Nominal Unification from a Higher-Order Perspective*

Jordi Levy[1] and Mateu Villaret[2]

[1] Artificial Intelligence Research Institute (IIIA),
Spanish Council for Scientific Research (CSIC), Barcelona, Spain.
http://www.iiia.csic.es/~levy
[2] Departament d'Informàtica i Matemàtica Aplicada (IMA),
Universitat de Girona (UdG), Girona, Spain.
http://ima.udg.edu/~villaret

Abstract. Nominal Logic is an extension of first-order logic with equality, name-binding, name-swapping, and freshness of names. Contrarily to higher-order logic, bound variables are treated as atoms, and only free variables are proper unknowns in nominal unification. This allows "variable capture", breaking a fundamental principle of lambda-calculus. Despite this difference, nominal unification can be seen from a higher-order perspective. From this view, we show that nominal unification can be reduced to a particular fragment of higher-order unification problems: higher-order patterns unification. This reduction proves that nominal unification can be decided in quadratic deterministic time.

1 Introduction

Nominal Logic is a version of first-order many-sorted logic with equality and mechanisms for renaming via name-swapping, for name-binding, and for freshness of names. It also provides a *new-quantifier* [GP99], to modeling name generation and locality. It was introduced at the beginning of this decade by Pitts [Pit01,Pit03]. These first works have inspired a sequel of papers where bindings and freshness are introduced in other topics, like equational logic [CP07], rewriting [FG05,FG07], unification [UPG03,UPG04], Prolog [CU04,UC05].

This paper is concerned with *Nominal Unification* [UPG03,UPG04], an extension of first-order unification where terms can contain binders and unification is performed modulo α-equivalence. Moreover, (first-order) variables (*unknowns*) are allowed to "capture" bound variables (*atoms*). [UPG03,UPG04] describe a sound and complete, but inefficient (exponential), algorithm for nominal unification. Later this algorithm was extended to deal with the new-quantifier and locality in [FG05]. In [CF07] there is a description of a direct but exponential implementation in Maude, and a polynomial implementation in OCAML based on termgraphs.

* This research has been partially founded by the CICYT research project TIN2007-68005-C04-01/02/03.

The use of α-equivalence and binders in nominal logic immediately suggests to view nominal unification from a higher-order perspective, the one that we adopt in this paper. Some intuitions about this relation are already roughly described in [UPG04]; and in [Che05] there is a reduction from higher-order pattern unification to nominal unification (here we prove the opposite reduction).

The main benefit of nominal logic, compared to a higher-order logic, is that it allows the use of binding and α-equivalence without the other difficulties associated with the λ-calculus. In particular, with respect to unification, we have that nominal unification is unitary (most general unifiers are unique) and decidable [UPG03,UPG04], whereas higher-order unification is undecidable and infinitary [Luc72,Gol81,Lev98,LV00].

In this paper we fully develop the study of nominal unification from the higher-order view. We show that full higher-order unification is not needed but only a fragment: *Higher-order Pattern Unification* [Mil91,Nip93,Qia96]. This subclass of problems were proposed by Miller [Mil91]. Contrarily to general higher-order unification, higher-order pattern unification is decidable and unitary [Mil91,Nip93]. Moreover, the problem can be solved in linear time [Qia96]. All this will lead us to show how to reduce nominal unification to higher-order pattern unification, and to conclude its decidability in quadratic time as well as the uniqueness of most general unifiers.

From a higher-order perspective, nominal unification can be seen as a variant of higher-order unification where:

1. variables are all first-order typed, and constants are of order at most three (therefore, nominal unification is a fragment of second-order unification),
2. unification is performed modulo α-equivalence, instead of the usual α and β-equivalence,
3. instances of variables (unknowns) are allowed to capture bound variables (atoms), contrarily to the standard higher-order definition, and
4. apart from the usual term-equality predicate, we use a "freshness" predicate $a \# t$ with the intended meaning: bound variable a does not occur free in the instance of term t.

The first requirement does not suppose a difficulty. On the contrary, in the reduction to higher-order unification we will add capturable variables as arguments of free variables. The fact that original variables do not have arguments will allow us to reduce nominal unification to higher-order pattern unification.

The second requirement is not a difficulty, either. As all variables are first-order typed, their instantiation can not introduce β-redexes, and β-reduction is not really necessary.

The third requirement is the key point that makes nominal unification an interesting subject of research. Variable capture is always a trouble spot. Roughly speaking, the main idea of this paper is to translate (first-order) nominal variables to higher-order variables with the list of bound variables that it can "capture" as arguments. This implies that the arguments of free variables will be lists of pairwise distinct bound variables, hence higher-order patterns.

The fourth requirement can also be overcome by translating freshness predicates into equality predicates.

We structure the paper as follows: after some preliminaries in Section 2, in Section 3, we illustrate by examples the main ideas of the reduction at the same time that we show the main features of nominal unification. In Section 4 we show how to translate a nominal unification problem into a higher-order patterns unification problem, and we prove that this translation is effectively a quadratic time reduction in Section 5. In Section 6 we conclude.

2 Preliminaries

2.1 Nominal Unification

In nominal logic we talk about *variables* and *atoms*. Only variables may be instantiated, and only atoms may be bounded. They roughly correspond to the higher-order notions of free and bound variables, respectively, but are considered as completely different entities. Therefore, contrarily to the higher-order perspective, the distinction between free and bound variables does not only depend on the occurrences, i.e. in the existence of a binder above them.

In *nominal signatures* we have *sorts of atoms* (typically ν) and *sorts of data* (typically δ) as disjoint sets. *Atoms* (typically $a, b, \ldots$) have one of the sorts of atoms. *Variables*, also called *unknowns*, (typically $X, Y, \ldots$) have a sort of atom or sort of data, i.e. of the form $\nu \mid \delta$. *Nominal function symbols* (typically $f, g, \ldots$) have an *arity* of the form $\tau \to \delta$, where δ is a sort of data and τ is a sort given by the grammar $\tau ::= \nu \mid \delta \mid \tau \times \tau \mid \langle \nu \rangle \tau$. Abstractions have sorts of the form $\langle \nu \rangle \tau$.

Nominal terms (typically $t, u, \ldots$) are given by the grammar:

$$t ::= \langle t_1, t_2 \rangle \mid f\, t \mid a \mid a.t \mid \pi \cdot X$$

where f is a function symbol, a is an atom, π is a permutation (finite list of swappings), and X is a variable. They are called respectively *pairs*, *application*, *atom*, *abstraction* and *suspension*. For simplicity, we do not consider the *unit value*.

