Nominal Unification from a Higher-Order Perspective

JORDI LEVY, IIIA - CSIC
MATEU VILLARET, Universitat de Girona

Nominal logic is an extension of first-order logic with equality, name-binding, renaming via name-swapping and freshness of names. Contrarily to lambda-terms, in nominal terms, bindable names, called atoms, and instantiable variables are considered as distinct entities. Moreover, atoms are capturable by instantiations, breaking a fundamental principle of the lambda-calculus. Despite these differences, nominal unification can be seen from a higher-order perspective. From this view, we show that nominal unification can be quadratically reduced to a particular fragment of higher-order unification problems: higher-order pattern unification. We also prove that the translation preserves most generality of unifiers.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Lambda Calculus and Related Systems*

General Terms: Unification, Lambda-calculus, Nominal

Additional Key Words and Phrases: Higher-Order Pattern Unification, Nominal Unification

ACM Reference Format:
Levy, J., Villaret, M. 2011. Nominal Unification from a Higher-Order Perspective ACM Trans. Comput. Logic V, N, Article A (January YYYY), 30 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Nominal terms were introduced by Urban et al. [Urban et al. 2003; 2004], and are based on nominal sets semantics [Gabbay and Pitts 1999; Pitts 2001; Gabbay and Pitts 2001; Pitts 2003]. They are characterized by a syntactic distinction between *atoms* (that roughly correspond to the notion of bound variables) and *variables* (that would correspond to free variables). Hence, binders can only bind atoms and only variables can be instantiated. These first works have inspired a sequel of papers where bindings and freshness are introduced in other areas, like nominal algebra [Gabbay and Mathijssen 2006; 2007; 2009], equational logic [Clouston and Pitts 2007], rewriting [Fernández and Gabbay 2005; 2007], unification [Urban et al. 2003; 2004], and Prolog [Cheney and Urban 2004; Urban and Cheney 2005].

This paper is concerned with *Nominal Unification*, the problem of deciding if two nominal terms can be made α-equivalent by instantiating their variables by nominal terms. In these instantiations of variables, it is allowed to *capture* atoms. Urban et al. [2003; 2004] describe a sound and complete, but inefficient (exponential), algorithm for nominal unification. Fernández and Gabbay [2005] extend this algorithm

This research has been partially founded by the CICYT research projects TASSAT (TIN2010-20967-C04-01) and SUROS (TIN2008-04547).
Author's addresses: J. Levy, Artificial Intelligence Research Institute (IIIA), Campus UAB, Bellaterra, Spain; M. Villaret, Departament d'Informàtica i Matemàtica Aplicada (IMA), Universitat de Girona (UdG), Av. Montilivi s/n, ed. P4, Girona, Spain.

to deal with the new-quantifier and locality. Nominal Logic's equivariance property suggested to Cheney [2005a] a stronger form of unification called equivariant unification. He proves that equivariant unification and matching are NP-hard problems. Another variant of nominal unification is permissive unification, defined by Dowek et al. [2009; 2010], that is also reducible to Higher-Order Pattern Unification. Calvès and Fernández [2007] describe a direct but exponential implementation of a nominal unification algorithm in Maude, and in [Calvès and Fernández 2008] a polynomial implementation, based on a graph representation of terms, and a lazy propagation of swappings.

Levy and Villaret [2008] prove that Nominal Unification can be quadratically reduced to Higher-Order Pattern Unification. The present paper is an extension of this preliminary paper, where we have simplified the reduction by removing freshness equations (Section 4), and we have included the proof of some important properties of pattern unifiers (Section 6). In particular, we prove that most general higher-order pattern unifiers can be written without using other bound-variable names than the ones used in the presentation of the unification problem. Moreover, we establish a precise correspondence between most general nominal unifiers and most general pattern unifiers (Section 8). In fact, Sections 4, 6 and 8 are completely new in this extended version. Recently, Calvès [2010], and Levy and Villaret [2010] have independently found a quadratic nominal unification algorithms based on the Paterson and Wegman's linear first-order unification algorithm [Paterson and Wegman 1978].

The use of α-equivalence and binders in nominal terms immediately suggests to look at nominal unification from a higher-order perspective, the one that we adopt in this paper. Some intuitions about this relation were already roughly described by Urban et al. [2004]. Cheney [2005b] reduces higher-order pattern unification to nominal unification (here we prove the opposite reduction).

The main benefit of nominal terms, compared to lambda-terms, is that they allow the use of binding and α-equivalence without the other difficulties associated with the λ-calculus, like the β and η equivalence. In particular, with respect to unification, we have that nominal unification is unitary (most general unifiers are unique) and decidable [Urban et al. 2003; 2004], whereas higher-order unification is undecidable and infinitary [Lucchesi 1972; Goldfarb 1981; Levy 1998; Levy and Veanes 2000; Levy and Villaret 2009].

In this paper we fully develop the study of nominal unification from a higher-order view. We show that *full* higher-order unification is not needed, and *Higher-order Pattern Unification* suffices to encode Nominal Unification. This subclass of problems was introduced by Miller [1991]. Contrarily to general higher-order unification, higher-order pattern unification is decidable and unitary [Miller 1991; Nipkow 1993].

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,
(2) unification is performed modulo α-equivalence, instead of the usual α and β-equivalence,
(3) instantiations for variables are allowed to capture atoms, contrarily to the standard higher-order definition of capture-avoiding substitution, and
(4) apart from the usual equality predicate, we use a *freshness* predicate $\mathsf{a} \# \mathsf{t}$ with the intended meaning: the atom a does not occur free in t.

