the case that } fa(\sigma(X^S)) \subseteq S.$$

Finitely-supported means there is a finite set of unknowns D such that if $X \notin D$ then $\sigma(X) = id \cdot X$.

σ will range over substitutions. If $fa(t) \subseteq S$ we write $[X^S \mapsto t]$ for the substitution mapping X to t and any other Y to $id \cdot Y$. Similarly if $fa(t) \subseteq S$ and $fa(u) \subseteq T$ then we write $[X^S \mapsto t, Y^T \mapsto u]$ for the substitution mapping X^S to t, Y^T to u, and any other Z to $id \cdot Z$, and so on. We will also write id for the **identity substitution**, the substitution mapping all unknowns to themselves. It will always be clear from context whether id refers to the identity substitution or the identity permutation.

For example if $a \in comb$ then $[X^{comb} \mapsto a]$ is a substitution and $[X^{comb \setminus \{a\}} \mapsto a]$ is not.

Definition 18 Define a **substitution action** inductively by:

$$a\sigma \equiv a \qquad \mathsf{f}(r_1,\ldots,r_n)\sigma \equiv \mathsf{f}(r_1\sigma,\ldots,r_n\sigma) \qquad ([a]r)\sigma \equiv [a](r\sigma)$$
$$(\pi \cdot X^S)\sigma \equiv \pi \cdot \sigma(X^S)$$

Lemma 19 $\pi \cdot (r\sigma) \equiv (\pi \cdot r)\sigma$.

Proof. By a routine induction on r like that in [GM08].

Distinctly from the case of nominal terms, a substitution only *exists* if it respects permissions sorts and α-equivalence:

Theorem 20 *1.* $fa(r\sigma) \subseteq fa(r)$.

2. $r =_\alpha s$ *implies* $r\sigma =_\alpha s\sigma$.

Proof. The first part is proved by a routine induction on r. We consider only the case that $r \equiv \pi \cdot X^S$. By definition $fa(\pi \cdot X^S) = \pi \cdot S$. By assumption in Definition 17, since σ is a substitution we know that $fa(\sigma(X^S)) \subseteq S$. Using Lemma 13, it follows that $fa(\pi \cdot \sigma(X^S)) \subseteq \pi \cdot S$ and $\pi \cdot S \subseteq \pi \cdot S$ always.

The second part is by a routine induction on the derivation of $r =_\alpha s$. We use Lemma 19, and we use part 1 of this result and Lemma 15 for the case of $(=_\alpha \mathbf{X})$, and part 1 for the case of $(=_\alpha [] \mathbf{ab})$.

Definition 21 Define $fV(r)$ inductively by:

$$fV(a) = \emptyset \qquad fV(\pi \cdot X^S) = \{X\}$$
$$fV([a]r) = fV(r) \qquad fV(\mathsf{f}(r_1,\ldots,r_n)) = fV(r_1) \cup \cdots \cup fV(r_n)$$

Theorem 22 *If* $X\sigma_1 =_\alpha X\sigma_2$ *for all* $X \in fV(r)$*, then* $r\sigma_1 =_\alpha r\sigma_2$.

Proof. By an easy induction on r. The case of $r \equiv \pi \cdot X$ uses Lemma 14.

4 Permissive nominal unification problems

Definition 23 An **equality predicate** is a pair $r \ {}_?{=}_?\ s$. An **freshness predicate** is a pair $a \#_? r$. A **problem** Pr is a finite set of equality and freshness predicates.

We may omit set brackets when we write problems down. For example, $b \#_? X^{comb}$ is a problem; $[a]X^{comb} \ {}_?{=}_?\ [b]X^{comb}$ is a problem; and $X^{comb} \ {}_?{=}_?\ a,\ a \#_? X^{comb}$ is a problem.

Definition 24 Call the rules in Figure 1 **simplification rules** for problems. Write $Pr \Longrightarrow Pr'$ when Pr' is obtained from Pr by applying a simplification rule. Write $\overset{*}{\Longrightarrow}$ for the transitive and reflexive closure of $\Longrightarrow$.

Lemma 25 $\Longrightarrow$ *is strongly normalising.*

That is: there is no infinite chain $Pr_1 \Longrightarrow Pr_2 \Longrightarrow Pr_3 \Longrightarrow \ldots$.