A *swapping* $(a\, b)$ is a pair of atoms of the same sort. The effect of a swapping over an atom is defined by $(a\, b) \cdot a = b$ and $(a\, b) \cdot b = a$ and $(a\, b) \cdot c = c$, when $c \neq a, b$. For the rest of terms the extension is straightforward, in particular, $(a\, b) \cdot (c.t) = \big((a\, b) \cdot c\big).\big((a\, b) \cdot t\big)$. A *permutation* is a (possibly empty) sequence of swappings. Its effect is defined by $(a_1\, b_1) \ldots (a_n\, b_n) \cdot t = (a_1\, b_1) \cdot \big((a_2\, b_2) \ldots (a_n\, b_n) \cdot t\big)$. Notice that every permutation π naturally defines a bijective function from the set of atoms to the sets of atoms, that we will also represent as π. *Suspensions* are uses of variables with a permutation of atoms waiting to be applied once the variable is instantiated.

A *nominal unification problem* (typically P) is a set of equations of the form $t \stackrel{?}{\approx} u$ or $a \#^{?} t$. A *freshness environment* (typically ∇) is a list of pairs $a \# X$ stating that the instantiation of X cannot capture a.

A *solution* of a nominal problem is given by a substitution σ and a freshness environment ∇. Substitutions are like in first-order logic, and allow atom capture,

for instance $[X \mapsto a]a.X = a.a$. Formally, the pair $\langle \nabla, \sigma \rangle$ solves P if, $\nabla \vdash a \# \sigma(t)$, for equations $a \#^? t \in P$, and $\nabla \vdash \sigma(t) \approx \sigma(u)$, for equations $t \stackrel{?}{\approx} u \in P$. The predicates $\approx$ and $\#$ are defined in [UPG03,UPG04] by means of a theory. Their intended meanings are: $\nabla \vdash a \# t$ holds if, for every substitution σ avoiding the atom captures forbidden by ∇, a is not free in $\sigma(t)$; $\nabla \vdash t \approx u$ holds if, for every substitution σ avoiding the atom captures forbidden by ∇, t and u are α-convertible.

2.2 Higher-Order Pattern Unification

In higher-order signatures we have types constructed from a set of basic types (typically $\alpha, \ldots$) using the grammar $\tau ::= \alpha \,|\, \tau \to \tau$, where $\to$ is associative to the right).

λ-terms are built using the grammar $t ::= x \,|\, c \,|\, \lambda x.t \,|\, t_1\, t_2$, where x is a variable and c is a constant. Other standard concepts of λ-calculus, like free variables (noted FV), bound and free occurrences of variables, α-conversion, β-reduction, η-long β-normal form, substitutions, most general unifiers, etc. are defined as usual (see [Dow01]). The domain of a substitution σ is denoted by $Dom(\sigma)$, and we say that X occurs in σ if X occurs free in $\sigma(Y)$ for some $Y \in Dom(\sigma)$.

A *higher-order pattern* is a simply typed λ-term where, when written in normal form, all free variable occurrences are applied to lists of pairwise distinct bound variables. *Higher-order pattern unification* is the problem of deciding if there exists a unifier for a set of equations $t \stackrel{?}{=} u$ between higher-order patterns. The most general unifiers of a pattern unification problem is unique (up to free variable renaming). Moreover, it instantiates variables by higher-order patterns. There is an algorithm that finds these unifiers, if exist, in linear time [Qia96].

3 Four Examples

In order to describe the reduction of nominal unification to higher-order pattern unification, we will use the unification problems proposed in [UPG03,UPG04] as a quiz.

Example 1. The nominal equation

$$a.b.\langle X_1, b\rangle \stackrel{?}{\approx} b.a.\langle a, X_1\rangle$$

has no nominal unifiers. Notice that, although unification is performed modulo α-equivalence, as far as we allow atom capture, we can not α-convert terms before instantiating them. Therefore, this problem is *not* equivalent to

$$a.b.\langle X_1, b\rangle \stackrel{?}{\approx} a.b.\langle b, X_1\rangle$$

which is solvable, and must be α-converted as

$$a.b.\langle X_1, b\rangle \stackrel{?}{\approx} a.b.\langle b, (a\,b) \cdot X_1\rangle$$

Recall that $(a\,b) \cdot X_1$ means that, after instantiating X_1 with a term that possibly contain a or b, we have to exchange these variables.

According to the ideas exposed in the introduction, we have to replace every occurrence of X_1 by $X'_1\, a'\, b'$, since a, b is the list of atoms (bound variables) that can be captured. We get[3]:

$$\lambda a'.\lambda b'.\langle X'_1\, a'\, b'\ ,\ b'\rangle \stackrel{?}{=} \lambda b'.\lambda a'.\langle a'\ ,\ X'_1\, a'\, b'\rangle$$

Since this is a higher-order unification problem, we can α-convert one of the sides of the equation and get:

$$\lambda a'.\lambda b'.\langle X'_1\, a'\, b'\ , b'\ \rangle \stackrel{?}{=} \lambda a'.\lambda b'.\langle b'\ ,\ X'_1\, b'\, a'\rangle$$

which is unsolvable, like the original nominal equation.

Example 2. The nominal equation

$$a.b.\langle X_2, b\rangle \stackrel{?}{\approx} b.a.\langle a, X_3\rangle$$

is solvable. Its translation is

$$\lambda a'.\lambda b'.\langle X'_2\, a'\, b'\ ,\ b'\rangle \stackrel{?}{=} \lambda b'.\lambda a'.\langle a'\ ,\ X'_3\, a'\, b'\rangle$$

The most general unifier of this higher-order pattern unification problem is

$$X'_2 \mapsto \lambda x.\lambda y.y$$
$$X'_3 \mapsto \lambda x.\lambda y.x$$

Now, taking into account that the first argument corresponds to the bound variable a', and the second one to b', we can reconstruct the most general nominal unifier as:

$$X_2 \mapsto b$$
$$X_3 \mapsto a$$

Example 3. In some cases, there are interrelationships between the instances of variables that make reconstruction of unifiers more difficult. This is shown with the following nominal equation:

$$a.b.\langle b, X_4\rangle \stackrel{?}{\approx} b.a.\langle a, X_5\rangle$$

that is solvable. Its translation results on:

$$\lambda a'.\lambda b'.\langle b'\ ,\ X'_4\, a'\, b'\rangle \stackrel{?}{=} \lambda b'.\lambda a'.\langle a'\ ,\ X'_5\, a'\, b'\rangle$$

and its most general unifier is:[4]

$$X'_4 \mapsto \lambda x.\lambda y.X'_5\, y\, x$$

This higher-order unifier can be used to reconstruct the nominal unifier

$$X_4 \mapsto (a\ b) \cdot X_5$$

The swapping $(a\, b)$ comes from the fact that the arguments of X'_5 and the lambda abstractions in front have a different order.