The third point is the key 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 atoms into bound variables, and variables into free variables with the list of atoms that they can *capture* as arguments. The first point will ensure that, since variables do not have parameters, after translation, the only arguments of free variables will be list of pairwise distinct bound variables, hence higher-order patterns. Moreover, since bound variables will be first-order typed, and constants third-order typed, the translated problems will be *second-order* patterns. The second point is not a difficulty. Since all nominal variables are first-order typed, their instantiation does not introduce β-redexes. Finally, the fourth point can also be overcome by translating freshness equations into equality equations, as described in Section 4.

The remainder of the paper proceeds 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 prove that freshness equations can be linearly translated into equality equations. In Section 5, we show how to translate a nominal unification problem into a higher-order patterns unification problem. Then, after proving some properties of Higher-Order Pattern Unification in Section 6, we prove that this translation is effectively a quadratic time reduction, in Section 7. In Section 8, we establish a correspondence between nominal unifiers and pattern unifiers of the translated problems. In particular, we prove that the translation function and its inverse are monotone w.r.t. the more general relation, and both translate most general unifiers into most general unifiers. We conclude in Section 9.

2. PRELIMINARIES

In this section we present some basic definitions of Nominal Unification and Higher-Order Pattern Unification. We will use two distinct typographic fonts to represent nominal terms and lambda-terms.

2.1. Nominal Unification

Nominal terms contain *variables* and *atoms*. Only variables may be instantiated, and only atoms may be bound. They roughly correspond to the notions of free and bound variables in λ-calculus, respectively, but are considered as completely different entities. However, atoms are not necessarily bound, and when they occur free, they are not instantiable.

In the following we introduce some basic definitions of nominal unification, they are imported from [Urban et al. 2003; 2004]. In *nominal signatures* we have *sorts of atoms* (typically ν) and *sorts of data* (typically δ) as disjoint sets. *Atoms* (typically $\mathsf{a}, \mathsf{b}, \dots$) have one of the sorts of atoms. *Variables*, also called *unknowns*, (typically $\mathsf{X}, \mathsf{Y}, \dots$) have a sort of atom or sort of data, i.e. of the form $\nu \mid \delta$. *Function symbols* (typically $\mathsf{f}, \mathsf{g}, \dots$) have an *arity* of the form $\tau_1 \times \dots \times \tau_n \to \delta$, where δ is a sort of data and τ_i are sorts given by the grammar $\tau ::= \nu \mid \delta \mid \langle\nu\rangle\tau$. Abstractions have sorts of the form $\langle\nu\rangle\tau$.

Nominal terms (typically $\mathsf{t}, \mathsf{u}, \dots$) are given by the grammar:

$$\mathsf{t} ::= \mathsf{f}(\mathsf{t}_1, \dots, \mathsf{t}_\mathsf{n}) \mid \mathsf{a} \mid \mathsf{a.t} \mid \pi \cdot \mathsf{X}$$

where f is a n-ary function symbol, a is an atom, π is a permutation (finite list of swappings), and X is a variable. They are called respectively *application*, *atom*, *abstraction* and *suspension*. The set of variables of a term t is denoted by $\mathrm{Vars}(\mathsf{t})$.

A *swapping* $(\mathsf{a}\,\mathsf{b})$ is a pair of atoms of the same sort. The effect of a swapping over an atom is defined by $(\mathsf{a}\,\mathsf{b}) \cdot \mathsf{a} = \mathsf{b}$ and $(\mathsf{a}\,\mathsf{b}) \cdot \mathsf{b} = \mathsf{a}$ and $(\mathsf{a}\,\mathsf{b}) \cdot \mathsf{c} = \mathsf{c}$, when $\mathsf{c} \neq \mathsf{a}, \mathsf{b}$. For the rest of terms the extension is straightforward, in particular, $(\mathsf{a}\,\mathsf{b}) \cdot (\mathsf{c.t}) = \big((\mathsf{a}\,\mathsf{b}) \cdot \mathsf{c}\big).\big((\mathsf{a}\,\mathsf{b}) \cdot \mathsf{t}\big)$.

A *permutation* is a (possibly empty) sequence of swappings. Its effect is defined inductively by $(\mathsf{a}_1\,\mathsf{b}_1) \dots (\mathsf{a}_\mathsf{n}\,\mathsf{b}_\mathsf{n}) \cdot \mathsf{t} = (\mathsf{a}_1\,\mathsf{b}_1) \cdot \Big((\mathsf{a}_2\,\mathsf{b}_2) \dots (\mathsf{a}_\mathsf{n}\,\mathsf{b}_\mathsf{n}) \cdot \mathsf{t}\Big)$. 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 variables with a permutation of atoms waiting to be applied once the variable is instantiated. Occurrences of an atom a are said to be *bound* if they are in the scope of an abstraction of a, otherwise are said to be *free*.

Substitutions are finite sets of pairs $[\mathsf{X}_1 \mapsto \mathsf{t}_1, \dots, \mathsf{X}_n \mapsto \mathsf{t}_n]$ where X_i and t_i have the same sort, and the X_i are pairwise distinct variables. They can be extended to sort-respecting functions between terms, and behave like in first-order terms, hence allowing atom capture (see [Urban et al. 2004]). For instance $[\mathsf{X} \mapsto \mathsf{a}]\mathsf{a}.\mathsf{X} = \mathsf{a}.\mathsf{a}$. Remember that when applying a substitution to a suspension, the permutation is immediately applied, for instance

$$[\mathsf{X} \mapsto \mathsf{g}(\mathsf{a})]\mathsf{f}\big((\mathsf{a}\,\mathsf{b})\cdot\mathsf{X}, \mathsf{X}\big) = \mathsf{f}\big((\mathsf{a}\,\mathsf{b})\cdot\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a})\big) = \mathsf{f}\big(\mathsf{g}((\mathsf{a}\,\mathsf{b})\cdot\mathsf{a}), \mathsf{g}(\mathsf{a})\big) = \mathsf{f}\big(\mathsf{g}(\mathsf{b}), \mathsf{g}(\mathsf{a})\big)$$