Proof. By induction on derivations, checking that reduction rules strictly reduce a natural measure based on (n, s) where s is the maximum size of term appearing in the problem, and n is the number of terms of that size, lexicographically ordered.

$$
\begin{array}{ll}
(?=?\mathbf{a}) & a \;?=?\; a,\, Pr \Longrightarrow Pr \\
(?=?\mathsf{f}) & \mathsf{f}(l_1,\ldots,l_n) \;?=?\; \mathsf{f}(s_1,\ldots,s_n),\, Pr \Longrightarrow l_1 \;?=?\; s_1,\ldots,l_n \;?=?\; s_n,\, Pr \\
(?=?[]\mathbf{aa}) & [a]l \;?=?\; [a]s,\, Pr \Longrightarrow l \;?=?\; s,\, Pr \\
(?=?[]\mathbf{ab}) & [a]l \;?=?\; [b]s,\, Pr \Longrightarrow (b\ a)\cdot l \;?=?\; s,\, b \#? l, Pr \\
(?=?\mathbf{X}) & \pi\cdot X^S \;?=?\; \pi'\cdot X^S,\, Pr \Longrightarrow \{a \#? X^S \mid a \in ds(\pi,\pi') \cap S\}, Pr \\
(\#?\mathbf{a}) & a \#? b, Pr \Longrightarrow Pr \\
(\#?\mathbf{X}) & a \#? \pi\cdot X^S, Pr \Longrightarrow Pr \qquad (\pi^{-1}(a) \notin S) \\
(\#?[]\mathbf{aa}) & a \#? [a]l, Pr \Longrightarrow Pr \\
(\#?[]\mathbf{ab}) & a \#? [b]l, Pr \Longrightarrow a \#? l, Pr \\
(\#?\mathsf{f}) & a \#? \mathsf{f}(l_1,\ldots,l_n), Pr \Longrightarrow a \#? l_1,\ldots,a \#? l_n, Pr
\end{array}
$$

Figure 1: Simplification rules for problems

Theorem 26 $\Longrightarrow$ *is confluent and strongly normalising. That is: if* $Pr \stackrel{*}{\Longrightarrow} Pr_1$ *and* $Pr \stackrel{*}{\Longrightarrow} Pr_2$ *then there exists some* Pr_3 *such that* $Pr_1 \stackrel{*}{\Longrightarrow} Pr_3$ *and* $Pr_2 \stackrel{*}{\Longrightarrow} Pr_3$*, and each* Pr *rewrites to a unique normal form.*

Proof. For each predicate there is at most one applicable rewrite rule, hence reductions are locally confluent. (That is: if $Pr \Longrightarrow Pr_1$ and $Pr \Longrightarrow Pr_2$ then there exists some Pr_3 such that $Pr_1 \stackrel{*}{\Longrightarrow} Pr_3$ and $Pr_2 \stackrel{*}{\Longrightarrow} Pr_3$.) The result follows from Newman's Lemma [New42] and Lemma 25.

5 Unification of permissive nominal terms

Definition 27 Call a substitution σ **idempotent** when $X^S\sigma \equiv X^S\sigma\sigma$ for all X^S.

Definition 28 Write

$$Pr\sigma \quad \text{for} \quad \{r\sigma =_\alpha s\sigma \mid r =_\alpha s \in Pr\} \cup \{a \#? r\sigma \mid a \#? r \in Pr\}.$$

Definition 29 A **(unification) solution** to a problem Pr is an idempotent substitution σ such that:[5]

- If $r \;?=?\; s \in Pr$ then $r\sigma =_\alpha s\sigma$.
- If $a \#? r \in Pr$ then $a \notin fa(r\sigma)$.

Write $Sol(Pr)$ for the set of solutions to Pr. We will call Pr **solvable** when $Sol(Pr)$ is non-empty (i.e. when a solution to Pr exists).

Remark 30 A solution to Pr is a substitution which 'makes the equalities and freshness problems valid', just as for first- and higher-order unification; this is simpler than the notion of solution in in nominal unification [UPG04, Definition 3.1], based on 'a substitution + a primitive freshness context'. Note

[5] The condition that σ be idempotent is not absolutely necessary, but it is technically convenient. Our algorithms will generate idempotent solutions so we lose no generality that we care about here.

also (**I3**) below in Definition 35, which does not appear in the theory of nominal unification.