[3] In this example we allow the use of the binary constant $\langle\ ,\ \rangle$ in λ-calculus for pairs. Later on we will describe formally the translation algorithm and how pairs are really translated.

[4] The unifier $X'_5 \mapsto \lambda x.\lambda y.X'_4\, y\, x$ is equivalent modulo variable renaming. In this case we obtain the also equivalent nominal unifier $X_5 \mapsto (a\ b) \cdot X_4$.

Example 4. The solution of a nominal unification problem is not just a substitution, but a pair (∇, σ) where σ is a substitution and ∇ is a freshness environment imposing some restrictions on the atoms that can occur free in the fresh variables introduced by σ. The nominal equation

$$a.b.\langle b, X_6\rangle \stackrel{?}{\approx} a.a.\langle a, X_7\rangle$$

has as solution

$$\begin{array}{l}\sigma = [X_6 \mapsto (b\ a) \cdot X_7] \\ \nabla = \{b \# X_7\}\end{array}$$

where the freshness environment is not empty and requires instances of X_7 to not contain (free) occurrences of b. Let us see how this is reflected when we translate the problem into a higher unification problem. The translation of the equation using the translation algorithm results on:

$$\lambda a'.\lambda b'.\langle b'\ ,\ X'_6\ a'\ b'\rangle \stackrel{?}{=} \lambda a'.\lambda a'.\langle a'\ ,\ X'_7\ a'\ b'\rangle \tag{1}$$

After a convenient α-conversion we get

$$\lambda a'.\lambda c'.\langle c'\ ,\ X'_6\ a'\ c'\rangle \stackrel{?}{=} \lambda a'.\lambda c'.\langle c'\ ,\ X'_7\ c'\ b'\rangle$$

The most general unifier is again unique:

$$\begin{array}{l}X'_6 \mapsto \lambda x.\lambda y.X_8\ y\ b' \\ X'_7 \mapsto \lambda x.\lambda y.X_8\ x\ y\end{array}$$

Nevertheless, in this case we cannot reconstruct the nominal unifier. Moreover, by instantiating the free variable b', we get other (non-most general) higher-order unifier without nominal counterpart. The translation does not work in this case because b' occurs free in the right hand side of (1). We translate both atoms and unknowns as variables. Occurrences of unknowns become free occurrences of variables, and occurrences of atoms, if are bounded, become bound occurrences of variables. Therefore, in most cases, after the translation the distinction atom/unknown become a distinction free/bound variable. However, if atoms are not bounded, as in this case, they are translated as free variables, hence are instantiable, whereas atoms are not instantiable.

To avoid this problem, we have to ensure that any occurrence of an atom is translated as a bound variable occurrence. This is easily achievable if we add binders in front of both sides of the equation. Therefore, the correct translation of this problem is:

$$\lambda a'.\lambda b'.\lambda a'.\lambda b'.\langle b'\ ,\ X'_6\ a'\ b'\rangle \stackrel{?}{=} \lambda a'.\lambda b'.\lambda a'.\lambda a'.\langle a'\ ,\ X'_7\ a'\ b'\rangle$$

where two new binder $\lambda a'.\lambda b'$ have been introduced in front of both sides of the equation. The most general unifier is now:

$$\begin{array}{l}X'_6 \mapsto \lambda x.\lambda y.X'_8\ y \\ X'_7 \mapsto \lambda x.\lambda y.X'_8\ x\end{array}$$

This can be used to reconstruct the nominal substitution:

$$\begin{array}{l}X_6 \mapsto (a\ b) \cdot X_8 \\ X_7 \mapsto X_8\end{array}$$

As far as $X'_8\,x$ is translated back as X_8, and $X'_8\,x$ does not uses the second argument (the one corresponding to b), we have to add a supplementary condition ensuring that X_8 does not contain free occurrences of b. This results on the freshness environment $\{b\,\#\,X_8\}$. Then, $X'_8\,y$ is translated back as $(a\,b)\cdot X_8$.

4 The Translation Algorithm

In this Section we formalize the translation algorithm. We transform nominal unification problems into higher-order unification problems. Both kinds of problems are expressed using distinct kinds of signatures. In nominal unification we have sorts of atoms and sorts of data. In higher-order this distinction is no longer necessary, and we will have a *base type* (typically ν' and δ') for every sort of atoms ν or sort of data δ. We give a *sort to types translation function* that allows us to translate any sort into a type.

Definition 1. *The translation function is defined on sorts inductively as follows.*

$$\begin{aligned}
&[\![\delta]\!] = \delta' \\
&[\![\nu]\!] = \nu' \\
&[\![\tau_1 \times \tau_2]\!] = ([\![\tau_1]\!] \to [\![\tau_2]\!] \to \top) \to \top \\
&[\![\tau_1 \to \tau_2]\!] = [\![\tau_1]\!] \to [\![\tau_2]\!] \\
&[\![\langle\nu\rangle\tau]\!] = \nu' \to [\![\tau]\!]
\end{aligned}$$

where $\top$, δ' and ν' are base types.

Remark 1. The translation function for terms depends on *all* the atoms occurring in the nominal unification problem. We assume that there exists a *fixed*, *finite* and *ordered* list of atoms $\langle a_1, \ldots, a_n\rangle$ used in the problem. This seems to contradict the assumption of a countably *infinite* set of atoms for every sort. However, this does not imply a loss of generality as far as every nominal unification problem only contains a finite set of atoms, and its solutions can be expressed without adding new atoms (see Lemma 5). From now on, we will consider this list given and fixed.

For every function symbol $f, \ldots$, we will use a constant with name $f', \ldots$. Nominal atoms $a, b \ldots$ are translated as (bound) variables, with the names $a', b', \ldots$. The lack of distinction between sorts of atoms and data, after the translation, forces us to ensure that the translation of every atom occurrence will correspond to a *bounded* occurrence of variable. For every variable (unknown) $X, Y, \ldots$, we will use a (free) variable with name $X', Y', \ldots$. Trivially, atom abstractions $a.t$ are translated as lambda abstractions, and data $f\,t$ as applications. The translation of suspensions $\pi \cdot X$ is more complicated, as far as it gets rid of atom capture.