The domain of a substitution $\sigma = [\mathsf{X}_1 \mapsto \mathsf{t}_1, \dots, \mathsf{X}_n \mapsto \mathsf{t}_n]$ is $\mathrm{Dom}(\sigma) = \{\mathsf{X}_1, \dots, \mathsf{X}_n\}$. For convenience we consider $\mathrm{Dom}([\mathsf{X} \mapsto \mathsf{X}]) = \{\mathsf{X}\} \neq \{\mathsf{Y}\} = \mathrm{Dom}([\mathsf{Y} \mapsto \mathsf{Y}])$, although both substitutions have the same effect when applied to any term.[1] Composition of substitutions is defined by $\sigma_1 \circ \sigma_2 = [\mathsf{X} \mapsto \sigma_1(\sigma_2(\mathsf{X})) \mid \mathsf{X} \in \mathrm{Dom}(\sigma_1) \cup \mathrm{Dom}(\sigma_2)]$. The restriction of a substitution σ to a set of variables V, written $\sigma|_{\mathsf{V}}$, is defined as $\sigma|_{\mathsf{V}} = [\mathsf{X} \mapsto \sigma(\mathsf{X}) \mid \mathsf{X} \in \mathsf{V}]$.

A *freshness environment* (typically ∇) is a list of *freshness constraints* $\mathsf{a} \# \mathsf{X}$ stating that the instantiation of X cannot contain free occurrences of a.

The notion of *α-equivalence* between terms, written $\approx$, is defined by means of the following theory:

$$\frac{}{\nabla \vdash \mathsf{a} \approx \mathsf{a}}\ (\approx\text{-atom}) \qquad \frac{\mathsf{a}\#\mathsf{X} \in \nabla \text{ for all } \mathsf{a} \text{ such that } \pi\cdot\mathsf{a} \neq \pi'\cdot\mathsf{a}}{\nabla \vdash \pi\cdot\mathsf{X} \approx \pi'\cdot\mathsf{X}}\ (\approx\text{-susp.})$$

$$\frac{\nabla \vdash \mathsf{t}_1 \approx \mathsf{t}'_1 \qquad \cdots \qquad \nabla \vdash \mathsf{t}_n \approx \mathsf{t}'_n}{\nabla \vdash \mathsf{f}(\mathsf{t}_1, \dots, \mathsf{t}_n) \approx \mathsf{f}(\mathsf{t}'_1, \dots, \mathsf{t}'_n)}\ (\approx\text{-application})$$

$$\frac{\nabla \vdash \mathsf{t} \approx \mathsf{t}'}{\nabla \vdash \mathsf{a}.\mathsf{t} \approx \mathsf{a}.\mathsf{t}'}\ (\approx\text{-abst-1}) \qquad \frac{\mathsf{a} \neq \mathsf{a}' \quad \nabla \vdash \mathsf{t} \approx (\mathsf{a}\,\mathsf{a}')\cdot\mathsf{t}' \quad \nabla \vdash \mathsf{a}\#\mathsf{t}'}{\nabla \vdash \mathsf{a}.\mathsf{t} \approx \mathsf{a}'.\mathsf{t}'}\ (\approx\text{-abst-2})$$

where the *freshness* predicate $\#$ is defined by:

$$\frac{\mathsf{a} \neq \mathsf{a}'}{\nabla \vdash \mathsf{a}\#\mathsf{a}'}\ (\#\text{-atom}) \qquad \frac{(\pi^{-1}\cdot\mathsf{a}\,\#\mathsf{X}) \in \nabla}{\nabla \vdash \mathsf{a}\#\pi\cdot\mathsf{X}}\ (\#\text{-susp.})$$

$$\frac{\nabla \vdash \mathsf{a}\#\mathsf{t}_1 \qquad \cdots \qquad \nabla \vdash \mathsf{a}\#\mathsf{t}_n}{\nabla \vdash \mathsf{a}\#\mathsf{f}(\mathsf{t}_1, \dots, \mathsf{t}_n)}\ (\#\text{-application})$$

$$\frac{}{\nabla \vdash \mathsf{a}\#\mathsf{a}.\mathsf{t}}\ (\#\text{-abst-1}) \qquad \frac{\mathsf{a} \neq \mathsf{a}' \quad \nabla \vdash \mathsf{a}\#\mathsf{t}}{\nabla \vdash \mathsf{a}\#\mathsf{a}'.\mathsf{t}}\ (\#\text{-abst-2})$$

Their intended meanings are:

[1]We have adopted this definition motivated by Remark 5.8.

— $\nabla \vdash \mathsf{a} \mathbin{\#} \mathsf{t}$ holds if, for every substitution σ respecting the freshness environment ∇ (i.e. avoiding the atom captures forbidden by ∇), a is not free in $\sigma(\mathsf{t})$;
— $\nabla \vdash \mathsf{t} \approx \mathsf{u}$ holds if, for every substitution σ respecting the freshness environment ∇, t and u are α-equivalent.

A *nominal unification problem* (typically P) is a set of equations of the form $\mathsf{t} \overset{?}{\approx} \mathsf{u}$, or of the form $\mathsf{a} \#^? \mathsf{t}$, called *equality equations* and *freshness equations*, respectively.