Permissive nominal terms have a distinctly 'nominal' flavour but we design them to have simpler behaviour, and of a kind which is closer to the theory of first- and higher-order unification: One of the motivations for considering nominal terms is the slogan 'ϵ away from informal practice'. In several respects, permissive nominal terms manage to get even closer to informal practice. We now proceed with the proofs, culminating with Theorem 42.

Lemma 31 *If* $Pr \Longrightarrow Pr'$ *then* $Sol(Pr) = Sol(Pr')$.

Proof. By a long but routine verification we check all the simplification rules in Figure 1 and see that σ solves Pr if and only if σ solves Pr'.

Definition 32 Define the **instantiation ordering** on solutions by

$$\sigma_1 \leq \sigma_2 \quad \text{when } \sigma' \text{ exists such that} \quad X^S\sigma_1\sigma' =_\alpha X^S\sigma_2 \text{ for all } X^S.$$

We may also write $\sigma_1 \leq_{\sigma'} \sigma_2$ if the substitution σ' is important. Intuitively $\sigma_1 \leq \sigma_2$ when $\sigma_1 \circ \sigma' =_\alpha \sigma_2$ for some σ'. Note that substitutions *post*-compose.

Lemma 33 $\leq$ *is reflexive and transitive.*

Proof.

- *Reflexive.* We take $\sigma' = id$ and use Lemma 16.
- *Transitive.* Suppose $\sigma_1 \leq \sigma_2$ and $\sigma_2 \leq \sigma_3$. So there exist σ' and σ'' such that

  $$X^S\sigma_1\sigma' =_\alpha X^S\sigma_2 \quad \text{and} \quad X^S\sigma_2\sigma'' =_\alpha X^S\sigma_3 \quad \text{for every } X^S.$$

  By Lemma 16 and Theorem 20 we have $X^S\sigma_1\sigma'\sigma'' =_\alpha X^S\sigma_3$ for every X^S.

Definition 34 A **principal** (or **most general**) solution to Pr is a solution $\sigma \in Sol(Pr)$ such that $\sigma \leq \sigma'$ for all other $\sigma' \in Sol(Pr)$.

Definition 35 Let **instantiating** rules be:

$$\textbf{(I1)}\quad \pi \cdot X^S \;?{=}?\; s, Pr \overset{X^S \mapsto \pi^{-1}\cdot s}{\Longrightarrow} Pr[X^S \mapsto \pi^{-1}\cdot s] \quad (X^S \notin fV(s),\ \pi^{-1}\cdot fv(s) \subseteq S)$$

$$\textbf{(I2)}\quad r \;?{=}?\; \pi \cdot X^S, Pr \overset{X^S \mapsto \pi^{-1}\cdot r}{\Longrightarrow} Pr[X^S \mapsto \pi^{-1}\cdot r] \quad (X^S \notin fV(r),\ \pi^{-1}\cdot fv(r) \subseteq S)$$

$$\textbf{(I3)}\quad a \#_? \pi \cdot X^S, Pr \overset{X^S \mapsto X^{S\setminus\{\pi^{-1}(a)\}}}{\Longrightarrow} Pr[X^S \mapsto X^{S\setminus\{\pi^{-1}(a)\}}] \quad (X^{S\setminus\{\pi^{-1}(a)\}} \notin fV(Pr),\ \pi^{-1}(a) \in S)$$

As is standard, we call '$X^S \notin fV(r)$' the **occurs check**. In (**I3**), $X^{S\setminus\{\pi^{-1}(a)\}}$ is a choice of a fresh unknown of the correct sort. Note that the case of $\pi^{-1}(a) \notin S$ is handled by ($\#_?$**X**) in Figure 1.

Note that (**I3**) is absent from nominal unification [UPG04, Figure 3]; there, $a\#X$ is recorded as part of the solution (rule (#?-**suspension**)).

Here $a, b, c \in comb$ and we write $S = comb \setminus \{a\}$ and $T = comb$.