Definition 2. *Let $\langle a_1, \ldots, a_n\rangle$ be an ordered list of atoms occurring in the nominal unification problem. The translation function from nominal terms with a*

freshness environments ∇ into λ-terms is defined inductively as follows.

$$\begin{array}{l}
[\![\langle t_1, t_2\rangle]\!]_\nabla = \lambda p.p\,[\![t_1]\!]_\nabla\,[\![t_2]\!]_\nabla \\
[\![a]\!]_\nabla = a' \\
[\![f\,t]\!]_\nabla = f'\,[\![t]\!]_\nabla \\
[\![a.t]\!]_\nabla = \lambda a'.[\![t]\!]_\nabla \\
[\![\pi \cdot X]\!]_\nabla = X'\,(\pi \cdot b_1)' \ldots (\pi \cdot b_m)' \quad \textit{where } b_j \,\#\, X \notin \nabla, \textit{ for } j = 1, \ldots, m
\end{array}$$

where, if $a : \nu$ is an atom, then $a' : [\![\nu]\!]$ is a bound variable, if $f : \tau$ is a function symbol, then $f' : [\![\tau]\!]$ is a constant, and if $X : \tau$ is a variable, then $\langle b_1, \ldots, b_m\rangle \subseteq \langle a_1, \ldots, a_n\rangle$ is the sublist of atoms such that $b_j \,\#\, X \notin \nabla$ and $X' : [\![\nu_1]\!] \to \ldots \to [\![\nu_m]\!] \to [\![\tau]\!]$ is a free variable and $b_j : \nu_j$.[5]

Lemma 1. *Let ∇ be a freshness environment.*
For every nominal term t of sort τ, the λ-term $[\![t]\!]_\nabla$ has type $[\![\tau]\!]$.
Therefore, $[\![t]\!]_\nabla$ is a well-typed λ-term, for every nominal term t.

Proof. The proof is simple by structural induction on t. The only point that merits a more detailed explanation is the case of suspensions. Since $a_i : \nu_i$, $X : \tau$, and $X' : [\![\nu_{i_1}]\!] \to \cdots \to [\![\nu_{i_m}]\!] \to [\![\tau]\!]$, we have $[\![X]\!]_\nabla = X'[\![a_{i_1}]\!]_\nabla \ldots [\![a_{i_m}]\!]_\nabla : [\![\tau]\!]$. When X is affected by a swapping $(a_{i_j}\ a_{i_k})$ we also have $[\![(a_{i_j}\ a_{i_k}) \cdot X]\!]_\nabla = X'[\![a_{i_1}]\!]_\nabla \ldots [\![a_{i_{j-1}}]\!]_\nabla [\![a_{i_k}]\!]_\nabla [\![a_{i_{j+1}}]\!]_\nabla \ldots [\![a_{i_{k-1}}]\!]_\nabla [\![a_{i_j}]\!]_\nabla [\![a_{i_{k+1}}]\!]_\nabla \ldots [\![a_{i_m}]\!]_\nabla \ : \ [\![\tau]\!]$ because the suspension is not a valid nominal term unless a_{i_j} and a_{i_k} belong to the same sort. The same applies to arbitrary permutations. □

Example 5. Given the nominal term $a.b.c.(c\,a)(a\,b) \cdot X$, after applying the substitution $[X \mapsto \langle\langle a, b\rangle, c\rangle]$ we get the instantiation $a.b.c.\langle\langle b, c\rangle, a\rangle$. Let $\langle a, b, c\rangle$ be the (ordered) list of atoms of our problem. The translation of the term and its instantiation results into $\lambda a'.\lambda b'.\lambda c'.X'b'c'a'$ and $\lambda a'.\lambda b'.\lambda c'.\lambda p.p(\lambda p.p\,b'c')a'$, respectively. There is a λ-substitution $[X' \mapsto \lambda a'.\lambda b'.\lambda c'.\lambda p.p(\lambda p.p\,a'b')c']$ that when applied to the translation of the original term results into the translation of its instantiation. Graphically this can be represented as the commutation of the following diagram, and is proved in general in Lemma 4.

$$\begin{array}{ccc}
a.b.c.(c\,a)(a\,b) \cdot X & \xrightarrow{\quad [X \mapsto \langle\langle a, b\rangle, c\rangle] \quad} & a.b.c.\langle\langle b, c\rangle, a\rangle \\
\Big\downarrow [\![\ ]\!] & & \Big\downarrow [\![\ ]\!] \\
\lambda a'.\lambda b'.\lambda c'.X'b'c'a' & \xrightarrow{[X' \mapsto \lambda a'.\lambda b'.\lambda c'.\lambda p.p(\lambda p.p\,a'b')c']} & \lambda a'.\lambda b'.\lambda c'.\lambda p.p(\lambda p.p\,b'c')a'
\end{array}$$

As we have said nominal unification problems contains two kinds of judgments: *freshness equations* like $a \,\#^?\, t$, and *equality equations* like $t \stackrel{?}{\approx} u$. Equality equations are trivially translated as higher-order unification problems, adding some λ-bindings in front of both terms to ensure that all occurrences of atoms are translated as bounded occurrences of variables. Freshness equations $a \,\#^?\, t$ are translated as equations of the form $Y \stackrel{?}{\approx} t$ where Y is a fresh variable that will not be able to capture free occurrences of a.

[5] Notice that b_j and $\pi \cdot b_j$ are of the same sort.

Definition 3. *Let $\langle a_1, \ldots, a_n \rangle$ be an ordered list of atoms occurring in the nominal unification problem. The translation function is defined on nominal problems inductively as follows*

$[\![\{a_i \#^? t\} \cup P]\!] = \{\lambda a'_1. \ldots . \lambda a'_n . Y' a'_1 \ldots a'_{i-1}\, a'_{i+1} \ldots a'_n \stackrel{?}{=} \lambda a'_1. \ldots . \lambda a'_n . [\![t]\!]_\emptyset\} \cup [\![P]\!]$
$[\![\{t \stackrel{?}{\approx} u\} \cup P]\!] = \{\lambda a'_1. \ldots . \lambda a'_n . [\![t]\!]_\emptyset \stackrel{?}{=} \lambda a'_1. \ldots . \lambda a'_n . [\![u]\!]_\emptyset\} \cup [\![P]\!]$

where Y' is a fresh variable with the appropriate type.

Remark 2. Alternatively to Definition 3, we could decompose freshness equations into simple pieces until we get a freshness environment, and use it in the translation of the equality equations. This would result into the following inductive definition.