A *solution* or *unifier* of a nominal problem P is a pair $\langle \nabla, \sigma \rangle$ satisfying $\nabla \vdash \mathsf{a} \mathbin{\#} \sigma(\mathsf{t})$, for all freshness equations $\mathsf{a} \#^? \mathsf{t} \in \mathsf{P}$, and $\nabla \vdash \sigma(\mathsf{t}) \approx \sigma(\mathsf{u})$, for all equality equations $\mathsf{t} \overset{?}{\approx} \mathsf{u} \in \mathsf{P}$. Later, in Section 5, we will also require solutions to satisfy $\mathrm{Dom}(\sigma) = \mathrm{Vars}(\mathsf{P})$. In Remark 5.8 we justify why this does not affect to solvability of nominal problems.

Given two substitutions σ_1 and σ_2, and two freshness environments ∇_1 and ∇_2, we say that $\nabla_2 \vdash \sigma_1(\nabla_1)$, if $\nabla_2 \vdash \mathsf{a} \mathbin{\#} \sigma_1(\mathsf{X})$ holds for each $\mathsf{a} \mathbin{\#} \mathsf{X} \in \nabla_1$; and we say that $\nabla_1 \vdash \sigma_1 \approx \sigma_2$, if $\nabla_1 \vdash \sigma_1(\mathsf{X}) \approx \sigma_2(\mathsf{X})$ holds for all $\mathsf{X} \in \mathrm{Dom}(\sigma_1) \cup \mathrm{Dom}(\sigma_2)$. Given a nominal unification problem P, we say that a solution $\langle \nabla_1, \sigma_1 \rangle$ is *more general* than another solution $\langle \nabla_2, \sigma_2 \rangle$, if there exists a substitution σ' satisfying $\nabla_2 \vdash \sigma'(\nabla_1)$ and $\nabla_2 \vdash \sigma' \circ \sigma_1|_{\mathrm{Dom}(\sigma_1) \cup \mathrm{Dom}(\sigma_2)} \approx \sigma_2$. As usual, we say that a solution σ is *most general* if, for any other solution σ' more general than σ, we have also that σ is also more general than σ'. Most general nominal unifiers are *unique*, in the usual sense: if σ_1 and σ_2 are both most general, then σ_1 is more general than σ_2, and vice versa.

Example 2.1. The solutions of the equation $\mathsf{a}.\mathsf{X} \overset{?}{\approx} \mathsf{b}.\mathsf{Y}$ can not instantiate X with terms containing free occurrences of the atom b, for instance if we apply the substitution $[\mathsf{X} \mapsto \mathsf{b}]$ to both sides of the equation we get $[\mathsf{X} \mapsto \mathsf{b}](\mathsf{a}.\mathsf{X}) = \mathsf{a}.\mathsf{b}$, for the left hand side, and $[\mathsf{X} \mapsto \mathsf{b}](\mathsf{b}.\mathsf{Y}) = \mathsf{b}.\mathsf{Y}$, for the right hand side, and obviously $\mathsf{a}.\mathsf{b} \overset{?}{\approx} \mathsf{b}.\mathsf{Y}$ is unsolvable.

A most general solution of this equation is $\langle \{\mathsf{b}\#\mathsf{X}\}, \mathsf{Y} \mapsto (\mathsf{a}\,\mathsf{b}) \cdot \mathsf{X}] \rangle$. Another most general solution is $\langle \{\mathsf{a}\#\mathsf{Y}\}, [\mathsf{X} \mapsto (\mathsf{a}\,\mathsf{b}) \cdot \mathsf{Y}] \rangle$. Notice that the first unifier is equal to the second composed with $\sigma' = [\mathsf{Y} \mapsto (\mathsf{a}\,\mathsf{b}) \cdot \mathsf{X}]$, hence the second one is more general than the first one. Similarly, the first one is more general that the second one. Hence, both are equivalent.

2.2. Higher-Order Pattern Unification

In the following we introduce some basic definitions of higher-order pattern unification. These definitions can also be found in [Nipkow 1993; Dowek 2001]. In higher-order signatures we have *types* constructed from a set of *basic types* (typically $\delta, \nu, \dots$) using the grammar $\tau ::= \delta \,|\, \nu \,|\, \tau \to \tau$, where $\to$ is associative to the right. *Variables* (typically $X, Y, Z, x, y, z, a, b, \dots$) and *constants* (typically $f, c, \dots$) have an assigned type.

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, and are typed as usual. For convenience, terms of the form $(\dots(a\, t_1)\dots t_n)$, where a is a constant or a variable, will be written as $a(t_1, \dots, t_n)$, and terms of the form $\lambda x_1. \cdots .\lambda x_n.t$ as $\lambda x_1, \dots, x_n.t$. We use $\vec{x}$ as a short-hand for $x_1, \dots, x_n$. If nothing is said, terms are assumed to be written in η-long β-normal form. Therefore, all terms have the form $\lambda x_1. \dots .\lambda x_n.h(t_1, \dots, t_m)$, where $m, n \geq 0$, h is either a constant or a variable, $t_1, \dots, t_m$ have also this form, and the term $h(t_1, \dots, t_m)$ has a basic type.

Other standard notions of the simply typed λ-calculus, like *bound* and *free* occurrences of variables, α*-conversion,* β*-reduction,* η*-long* β*-normal form*, etc. are defined as usual (see [Dowek 2001]). We will write free occurrences of variables with capital letters $X, Y, \dots$, for the sake of readability. The set of free variables of a term t is denoted by $\mathrm{Vars}(t)$. When we write an equality between two λ-terms, we mean that they are equivalent modulo α, β and η equivalence. When we write an equality $=_\alpha$, we mean that they are α-equivalent.