$\lambda[a]\lambda[b](X^S b) \;?{=}?\; \lambda[b]\lambda[a](aY^T)$

$\overset{(?=?\mathbf{f},?=?[]\mathbf{aa})}{\Longrightarrow}$ $\{\lambda[a](((b\ a)\cdot X^S)a) \;?{=}?\; \lambda[a](aY^T),\ b \#? \lambda[b](X^S b)\}$

$\overset{(?=?\mathbf{f},?=?[]\mathbf{aa},?=?\mathbf{f},\#?\mathbf{f},\#?[]\mathbf{aa})}{\Longrightarrow}$ $\{(b\ a)\cdot X^S \;?{=}?\; a,\ a \;?{=}?\; Y^T\}$

$\overset{(\mathbf{I1}),\ X^S \mapsto b}{\Longrightarrow}$ $\{\ a \;?{=}?\; Y^T\}$

$\overset{(\mathbf{I2}),\ Y^T \mapsto a}{\Longrightarrow}$ $\{\}$ *Solution:* $[X^S \mapsto b, Y^T \mapsto a]$

$\lambda[c]\lambda[b](bX^S) \;?{=}?\; \lambda[c]\lambda[c](cY^T)$

$\overset{(?=?\mathbf{f},?=?[]\mathbf{aa})}{\Longrightarrow}$ $\{\lambda[b](bX^S) \;?{=}?\; \lambda[c](cY^T)\}$

$\overset{(?=?\mathbf{f},?=?[]\mathbf{ab})}{\Longrightarrow}$ $\{c((b\ c)\cdot X^S) \;?{=}?\; cY^T,\ c \#? bX^S\}$

$\overset{(?=?\mathbf{f},?=?\mathbf{a},\mathbf{I1}),\ Y^T \mapsto (b\ c)\cdot X^S}{\Longrightarrow}$ $\{(b\ c)\cdot X^S \;?{=}?\; (b\ c)\cdot X^S,\ c \#? bX^S\}$

$\overset{(?=?\mathbf{X},\#?\mathbf{f},\#?\mathbf{a})}{\Longrightarrow}$ $\{c \#? X^S\}$

$\overset{(\mathbf{I3}),\ X^S \mapsto X^{S\setminus\{c\}}}{\Longrightarrow}$ $\{\}$ *Solution:* $[Y^T \mapsto (b\ c)\cdot X^{S\setminus\{c\}}, X^S \mapsto X^{S\setminus\{c\}}]$

$\lambda[a]\lambda[b](bX^S) \;?{=}?\; \lambda[a]\lambda[a](aY^T)$

$\overset{(?=?\mathbf{f},?=?[]\mathbf{aa})}{\Longrightarrow}$ $\{\lambda[b](bX^S) \;?{=}?\; \lambda[a](aY^T)\}$

$\overset{(?=?\mathbf{f},?=?[]\mathbf{ab})}{\Longrightarrow}$ $\{a((b\ a)\cdot X) \;?{=}?\; aY,\ a \#? bX^S\}$

$\overset{(?=?\mathbf{f},?=?\mathbf{a},\ \mathbf{I1}),\ Y^T \mapsto (b\ a)\cdot X^S}{\Longrightarrow}$ $\{(b\ a)\cdot X \;?{=}?\; (b\ a)\cdot X,\ a \#? bX\}$

$\overset{(?=?\mathbf{X},\#?\mathbf{f},\#?\mathbf{a})}{\Longrightarrow}$ $\{a \#? X^S\}$

$\overset{(\#?\mathbf{X})}{\Longrightarrow}$ $\{\}$ *Solution:* $[Y^T \mapsto (b\ a)\cdot X^S]$

Figure 2: Examples of the permissive nominal unification algorithm in action.

Definition 36 Define a **unification algorithm** as follows:

1. Reduce the problem using simplification rules from Figure 1, as much as possible.
2. Reduce the problem using an instantiation rule from Defininition 35.
3. Repeat steps 1 and 2 as much as possible. If we reduce the problem to $\emptyset$ then we succeed and return the functional composition in order of all substitutions annotating instantiation rules; otherwise we fail.