$[\![\{a \#^? \langle t_1, t_2 \rangle\} \cup P]\!] = [\![\{a \#^? t_1, a \#^? t_2\} \cup P]\!]$
$[\![\{a \#^? b\} \cup P]\!] = [\![P]\!]$ if $a \neq b$
$[\![\{a \#^? f\, t\} \cup P]\!] = [\![\{a \#^? t\} \cup P]\!]$
$[\![\{a \#^? b.t\} \cup P]\!] = [\![\{a \#^? t\} \cup P]\!]$ if $a \neq b$
$[\![\{a \#^? a.t\} \cup P]\!] = [\![P]\!]$
$[\![\{a \#^? \pi \cdot X\} \cup P]\!] = [\![\{\pi^{-1} \cdot a \# X\} \cup P]\!]$
$[\![\nabla \cup P]\!] = [\![P]\!]_\nabla$ if for all $\{a \#^? t\} \in \nabla, t$ is a variable
$[\![\{t \stackrel{?}{\approx} u\} \cup P]\!]_\nabla = \{\lambda a'_1. \ldots . \lambda a'_n . [\![t]\!]_\nabla \stackrel{?}{=} \lambda a'_1. \ldots . \lambda a'_n . [\![u]\!]_\nabla\} \cup [\![P]\!]_\nabla$

However, in this case, the type of the free variables X' would depend on the freshness equations, and we would have problems to define the translation of a substitution that would have to instantiate such variables. Therefore, for simplicity we opt for Definition 3.

Lemma 2. *Given a nominal unification problem P, its translation $[\![P]\!]$ is a higher-order pattern unification problem.*
Moreover, the size of $[\![P]\!]$ is bounded by the square of the size of P.

Finally, we have to translate solutions of nominal unification problems.

Definition 4. *Let $\langle a_1, \ldots, a_n \rangle$ be an ordered list of atoms occurring in the nominal unification problem. The translation function is defined on solutions of nominal unification problems inductively as follows.*

$$[\![\langle \nabla, \sigma \rangle]\!] = \bigcup_{X \in Dom(\sigma)} \left[X' \mapsto \lambda a'_1. \cdots \lambda a'_n . [\![\sigma(X)]\!]_\nabla \right]$$

Remark 3. Notice that, if P contains freshness equations, then the set of free variables of $[\![P]\!]$ is bigger than the set of unknowns in P (we have a new free variable, called Y', for every freshness equation). However, σ and $[\![\langle \nabla, \sigma \rangle]\!]$ have equivalent domains. This would imply that, if $\langle \nabla, \sigma \rangle$ is a solution of P, we will have to extend $[\![\langle \nabla, \sigma \rangle]\!]$, by instantiating also the variables Y', to get a unifier of $[\![P]\!]$. If we had translated nominal unification problems as described in Remark 2, we would obtain a pattern unification problem with equivalent sets of variables. But, we would lose simplicity in other proofs.

We start by proving the following two technical lemmas.

Lemma 3. *For any freshness environment ∇, nominal terms t, u, and atom a, we have*

1. $\nabla \vdash a \# t$ *if, and only if,* $a' \notin FV(\llbracket t \rrbracket_\nabla)$, *and*
2. $\nabla \vdash t \approx u$ *if, and only if,* $\llbracket t \rrbracket_\nabla =_\alpha \llbracket u \rrbracket_\nabla$.

Proof. The first statement can be proved by routine induction on t and its translation. Notice that atoms are translated "nominally" into variables and that the binding structure is also identically translated, hence, the freshness of an atom a corresponds to the free occurrence of its variable counterpart a'. We here only comment the case $t = \pi \cdot X$, in this case, $\llbracket \pi \cdot X \rrbracket_\nabla = X'(\pi \cdot a_1)' \ldots (\pi \cdot a_m)'$, where $a_i \# X \notin \nabla$, for any $i \in \{1..m\}$. Therefore, we can establish the following sequence of equivalences $\nabla \vdash a \# \pi \cdot X$ iff $\pi^{-1} \cdot a \# X \in \nabla$ iff $\pi^{-1} \cdot a \notin \{a_1, \ldots, a_m\}$ iff $a \notin \{\pi \cdot a_1, \ldots, \pi \cdot a_m\}$ iff $a' \notin FV(X'(\pi \cdot a_1)' \ldots (\pi \cdot a_m)')$ iff $a' \notin FV(\llbracket \pi \cdot X \rrbracket)$.

The proof of the second statement can be done by induction on the equivalence $t \approx u$. We only comment the equivalence between suspensions: $\pi \cdot X \approx \pi' \cdot X$. Notice that, $\pi \cdot X \approx \pi' \cdot X$ if, and only if, for all atoms a such that $\pi \cdot a \neq \pi' \cdot a$, we have $a \# X \in \nabla$. This condition is equivalent to: the bound variables $(\pi \cdot a)'$ and $(\pi' \cdot a)'$ are passed as a parameter to X' in $\llbracket \pi \cdot X \rrbracket_\nabla$ and $\llbracket \pi' \cdot X \rrbracket_\nabla$ only when $\pi \cdot a = \pi' \cdot a$. Finally, this condition is equivalent to $\llbracket \pi \cdot X \rrbracket_\nabla = \llbracket \pi' \cdot X \rrbracket_\nabla$. □

Lemma 4. *For any freshness environment ∇, nominal terms t, and nominal substitution σ, we have* $\llbracket \langle \nabla, \sigma \rangle \rrbracket(\llbracket t \rrbracket_\emptyset) = \llbracket \sigma(t) \rrbracket_\nabla$.[6]

Proof. Again this lemma can be proved by structural induction on t. We only sketch the suspension case. Let $t = \pi \cdot X$. We have the equalities:

$$\begin{aligned}
\llbracket \langle \nabla, \sigma \rangle \rrbracket(\llbracket \pi \cdot X \rrbracket_\emptyset) &= [\ldots, X' \mapsto \lambda a'_1 \ldots \lambda a'_n \,.\, \llbracket \sigma(X) \rrbracket_\nabla, \ldots](X'(\pi \cdot a_1)' \ldots (\pi \cdot a_n)') \\
&= (\lambda a'_1 \ldots \lambda a'_n \,.\, \llbracket \sigma(X) \rrbracket_\nabla)(\pi \cdot a_1)' \ldots (\pi \cdot a_n)' \\
&= \llbracket [a_1 \mapsto \pi \cdot a_1, \ldots, a_n \mapsto \pi \cdot a_n] \sigma(X) \rrbracket_\nabla \\
&= \llbracket \pi \cdot \sigma(X) \rrbracket_\nabla \\
&= \llbracket \sigma(\pi \cdot X) \rrbracket_\nabla
\end{aligned}$$

□

From these two lemmas we can prove the following result and corollary.