Substitutions are also finite sets of pairs $\sigma = [X_1 \mapsto t_1, \dots, X_n \mapsto t_n]$ where X_i and t_i have the same type and the X_i are pairwise distinct variables. They can be extended to type preserving function from terms to terms as usual. We say that a substitution σ_1 is *more general* than another substitution σ_2, if there exists a substitution σ' satisfying $\sigma' \circ \sigma_1(X) = \sigma_2(X)$, for all $X \in \mathrm{Dom}(\sigma_1) \cup \mathrm{Dom}(\sigma_2)$. We say that a variable X *occurs* in a substitution σ, if $X \in \mathrm{Vars}(\sigma(Y))$, for some $Y \in \mathrm{Dom}(\sigma)$.

A *higher-order unification problem* is a finite set of equations $P = \{t_1 \stackrel{?}{=} u_1, \dots, t_n \stackrel{?}{=} u_n\}$, where t_i and u_i have the same type. A *solution* or *unifier* of a unification problem P is a substitution σ satisfying $\sigma(t) = \sigma(u)$, for all equations $t \stackrel{?}{=} u \in P$. We say that a unifier σ is *most general* if, for any other unifier σ' more general than σ, we have σ is also more general than σ'.

A *higher-order pattern* is a λ-term where, when written in $\beta\eta$-normal form, all free variable occurrences are applied to lists of pairwise distinct bound variables. For instance, $\lambda x.f(X(x), Y)$, $f(c, \lambda x.x)$ and $\lambda x.\lambda y.X(\lambda z.x(z), y)$ are patterns, while $\lambda x.f(X(X(x)), Y)$, $f(X(c), c)$ and $\lambda x.\lambda y.X(x, x)$ are not. Notice that, since $\lambda z.x(z)$ is equivalent to x, the parameters of $X(\lambda z.x(z), y)$ are considered a list of pairwise distinct bound variables.

Higher-order pattern unification is the problem of deciding if there exists a unifier for a set of equations between higher-order patterns. Like in nominal unification, most general pattern unifiers are unique. Moreover, most general unifiers instantiate variables by higher-order patterns.

The following is a set of rules defining Nipkow's algorithm [Nipkow 1993] that computes, when it exists, the most general unifier of a pattern unification problem.

$$\begin{array}{rcl}
\lambda x.s \stackrel{?}{=} \lambda x.t & \to & \langle \{s \stackrel{?}{=} t\}, [\,]\rangle \\
a(t_1, \dots, t_n) \stackrel{?}{=} a(u_1, \dots, u_n) & \to & \langle \{t_1 \stackrel{?}{=} u_1, \dots, t_n \stackrel{?}{=} u_n\}, [\,]\rangle \\
 & & \text{where } a \text{ is a constant or bound variable} \\
Y(\vec{x}) \stackrel{?}{=} a(u_1, \dots, u_m) & \to & \langle\ \{Y_1(\vec{x}) \stackrel{?}{=} u_1, \dots, Y_m(\vec{x}) \stackrel{?}{=} u_m\}, \\
 & & \quad [Y \mapsto \lambda\vec{x}.a(Y_1(\vec{x}), \dots, Y_m(\vec{x}))]\ \rangle \\
 & & \text{where } Y \notin \mathrm{FV}(u_1, \dots, u_m) \\
 & & \text{and } a \text{ is a constant or } a \in \{\vec{x}\} \\
X(\vec{x}) \stackrel{?}{=} X(\vec{y}) & \to & \langle \emptyset, [X \mapsto \lambda\vec{x}.Z(\vec{z})]\rangle \\
 & & \text{where } \{\vec{z}\} = \{x_i \mid x_i = y_i\} \\
X(\vec{x}) \stackrel{?}{=} Y(\vec{y}) & \to & \langle \emptyset, [X \mapsto \lambda\vec{x}.Z(\vec{z}), Y \mapsto \lambda\vec{y}.Z(\vec{z})]\rangle \\
 & & \text{where } X \neq Y \text{ and } \{\vec{z}\} = \{\vec{x}\} \cap \{\vec{y}\}
\end{array}$$

The rules transform any equation into a pair $\langle$set of equations, substitution$\rangle$. The algorithm proceeds by replacing the equation on the left of the rule by the set of equations on the right. The substitution is applied to the new set of equations, and used to, step by step, construct the unifier. Therefore, any rule of the form $t \stackrel{?}{=} u \to \langle E, \rho\rangle$ produces a transformation of the form

$$\langle P \cup \{t \stackrel{?}{=} u\}, \sigma\rangle \Rightarrow \langle \rho(P) \cup E, \rho \circ \sigma\rangle$$

The algorithm starts with the pair $\langle P, Id\rangle$ and, if P is solvable, finishes with $\langle \emptyset, \sigma\rangle$, where σ with domain restricted to $\mathrm{FV}(P)$ is the most general unifier [Nipkow 1993, Theorem 3.1].

In the first rule the binder can be removed because, in Nipkow's presentation, free and bound variable names are assumed to be from distinct sets, and can be distinguished. The equations on the right of the second rule may not be normalized, i.e. the term $\lambda\vec{x}.Y_i(x_1,\dots,x_n)$ may require an η-expansion when u_i is not base typed.

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 [Urban et al. 2003; 2004] as a quiz.

Example 3.1. The nominal equation

$$\mathsf{a.b.f}(\mathsf{X}_1,\mathsf{b}) \overset{?}{\approx} \mathsf{b.a.f}(\mathsf{a},\mathsf{X}_1)$$

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

$$\mathsf{a.b.f}(\mathsf{X}_1,\mathsf{b}) \overset{?}{\approx} \mathsf{a.b.f}(\mathsf{b},\mathsf{X}_1)$$

which is solvable, and must be α-converted as

$$\mathsf{a.b.f}(\mathsf{X}_1,\mathsf{b}) \overset{?}{\approx} \mathsf{a.b.f}(\mathsf{b},(\mathsf{a}\,\mathsf{b})\cdot\mathsf{X}_1)$$

Recall that $(\mathsf{a}\,\mathsf{b})\cdot\mathsf{X}_1$ 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 described in the introduction, we have to replace every occurrence of X_1 by $X_1(a,b)$, since $\langle\mathsf{a},\mathsf{b}\rangle$ is the list of atoms (bound variables a,b) that can be captured. We get:

$$\lambda a.\lambda b.f(X_1(a,b),b) \overset{?}{=} \lambda b.\lambda a.f(a,X_1(a,b))$$

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

$$\lambda a.\lambda b.f(X_1(a,b),b) \overset{?}{=} \lambda a.\lambda b.f(b,X_1(b,a))$$

which is unsolvable, like the original nominal equation.