We give three examples of the unification algorithm in action in Figure 2. The interested reader can compare these with corresponding examples in [FG07, Figure 1, page 29]. Reductions always terminate because at each step either a formula becomes smaller, or the number of unknowns in the problem is reduced by one.

Lemma 37 *Suppose that*

$$\sigma_1 \leq_{\sigma'} \sigma_2 \qquad \sigma_1(X^S) \equiv X^S \qquad X \notin fV(t).$$

Then $[X^S \mapsto t]\circ\sigma_1 \leq_{\sigma'} [X^S \mapsto t]\circ\sigma_2$.

Proof. Unpacking Definition 32, we see that we must show

$$Y([X \mapsto t]\circ\sigma_1\circ\sigma') =_\alpha Y([X^S \mapsto t]\circ\sigma_2) \quad \text{for all } Y$$

There are two cases:

- The case $Y \equiv X^S$. We must show $t\sigma_1\sigma' =_\alpha t\sigma_2$. This follows by Theorem 22.
- The case $Y \not\equiv X^S$. We must show $Y\sigma_1\sigma' =_\alpha Y\sigma_2$. This is by assumption.

Definition 38 Write $\sigma - X^S$ for the substitution such that

- $(\sigma - X^S)(X^S) \equiv X^S$ and
- $(\sigma - X^S)(Y^T) \equiv \sigma(Y^T)$ for all other Y^T.

Lemma 39 *$\pi\cdot X^S\sigma' =_\alpha s\sigma'$ and $X \notin fV(s)$ imply $\sigma' =_\alpha [X^S \mapsto \pi^{-1}\cdot s]\circ(\sigma' - X^S)$.*

Proof. We reason as follows:

$$\begin{array}{lcll} \pi\cdot X^S\sigma' & =_\alpha & (\pi\cdot X^S)([X^S \mapsto \pi^{-1}\cdot s]\circ(\sigma' - X^S)) & \text{Lemma 12} \\ & \equiv & s(\sigma' - X^S) & \\ & \equiv & s[X^S \mapsto \pi^{-1}\cdot s]\circ(\sigma' - X^S) & X^S \notin fV(s) \\ & \equiv & s\sigma' & \end{array}$$

Lemma 40 *If $Pr \stackrel{\sigma}{\Longrightarrow} Pr'$ and $\sigma' \in Sol(Pr')$ then $\sigma\circ\sigma' \in Sol(Pr)$.*

Proof. Suppose $Pr \stackrel{\sigma}{\Longrightarrow} Pr'$. There are two cases:

– The case for (**I1**) and (**I2**). The following conditions hold:

– $Pr = \{\pi\cdot X \;?\!=\!?\; r\} \cup Pr''$ or $Pr = \{r \;?\!=\!?\; \pi\cdot X\} \cup Pr''$,

– $X \notin fV(r)$,

– $\sigma \equiv [X \mapsto \pi^{-1}\cdot r]$, and

– $Pr' = Pr''\sigma$.

Suppose also that $\sigma' \in Sol(Pr')$. Then σ' is idempotent and $Pr''(\sigma\circ\sigma') = Pr'\sigma'$. It is easy to verify that $\sigma\circ\sigma'$ is idempotent. By Lemmas 19 and 14 we conclude $(\pi\cdot X)(\sigma\circ\sigma') =_\alpha r(\sigma\circ\sigma')$ and the result follows.

– The case (**I3**). Suppose $Pr = \{a \#_? \pi\cdot X^S\} \cup Pr''$ and $Pr \stackrel{\sigma}{\Longrightarrow} Pr''\sigma$ with $\sigma \equiv [X^S \mapsto X^{S\setminus\{\pi^{-1}(a)\}}]$ and $\pi^{-1}(a) \in S$. Further, suppose $\sigma' \in Sol(Pr''\sigma)$. Then $\sigma\circ\sigma' \in Sol(Pr'')$ and it suffices to show that $a\#(\pi\cdot X^{S\setminus\{\pi^{-1}(a)\}})\sigma'$. This follows immediately from Definition 17, and we have the result.

Theorem 41 *If $Pr \stackrel{\sigma}{\Longrightarrow} \emptyset$ then $\sigma \in Sol(Pr)$.*

Proof. By induction on the length of the path in $\stackrel{\sigma}{\Longrightarrow}$.