Theorem 1. *For any freshness environment ∇, nominal unification problem P, and nominal substitution σ, we have that:*

$\langle \nabla, \sigma \rangle$ solves the nominal unification problem P, if, and only if, there exists an extension of $\llbracket \langle \nabla, \sigma \rangle \rrbracket$, for the variables of $\llbracket P \rrbracket$ not occurring in P, that solves the pattern unification problem $\llbracket P \rrbracket$.

Proof. The pair $\langle \nabla, \sigma \rangle$ solves P iff

$$\begin{array}{ll}
\nabla \vdash a_i \# \sigma(t) & \text{for all } a_i \#^? t \in P \\
\nabla \vdash \sigma(t) \approx \sigma(u) & \text{for all } t \stackrel{?}{\approx} u \in P
\end{array}$$

[6] When we write = between λ-terms, we mean that they are $\alpha\beta\eta$-equivalent, i.e. that have the same η-long β-normal form.

By Lemma 3 this is equivalent to:

$$\begin{array}{ll} a' \notin FV([\![\sigma(t)]\!]_\nabla) & \text{for all } a \#^? t \in P \\ [\![\sigma(t)]\!]_\nabla =_\alpha [\![\sigma(u)]\!]_\nabla & \text{for all } t \stackrel{?}{\approx} u \in P \end{array}$$

By Lemma 4 this is equivalent to:

$$\begin{array}{ll} a'_i \notin FV([\![\langle\nabla,\sigma\rangle]\!][\![t]\!]_\emptyset) & \text{for all } a_i \#^? t \in P \\ [\![\langle\nabla,\sigma\rangle]\!][\![t]\!]_\emptyset = [\![\langle\nabla,\sigma\rangle]\!][\![u]\!]_\emptyset & \text{for all } t \stackrel{?}{\approx} u \in P \end{array}$$

Since we avoid variable capture, this is equivalent to:

$$\begin{array}{ll} \Big[Y' \mapsto \lambda a'_1 \ldots a'_{i-1} a'_{i+1} \ldots a'_n.[\![\langle\nabla,\sigma\rangle]\!][\![t]\!]_\emptyset\Big]\Big(\lambda a'_1 \ldots a'_n.Y' a'_1 \ldots a'_{i-1}\, a'_{i+1} \ldots a'_n\Big) \stackrel{?}{=} & \\ \quad \stackrel{?}{=} [\![\langle\nabla,\sigma\rangle]\!]\Big(\lambda a'_1 \ldots a'_n.[\![t]\!]_\emptyset\Big) & \text{for all } a_i \#^? t \in P \\ [\![\langle\nabla,\sigma\rangle]\!]\Big(\lambda a'_1.\ldots.\lambda a'_n.[\![t]\!]_\emptyset\Big) \stackrel{?}{=} [\![\langle\nabla,\sigma\rangle]\!]\Big(\lambda a'_1.\ldots.\lambda a'_n.[\![u]\!]_\emptyset\Big) & \text{for all } t \stackrel{?}{\approx} u \in P \end{array}$$

Finally, this means that the following extension solves $[\![P]\!]$.

$$\sigma' = [\![\langle\nabla,\sigma\rangle]\!] \cup \bigcup_{Y'} [Y' \mapsto \lambda a'_1.\ldots.\lambda a'_{i-1}.\lambda a'_{i+1}.\ldots.\lambda a'_n.[\![\langle\nabla,\sigma\rangle]\!][\![t]\!]_\emptyset]$$

□

Corollary 1. *If the nominal unification problem P is solvable, then the higher-order pattern unification problem $[\![P]\!]$ is solvable.*

5 The Reverse Translation

Notice that Theorem 1 is not enough to prove that, if $[\![P]\!]$ is solvable, then P is solvable. We still have to prove that if $[\![P]\!]$ is solvable, then for some solution σ' of $[\![P]\!]$ we can build a nominal solution $\langle\nabla,\sigma\rangle$ of P. This is the main objective of this section. Taking into account that $[\![P]\!]$ is a higher-order pattern unification problem, and that these problems are unitary, we will prove something stronger: if $[\![P]\!]$ is solvable, then $[\![\sigma']\!]^{-1}$ is defined for some most general unifier σ' of $[\![P]\!]$.

In the following example we note that in the solution of pattern unification problems it is important to save *names* of bound variables.

Example 6. Consider the nominal problem $a.X \stackrel{?}{\approx} a.f\,(b.Y)$. Its translation is $\lambda a'.\lambda b'.\lambda a'.X'\,a'\,b' \stackrel{?}{=} \lambda a'.\lambda b'.\lambda a'.f'\,(\lambda b'.Y'\,a'\,b')$. An α-conversion results in $\lambda a'.\lambda b'.\lambda c'.X'\,c'\,b' \stackrel{?}{=} \lambda a'.\lambda b'.\lambda c'.f'\,(\lambda d'.Y'\,c'\,d')$ and it shows that the parameters of X' and Y' are in fact different. A most general solution is $[X' \mapsto \lambda c'.\lambda b'.f'(\lambda d'.Y'\,c'\,d')]$. Since Y is translated as $Y'\,a'\,b'$, we have to translate back $Y'\,c'\,d'$ as $(a\,c)(d\,b)\cdot Y$. And, since $[X \mapsto t]$ is translated as $[X' \mapsto \lambda a'.\lambda b'.[\![t]\!]_\nabla]$, we have to translate back $[X' \mapsto \lambda c'.\lambda b'.[\![t]\!]_\nabla$ as $[X \mapsto (a\,c)\cdot t]$. Therefore, our pattern unifier can be translated back as $[X \mapsto (a\,c)\cdot f(d.(a\,c)(d\,b)\cdot Y)]$. However, the list of atoms is fixed as the list of atoms occurring in the problem, therefore, we know how to translate a and b as a' and b' and vice versa, but we do not know how to translate back c' and d' (here it is done introducing new atoms).

To avoid the problem discussed in the previous example, we will be cautious with the α-conversions. Lemma 5 justifies why we can avoid the use of other

atoms but the ones occurring in the original nominal problem. Lemma 6 will be also necessary, and also describes some properties of pattern unifiers.