Example 3.2. The nominal equation

$$\mathsf{a.b.f}(\mathsf{X}_2,\mathsf{b}) \overset{?}{\approx} \mathsf{b.a.f}(\mathsf{a},\mathsf{X}_3)$$

is solvable. Its translation is

$$\lambda a.\lambda b.f(X_2(a,b),b) \overset{?}{=} \lambda b.\lambda a.f(a,X_3(a,b))$$

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

$$\begin{array}{l} X_2 \mapsto \lambda x.\lambda y.y \\ X_3 \mapsto \lambda x.\lambda y.x \end{array}$$

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

$$\begin{array}{l} \mathsf{X}_2 \mapsto \mathsf{b} \\ \mathsf{X}_3 \mapsto \mathsf{a} \end{array}$$

Example 3.3. In some cases, there are interrelationships between the instances of variables that make reconstruction of unifiers more difficult. This is shown with the following example:

$$\mathsf{a.b.f}(\mathsf{b},\mathsf{X}_4) \overset{?}{\approx} \mathsf{b.a.f}(\mathsf{a},\mathsf{X}_5)$$

that is solvable. Its translation gives:

$$\lambda a.\lambda b.f(b, X_4(a,b)) \stackrel{?}{=} \lambda b.\lambda a.f(a, X_5(a,b))$$

and its most general unifier is:[2]

$$X_4 \mapsto \lambda x.\lambda y.X_5(y,x)$$

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

$$\mathsf{X}_4 \mapsto (\mathsf{a}\ \mathsf{b})\cdot\mathsf{X}_5$$

The swapping $(\mathsf{a}\,\mathsf{b})$ comes from the fact that the arguments of X_5 and the lambda abstractions in front have a different order.

Example 3.4. The solution of a nominal unification problem is not just a substitution, but a pair $\langle\nabla,\sigma\rangle$ 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

$$\mathsf{a}.\mathsf{b}.\mathsf{f}(\mathsf{b}, \mathsf{X}_6) \stackrel{?}{\approx} \mathsf{a}.\mathsf{a}.\mathsf{f}(\mathsf{a}, \mathsf{X}_7)$$

has as solution

$$\begin{array}{l}\sigma = [\mathsf{X}_6 \mapsto (\mathsf{b}\ \mathsf{a})\cdot\mathsf{X}_7]\\ \nabla = \{\mathsf{b}\,\#\,\mathsf{X}_7\}\end{array}$$

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-order unification problem. The translation of the equation using the translation algorithm results on:

$$\lambda a.\lambda b.f(b, X_6(a,b)) \stackrel{?}{=} \lambda a.\lambda a.f(a, X_7(a,b)) \tag{1}$$

After an α-conversion we get

$$\lambda a.\lambda c.f(c, X_6(a,c)) \stackrel{?}{=} \lambda a.\lambda c.f(c, X_7(c,b))$$

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 nominal variables as higher-order variables. Occurrences of nominal variables become free occurrences of variables, and occurrences of atoms, if are bound, become bound occurrences of variables. Therefore, in most cases, after the translation the distinction atom/variable becomes a distinction free/bound variable. However, if atoms are not bound, 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:

$$\lambda a.\lambda b.\lambda a.\lambda b.f(b, X_6(a,b)) \stackrel{?}{=} \lambda a.\lambda b.\lambda a.\lambda a.f(a, X_7(a,b))$$

[2]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 $\mathsf{X}_5 \mapsto (\mathsf{a}\ \mathsf{b})\cdot\mathsf{X}_4$.

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:

$$\begin{array}{l} \mathsf{X}_6 \mapsto (\mathsf{a}\ \mathsf{b})\cdot\mathsf{X}_8 \\ \mathsf{X}_7 \mapsto \mathsf{X}_8 \end{array}$$

As far as $X_8(x)$ is translated back as X_8, and $X_8(x)$ does not use 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 $\{\mathsf{b}\,\#\,\mathsf{X}_8\}$. Then, $X_8(y)$ is translated back as $(\mathsf{a}\,\mathsf{b})\cdot\mathsf{X}_8$.

4. REMOVING FRESHNESS EQUATIONS

In this section we show that freshness equations do not contribute to make nominal unification more expressive. We prove that nominal unification can be linearly-reduced to nominal unification without freshness equations (Corollary 4.5). We call this restriction of nominal unification *equational nominal unification* (Definition 4.1). In the next sections we will describe a quadratic reduction of equational nominal unification to higher-order pattern unification. The absence of freshness equations makes the reduction to higher-order pattern unification simpler, compared with the reduction described in the preliminary version of this paper [Levy and Villaret 2008].

Definition 4.1. Equational Nominal Unification is the problem of deciding if, a given set of nominal equality equations $\{t_1 \overset{?}{\approx} u_1, \ldots, t_n \overset{?}{\approx} u_n\}$ has a solution.