– *Path length* 0. Then $Pr = \emptyset$ and $\sigma \equiv id$. The result follows.

– *Path length* $k+1$. There are two cases:

– *The non-instantiating case.* Suppose $Pr \Longrightarrow Pr' \stackrel{\sigma}{\Longrightarrow} \emptyset$. By Lemma 31, $Sol(Pr) = Sol(Pr')$, and the result follows by the inductive hypothesis.

– *The instantiating case.* Suppose $Pr \stackrel{\sigma}{\Longrightarrow} Pr' \stackrel{\sigma'}{\Longrightarrow}{}^{*} \emptyset$ and $\sigma'' = \sigma \circ \sigma'$. It follows by the inductive hypothesis that $\sigma' \in Sol(Pr')$ and the result follows by Lemma 40.

Theorem 42 *If $Pr \stackrel{\sigma}{\Longrightarrow}{}^{*} \emptyset$ then σ is a principal solution to Pr. Unpacking Definition 34 this means that $\sigma \in Sol(Pr)$ and if $\sigma' \in Sol(Pr)$ then $\sigma \leq \sigma'$.*

Proof. By Theorem 41 $\sigma \in Sol(Pr)$. We work by induction on the path length of $Pr \stackrel{\sigma}{\Longrightarrow}{}^{*} \emptyset$ showing that it is principal.

– *Path length* 0. Then $Pr = \emptyset$ and $\sigma = id$. Suppose $\sigma' \in Sol(Pr)$. Then $X^S(id \circ \sigma') \leq X^S\sigma'$ for all X^S. The result follows.

– *Path length* $k+1$. There are two cases:

– *The non-instantiating case.* Suppose $Pr \Longrightarrow Pr' \stackrel{\sigma}{\Longrightarrow}{}^{*} \emptyset$ where $Pr \Longrightarrow Pr'$ is a non-instantiating simplification. By inductive hypothesis, σ is a principal solution of Pr', and by Lemma 31, σ is also a principal solution of Pr. The result follows.

– *The instantiating case.* There are two cases:

– The cases (**I1**) and (**I2**). Without loss of generality, suppose $Pr \cup \{\pi \cdot X^S \ {}_?{=}_?\ r\} \stackrel{\sigma}{\Longrightarrow} Pr\sigma$ and $Pr \stackrel{\sigma'}{\Longrightarrow}{}^{*} \emptyset$, where $\sigma \equiv [X^S \mapsto \pi^{-1} \cdot r]$. Further, suppose $\sigma'' \in Sol(Pr \cup \{\pi \cdot X^S \ {}_?{=}_?\ r\})$. By Lemma 40, $\sigma'' - X^S \in Sol(Pr\sigma)$, therefore by inductive hypothesis and Theorem 41, $\sigma' \leq \sigma'' - X^S$. By Lemma 37, $\sigma \circ \sigma' \leq \sigma \circ (\sigma'' - X^S)$. By Lemma 39, $\sigma'' =_\alpha \sigma \circ (\sigma'' - X^S)$ hence $\sigma \circ (\sigma'' - X^S) \leq \sigma''$. Therefore, using Lemma 33, we have $\sigma \circ \sigma' \leq \sigma''$, as required.

– The case (**I3**). Suppose $Pr \cup \{a \#_? X^S\} \stackrel{\sigma}{\Longrightarrow} Pr\sigma$ and $Pr \stackrel{\sigma'}{\Longrightarrow}{}^{*} \emptyset$, where $\sigma \equiv [X^S \mapsto X^{S \setminus \{a\}}]$, $\pi^{-1}(a) \in S$, and $X^{S \setminus \{a\}} \notin fV(Pr)$. By inductive hypothesis σ' is a principal solution for $Pr\sigma$. By Lemma 40 $\sigma \circ \sigma' \in Sol(Pr \cup \{a \#_? X^S\})$. The result follows by Lemma 31.