Lemma 5. *For any solvable pattern unification problem, there exists a most general unifier that does not use other names and types for bound variables than the ones already used in the problem.*

Proof. It can be proved by inspection of the transformations rules in [Mil91,Nip93], that describe a sound and complete algorithm for pattern unification. These transformations introduce fresh variables and new lambda binders. However, the names of the new bound variables can always be chosen to coincide with names of already existing bound variables with the same type. □

Lemma 6. *For every pattern unification problem P and most general unifier σ, if X occurs free in σ, then for every type of an argument of X, there exists a variable Y in P with an argument of the same type, and there exists a variable Z in P with the same return type as X.*

Proof. Like for the previous lemma, we can analyze the transformations rules in [Mil91,Nip93]. It can be seen that when we introduce a fresh variable with type $\tau_1 \to \ldots \to \tau_n \to \tau_0$, there exist already another variable with type $\tau'_1 \to \ldots \to \tau'_m \to \tau'_0$, such that $\{\tau_1, \ldots, \tau_n\} \subseteq \{\tau'_1, \ldots, \tau'_m\}$ and $\tau_0 = \tau'_0$. The only exception is in rigid-flexible pairs (imitation rule), where fresh variables not satisfying these properties are introduced. But it is easy to see that they always disappear from the solution. □

Even using a solution of the higher-order pattern unification problem with a restricted use of names of bound variables, we still have some freedom to select the unifier $[\![\sigma']\!]^{-1}$. This is reflected in the way we translate back applications of free variables, i.e. in the definition of the list of variable indices $L_{X'}$ for every free variable X'.

Definition 5. *Let $\langle a_1, \ldots, a_n \rangle$ be the fixed list of atoms. For every free variable $X' : \tau_1 \to \ldots \tau_m \to \tau_0$ we define the list of indexes of atoms $L_{X'} = \langle i_1, \ldots i_m \rangle$ such that a_{i_j} has sort $[\![\tau_j]\!]^{-1}$, [7] for $j = 1, \ldots, m$, and we also define the corresponding freshness environment $\nabla_{X'} = \{a_i \# X \mid i \notin L_{X'}\}$.*

Definition 6. *Let $\langle a_1, \ldots, a_n \rangle$ be the fixed list of atoms. The back-translation function is defined on λ-terms in η-long β-normal form as follows:*

$$
\begin{array}{ll}
[\![\lambda p.\, p\, t_1\, t_2]\!]^{-1} = \langle [\![t_1]\!]^{-1}, [\![t_2]\!]^{-1} \rangle & \text{if } p : \tau_1 \to \tau_2 \to \top \\
[\![a']\!]^{-1} = a & \\
[\![f'\, t]\!]^{-1} = f\, [\![t]\!]^{-1} & \\
[\![\lambda a'.t]\!]^{-1} = a\, .\, [\![t]\!]^{-1} & \text{if } a' \text{ is base typed} \\
[\![X'\, a'_{j_1} \ldots a'_{j_m}]\!]^{-1} = \begin{pmatrix} a_{i_1} & \cdots & a_{i_m} \\ a_{j_1} & \cdots & a_{j_m} \end{pmatrix} \cdot X & \text{where } L_{X'} = \langle i_1, \ldots i_m \rangle \\
 & \text{and } \{a_{j_1}, \cdots, a_{j_m}\} \subseteq \{a_1, \ldots, a_n\}
\end{array}
$$

[7] Notice that Lemma 6 ensures that, even for the introduced fresh variables, for every type τ_j there exists at least one atom a_{i_j} satisfying $a_{i_j} : [\![\tau_j]\!]^{-1}$. Notice also that for every X', we freely choose one among many possible lists $L_{X'}$.

where a' is a bound variable with name a, f' is the constant associated to the function symbol f, either X' is the free variable associated to X, or if X' is a fresh variable then X is a fresh unknown, and the permutation is supposed to be decomposed in terms of transpositions (swappings).

Notice that the back-translation function is not defined for all λ-terms, even for all higher-order patterns. In particular, $[\![\lambda x.t]\!]^{-1}$ is not defined when x is not base typed nor returns something of type $\top$, or $[\![x\,t]\!]^{-1}$ is not defined when x is a bound variable.

Definition 7. *The back-translation function is defined on substitutions inductively as follows.*

$$[\![\sigma']\!]^{-1} = \left\langle \bigcup_{\substack{X' \in Dom(\sigma') \\ X' : \nu'_1 \to \ldots \to \nu'_n \to \delta'}} \left[X \mapsto [\![\sigma'(X')\, a'_1 \ldots a'_n]\!]^{-1} \right], \bigcup_{Z' \ occurs \ in \ \sigma'} \nabla_{Z'} \right\rangle$$

Notice that the back-translation only translates those instantiations affecting variables with type $\nu'_1 \to \ldots \to \nu'_n \to \delta'$. When σ' is a unifier of a problem $[\![P]\!]$, then it contains in $Dom(\sigma')$ variables X' associated to a nominal variable X, that satisfies this condition, and variables Y' resulting from the translation of freshness equations, that do not satisfy the condition because they have type $\nu'_1 \to \ldots \to \nu'_{i-1} \to \nu'_{i+1} \to \ldots \to \nu'_n \to \delta'$. These second type of variables have not back-translation and do not occur in the domain of $[\![\sigma']\!]^{-1}$.

Example 7. The nominal unification problem

$$P = \{a.a.X \stackrel{?}{\approx} c.a.X \ , \ a.b.X \stackrel{?}{\approx} b.a.(a\,b) \cdot X \ , \ a \#^{?} X\}$$

is translated as

$$\begin{array}{rl} [\![P]\!] = \{ & \lambda a'.\lambda b'.\lambda c'.\lambda a'.\lambda a'.\ X'\,a'\,b'\,c' \stackrel{?}{=} \lambda a'.\lambda b'.\lambda c'.\lambda c'.\lambda a'.\ X'\,a'\,b'\,c' \ , \\ & \lambda a'.\lambda b'.\lambda c'.\lambda a'.\lambda b'.\ X'\,a'\,b'\,c' \stackrel{?}{=} \lambda a'.\lambda b'.\lambda c'.\lambda b'.\lambda a'.\ X'\,b'\,a'\,c' \ , \\ & \lambda a'.\lambda b'.\lambda c'.\ Y'\,b'\,c' \stackrel{?}{=} \lambda a'.\lambda b'.\lambda c'.\ X'\,a'\,b'\,c'\} \end{array}$$

The pattern unifier is

$$\sigma' = [X' \mapsto \lambda a'.\lambda b'.\lambda c'.\ Z'\,b' \ , \ Y' \mapsto \lambda b'.\lambda c'.\ Z'\,b']$$