Definition 4.2. We define the translation of nominal unification problems into equational nominal unification problems inductively as follows:

$$\begin{array}{l} Eq(\{\mathsf{a}\#^{?}\mathsf{t}\} \cup \mathsf{P}) = \{\mathsf{a.b.t} \overset{?}{\approx} \mathsf{b.b.t}\} \cup Eq(\mathsf{P}) \ \text{ for some } \mathsf{b} \neq \mathsf{a} \\ Eq(\{\mathsf{t} \overset{?}{\approx} \mathsf{u}\} \cup \mathsf{P}) = \{\mathsf{t} \overset{?}{\approx} \mathsf{u}\} \cup Eq(\mathsf{P}) \end{array}$$

LEMMA 4.3. *Given a nominal unification problem* P*, its translation into equational nominal unification* $Eq(\mathsf{P})$ *can be calculated in linear time. Hence,* $Eq(\mathsf{P})$ *has linear-size on the size of* P.

LEMMA 4.4. *The pair* $\langle\nabla,\sigma\rangle$ *solves* P*, if, and only if,* $\langle\nabla,\sigma\rangle$ *solves* $Eq(\mathsf{P})$.

PROOF. We first prove that $\langle \mathsf{a}\#\mathsf{t}, \mathsf{Id}\rangle$ is a solution of $\{\mathsf{a.b.t} \overset{?}{\approx} \mathsf{b.b.t}\}$ when $\mathsf{b} \neq \mathsf{a}$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \mathsf{t}\approx\mathsf{t}\end{array}\quad\begin{array}{c}\mathsf{a}\#\mathsf{t}\\ \vdots\ \text{(lemma 2.7)}\\ \mathsf{b}\#(\mathsf{a}\,\mathsf{b})\cdot\mathsf{t}\end{array}}{\mathsf{b.t}\approx\mathsf{a}.(\mathsf{a}\,\mathsf{b})\cdot\mathsf{t}}\ (\approx\text{-abst-2})\qquad \dfrac{\mathsf{a}\#\mathsf{t}}{\mathsf{a}\#\mathsf{b.t}}\ (\#\text{-abst-2})}{\mathsf{a.b.t}\approx\mathsf{b.b.t}}\ (\approx\text{-abst-2})$$

In this proof we prove $\mathsf{t}\approx\mathsf{t}$ from an empty set of assumptions. We can prove that this is always possible, for any term t, by structural induction on t. We also prove $\mathsf{b}\#(\mathsf{a}\,\mathsf{b})\cdot\mathsf{t}$ from $\mathsf{a}\#\mathsf{t}$, using Lemma 2.7 of [Urban et al. 2004].

Lemma 2.14 of [Urban et al. 2004] states that $\nabla' \vdash \sigma(\nabla)$ and $\nabla \vdash \mathsf{t} \approx \mathsf{t}'$ implies $\nabla' \vdash \sigma(\mathsf{t}) \approx \sigma(\mathsf{t}')$. In particular, $\nabla \vdash \sigma(\mathsf{a}\#\mathsf{t})$ and $\mathsf{a}\#\mathsf{t} \vdash \mathsf{a.b.t} \approx \mathsf{b.b.t}$ implies $\nabla \vdash \sigma(\mathsf{a.b.t}) \approx \sigma(\mathsf{b.b.t})$. Therefore, if $\langle\nabla,\sigma\rangle$ solves $\mathsf{a}\#^{?}\mathsf{t}$, then $\langle\nabla,\sigma\rangle$ solves $\mathsf{a.b.t} \overset{?}{\approx} \mathsf{b.b.t}$.

Second, analyzing the previous proof, we see that the inference rules applied in each situation were the only applicable rules. Therefore, any solution $\langle\nabla,\sigma\rangle$ solving $\mathsf{a.b.t} \overset{?}{\approx} \mathsf{b.b.t}$, also solves $\mathsf{a}\#^?\mathsf{t}$, because any proof of $\sigma(\mathsf{a.b.t}) \approx \sigma(\mathsf{b.b.t})$ contains a proof of $\mathsf{a}\#\sigma(\mathsf{t})$ as a sub-proof.

From, these two facts we conclude that $\mathsf{a}\#^?\mathsf{t}$ and $\mathsf{a.b.t} \overset{?}{\approx} \mathsf{b.b.t}$ have the same set of solutions, for any $\mathsf{b} \neq \mathsf{a}$. Therefore, $\{\mathsf{a}\#^?\mathsf{t}\} \cup \mathsf{P}$ and $\{\mathsf{a.b.t} \overset{?}{\approx} \mathsf{b.b.t}\} \cup \mathsf{P}$, also have the same set of solutions, for any nominal unification problem P. From this we conclude that P and $Eq(\mathsf{P})$ have the same set of solutions. □

COROLLARY 4.5. *Nominal Unification can be linearly-reduced to Equational Nominal Unification.*

5. THE TRANSLATION ALGORITHM

In this section we formalize the translation algorithm for types, terms, problems and solutions (Definitions 5.1, 5.2, 5.5, and 5.7, respectively). We prove that, a pair $\langle\nabla,\sigma\rangle$ solves a problem P if, and only if, the translation of $\langle\nabla,\sigma\rangle$ solves the translation of P (Theorem 5.12). This allows us to conclude that, if an equational nominal unification problem is solvable, then its translation is also solvable (Theorem 5.14), but not the reverse implication. This reverse implication will be proved in Section 7 (Theorem 7.7).

We transform equational 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* 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 5.1. The translation function is defined on sorts inductively as follows.

$$\begin{array}{l}
[\![\delta]\!] = \delta \\
[\![\nu]\!] = \nu \\
[\![\tau_1 \times \cdots \times \tau_n \to \tau]\!] = [\![\tau_1]\!] \to \cdots \to [\![\tau_n]\!] \to [\![\tau]\!] \\
[\![\langle\nu\rangle\tau]\!] = \nu \to [\![\tau]\!]
\end{array}$$

where δ and ν are base types.