Theorem 43 *Given a problem Pr, if the algorithm of Definition 36 succeeds then it returns a principal solution; if it fails then there is no solution.*

Proof. If the algorithm fails then it does so because it reduces Pr to a problem that can have no solution because it contains an equality or freshness predicate of the form $\mathsf{f}(\ldots) \ {}_?{=}_?\ \mathsf{g}(\ldots)$, $\mathsf{f}(\ldots) \ {}_?{=}_?\ [a]s$, $\mathsf{f}(\ldots) \ {}_?{=}_?\ a$, $a \#_? a$, or $\pi \cdot X^S \ {}_?{=}_?\ s$ where $\pi^{-1} \cdot fa(s) \not\subseteq S$ (we elide symmetric cases). In that case by Lemmas 31 and 40 no solution to Pr can exist. If the algorithm succeeds, the result follows by Theorem 42.

References

[BU07] Stefan Berghofer and Christian Urban. A head-to-head comparison of de Bruijn indices and names. *Electronic Notes in Theoretical Computer Science*, 174(5):53–67, 2007.

[Che05] James Cheney. Nominal logic and abstract syntax. *SIGACT News (logic column 14)*, 36(4):47–69, 2005.

[CP07] Ranald A. Clouston and Andrew M. Pitts. Nominal equational logic. *Electronic Notes in Theoretical Computer Science*, 172:223–257, 2007.

[CU03] J. Cheney and C. Urban. System description: Alpha-Prolog, a fresh approach to logic programming modulo alpha-equivalence. Technical Report DSIC-II/12/03, Departamento de Sistemas Informaticos y Computacion, Universidad Politecnica de Valencia, 2003.

[dB72] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 5(34):381–392, 1972.

[Dow01] Gilles Dowek. Higher-order unification and matching. In *Handbook of automated reasoning*, pages 1009–1062. Elsevier, 2001.

[FG05] Maribel Fernández and Murdoch J. Gabbay. Nominal rewriting with name generation: abstraction vs. locality. In *Proc. 7th Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming (PPDP'2005)*, pages 47–58. ACM Press, 2005.

[FG07] Maribel Fernández and Murdoch J. Gabbay. Nominal rewriting. *Information and Computation*, 205(6):917–965, 2007.

[Gab07] Murdoch J. Gabbay. A General Mathematics of Names. *Information and Computation*, 205:982–1011, July 2007.

[Gab08] Murdoch J. Gabbay. Nominal algebra and the HSP theorem. *Journal of Logic and Computation*, 2008. To appear.

[GL08] Murdoch J. Gabbay and Stéphane Lengrand. The lambda-context calculus. *Electronic Notes in Theoretical Computer Science*, 196:19–35, 2008.

[GM07] Murdoch J. Gabbay and Aad Mathijssen. A formal calculus for informal equality with binding. In *WoLLIC'07: 14th Workshop on Logic, Language, Information and Computation*, volume 4576 of *Lecture Notes in Computer Science*, pages 162–176, 2007.

[GM08] Murdoch J. Gabbay and Dominic P. Mulligan. Two-and-a-halfth Order Lambda-calculus. In *Workshop on Functional and Logic Programming (WFLP)*, 2008. To appear.

[GP99] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224. IEEE Computer Society Press, 1999.

[GP01] Murdoch J. Gabbay and A. M. Pitts. A New Approach to Abstract Syntax with Variable Binding (journal version). *Formal Aspects of Computing*, 13(3–5):341–363, 2001.

[Ham01] Makoto Hamana. A logic programming language based on binding algebras. In *TACS'01*, volume 2215 of *Lecture Notes in Computer Science*, pages 243–262. Springer, 2001.

[LV08] Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. In *Proceedings of RTA'08*, volume 5117 of *Lecture Notes in Computer Science*. Springer, 2008.

[Mat07] Aad Mathijssen. *Logical Calculi for Reasoning with Binding*. PhD thesis, Technische Universiteit Eindhoven, 2007.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497 – 536, 1991.

[New42] M.H.A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942.

[NR96] Peter Nickolas and Peter J. Robinson. The Qu-Prolog unification algorithm: formalisation and correctness. *Theoretical Computer Science*, 169(1):81–112, 1996.

[Pit02] Andrew M. Pitts. Equivariant syntax and semantics. In *Proceedings of ICALP'02*, pages 32–36. Springer, 2002.

[UPG04] C. Urban, A. M. Pitts, and Murdoch J. Gabbay. Nominal Unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 Orsay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

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