Fixed $\langle a, b, c\rangle$ as the fixed list of atoms, and taking $L_{Z'} = \langle 2\rangle$, the nominal solution is

$$\langle \nabla, \sigma\rangle = [\![\sigma']\!]^{-1} = \langle \{a \# Z, c \# Z\} \ , \ [X \mapsto Z]\rangle$$

They satisfy the relation

$$\sigma' = [\![\langle \nabla, \sigma\rangle]\!] \cup [Y' \mapsto \lambda b'.\lambda c'.\ Z'\,b']$$

Notice that a and b are of the same sort. Therefore, we could take $L_{Z'} = \langle 1\rangle$. Then, we would obtain another "equivalent" nominal unifier:

$$\langle \nabla, \sigma\rangle = [\![\sigma']\!]^{-1} = \langle \{b \# Z, c \# Z\} \ , \ [X \mapsto (a\,b) \cdot Z]\rangle$$

Lemma 7. *For every λ-substitution σ', if $[\![\sigma']\!]^{-1}$ exists, then $[\![[\![\sigma']\!]^{-1}]\!]$ is extensible to σ'.*

Proof. Straightforward from the definition of $[\![\ ]\!]$ and $[\![\ ]\!]^{-1}$. □

Lemma 8. *For every nominal unification problem P, if the pattern unification problem $[\![P]\!]$ is solvable, then it has a most general unifier σ' such that $[\![\sigma']\!]^{-1}$ is defined.*

Proof. By Lemma 5, there exists a most general unifier that does not use bound variables with other names and types than the ones already used in the original problem. This ensures that we can always translate back bound variables a' as the atom with the same name a. For the same reason, in all λ-expressions $\lambda x.t$ the bound variable x will have type $\tau_1 \to \tau_2 \to \top$ (for pairs) or base type ν'_i, which will ensure that its translation back is possible.

By Lemma 6, since all free variables in the original problem $[\![P]\!]$ have type of the form $\nu_1 \to \ldots \to \nu_n \to \delta$, all free variables in the unifier will have type of the form $X' : \nu_{i_i} \to \ldots \to \nu_{i_m} \to \delta$. This ensures the existence of $L_{X'} = \{i_1, \ldots, i_m\}$, as well as the translation back of suspensions.

Finally, we will never be forced to translate back terms of the form $at_1 \ldots t_m$ where a is a bound variable, because in $[\![P]\!]$, hence in σ', all bound variables are base typed. □

Theorem 2. *For every nominal unification problem P, if the pattern unification problem $[\![P]\!]$ is solvable, then P is solvable.*

Proof. By Lemma 8, if $[\![P]\!]$ is solvable then there exist a most general unifier σ' of such that $\langle \nabla, \sigma \rangle = [\![\sigma']\!]^{-1}$ is defined. By Lemma 7, we have $[\![\langle \nabla, \sigma \rangle]\!]$ is extensible (by instantiating all the variables of $[\![P]\!]$ not corresponding to any variable in P) to σ', which solves $[\![P]\!]$. Hence, by Theorem 1, $\langle \nabla, \sigma \rangle$ solves P. □

Corollary 2. *Nominal Unification is quadratic reducible to Higher-Order Pattern Unification.*
Nominal Unification can be decided in quadratic deterministic time.

6 Conclusion

The paper describes a precise quadratic reduction from Nominal Unification to Higher-order Pattern Unification. This helps to better understand the semantics of the nominal binding and permutations in comparison with λ-binding and α-conversion. Moreover, using the result of linear time decidability for Higher-Order Patterns Unification [Qia96], we prove that Nominal Unification can be decided in quadratic time. It seems not difficult to prove that the translation and the back-translation function that we present transform most general nominal unifiers into most general higher-order patter unifiers and vice versa.

References

[CF07] C. Calvès and M. Fernández. Implementing nominal unification. *ENTCS*, 176(1):25–37, 2007.

[Che05] J. Cheney. Relating higher-order pattern unification and nominal unification. In *Proc. of the 19th Int. Work. on Unification, UNIF'05*, pages 104–119, 2005.

[CP07] R. A. Clouston and A. M. Pitts. Nominal equational logic. *ENTCS*, 1496:223–257, 2007.

[CU04] J. Cheney and C. Urban. α-prolog: A logic programming language with names, binding and α-equivalence. In *Proc. of the 20th Int. Conf. on Logic Programming,ICLP'04*, volume 3132 of *LNCS*, pages 269–283, 2004.

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

[FG05] M. Fernández and M. Gabbay. Nominal rewriting with name generation: abstraction vs. locality. In *Proc. of the 7th Int. Conf. on Principles and Practice of Declarative Programming, PPDP'05*, pages 47–58, 2005.

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

[GC04] M. Gabbay and J. Cheney. A sequent calculus for nominal logic. In *Proc. of the 19th Symp. on Logic in Computer Science, LICS'04*, pages 139–148, 2004.

[Gol81] W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[GP99] M. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In *Proc. of the 14th Symp. on Logic in Computer Science, LICS'99*, pages 214–224, 1999.

[Lev98] J. Levy. Decidable and undecidable second-order unification problems. In *Proc. of the 9th Int. Conf. on Rewriting Techniques and Applications, RTA'98*, volume 1379 of *LNCS*, pages 47–60, 1998.

[Luc72] C. L. Lucchesi. The undecidability of the unification problem for third-order languages. Technical Report CSRR 2059, Dept. of Applied Analysis and Computer Science, Univ. of Waterloo, 1972.

[LV00] J. Levy and M. Veanes. On the undecidability of second-order unification. *Information and Computation*, 159:125–150, 2000.

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

[Nip93] T. Nipkow. Functional unification of higher-order patterns. In *Proc. of the 8th Symp. on Logic in Computer Science, LICS'93*, pages 64–74, 1993.

[Pit01] A. M. Pitts. Nominal logic: A first order theory of names and binding. In *Proc. of the 4th Int. Symp. on Theoretical Aspects of Computer Software, TACS'01*, volume 2215 of *LNCS*, pages 219–242, 2001.

[Pit03] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.

[Qia96] Z. Qian. Unification of higher-order patterns in linear time and space. *J. of Logic and Computation*, 6(3):315–341, 1996.

[UC05] C. Urban and J. Cheney. Avoiding equivariance in alpha-prolog. In *Proc. of the Int. Conf. on Typed Lambda Calculus and Applications, TLCA'05*, volume 3461 of *LNCS*, pages 401–416, 2005.

[UPG03] C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. In *Proc. of the 17th Int. Work. on Computer Science Logic, CSL'03*, volume 2803 of *LNCS*, pages 513–527, 2003.

[UPG04] C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323:473–497, 2004.