The translation function for terms depends on a list of atoms L and a freshness environment ∇. Later, in Theorems 5.14, 7.7 and 8.7 we particularize this list as an enumeration of all the atoms of a nominal unification problem without repetitions. Therefore, this list depends on the unification problem, and its length is bounded on the size of the problem. This fact allows us to ensure that the translation of a problem $[\![\mathsf{P}]\!]_{\mathsf{L}}$, which is bounded by $|\mathsf{P}|\cdot|\mathsf{L}|$ in Lemma 5.6, is in fact quadratic. Urban's nominal unification algorithm [Urban et al. 2003; 2004] allows us to ensure that the solution of a nominal unification problem can also be expressed only using atoms occurring in the problem, i.e. using atoms from L. This property is essential to prove Theorem 5.14. The freshness environment indicates which atoms are not *capturable* by a variable. As defined below in Definition 5.2, we translate every nominal variable X, as the application of a free variable X to a list of the atoms that it can capture. Therefore, this list is constructed as the sublist of atoms in L that are not in ∇. This is the only case in Definition 5.2 where the translation function uses the parameters L and ∇.

For every function symbol f, we will use a constant with the same name f. Every atom a is translated as a (bound) variable, with the same name a. For every variable (unknown) X, we will use a (free) variable with the same name X. Trivially, atom abstractions $\mathsf{a.t}$ are translated as lambda abstractions $\lambda a.t$, and applications $\mathsf{f}(\mathsf{t}_1,\dots,\mathsf{t}_n)$

as applications $f\,t_1 \cdots t_n$. As we say in Section 2, for clarity, we will write these applications in uncurried form as $f(t_1, \ldots, t_n)$. Similarly, we will also uncurry applications of higher-order variables. The translation of suspensions $\pi \cdot \mathsf{X}$ is more complicated, as far as it gets rid of atom capture. Recall that in all cases we use distinct character fonts for symbols of nominal terms and symbols of lambda-terms. The translation is parametric on a freshness environment. Notice that, although we have removed freshness equations, nominal unifiers are composed by a freshness environment and a substitution.

Definition 5.2. The translation function from nominal terms into λ-terms is parametric on a freshness environments ∇ and a list L of atoms, and is defined inductively as follows.

$$\begin{array}{ll}
[\![\mathsf{a}]\!]_{\mathsf{L},\nabla} = a & \\
[\![\mathsf{f}(\mathsf{t}_1,\ldots,\mathsf{t}_n)]\!]_{\mathsf{L},\nabla} = f([\![\mathsf{t}_1]\!]_{\mathsf{L},\nabla},\ldots,[\![\mathsf{t}_n]\!]_{\mathsf{L},\nabla}) & \\
[\![\mathsf{a}.\mathsf{t}]\!]_{\mathsf{L},\nabla} = \lambda a.[\![\mathsf{t}]\!]_{\mathsf{L},\nabla} & \\
[\![\pi\cdot\mathsf{X}]\!]_{\mathsf{L},\nabla} = X([\![\pi\cdot\mathsf{a}_1]\!]_{\mathsf{L},\nabla},\ldots,[\![\pi\cdot\mathsf{a}_n]\!]_{\mathsf{L},\nabla}) & \text{being } \langle\mathsf{a}_1,\ldots,\mathsf{a}_n\rangle \text{ the sublist}^{3}\text{of} \\
& \text{atoms of } \mathsf{L} \text{ allowed in } X \text{ by } \nabla, \text{ i.e.} \\
& \langle\mathsf{a}_1,\ldots,\mathsf{a}_n\rangle = \langle \mathsf{a}\in\mathsf{L} \mid \mathsf{a}\,\#\,\mathsf{X}\notin\nabla\rangle
\end{array}$$

where, for any atom $\mathsf{a} : \nu$, $a : [\![\nu]\!]$ is the corresponding bound variable, for any function symbol $\mathsf{f} : \tau$, $f : [\![\tau]\!]$ is the corresponding constant, and for any variable $\mathsf{X} : \tau$, X is a variable of type $X : [\![\nu_1]\!] \to \ldots \to [\![\nu_n]\!] \to [\![\tau]\!]$ where $\mathsf{a_i} : \nu_\mathsf{i}$.[4]

LEMMA 5.3. *For every nominal term* t *of sort* τ*, list of atoms* L*, and freshness environment* ∇*,* $[\![\mathsf{t}]\!]_{\mathsf{L},\nabla}$ *is a* λ*-term with type* $[\![\tau]\!]$.

PROOF. The proof is simple by structural induction on t. The only point that needs a more detailed explanation is the case of suspensions. Since $\mathsf{a_i} : \nu_\mathsf{i}$, $\mathsf{X} : \tau$, for $i = 1, \ldots, n$, and $X : [\![\nu_1]\!] \to \cdots \to [\![\nu_n]\!] \to [\![\tau]\!]$, we have $[\![\mathsf{X}]\!]_{\mathsf{L},\nabla} = X\left([\![\mathsf{a}_1]\!]_{\mathsf{L},\nabla}, \ldots, [\![\mathsf{a}_n]\!]_{\mathsf{L},\nabla}\right) : [\![\tau]\!]$. When X is affected by a permutation π we also have $[\![\pi\cdot\mathsf{X}]\!]_{\mathsf{L},\nabla} = X\left([\![\pi\cdot\mathsf{a}_1]\!]_{\mathsf{L},\nabla}, \ldots, [\![\pi\cdot\mathsf{a}_n]\!]_{\mathsf{L},\nabla}\right) : [\![\tau]\!]$ because the suspension $\pi\cdot\mathsf{X}$ is not a valid nominal term unless $\mathsf{a_i}$ and $\pi\cdot\mathsf{a_i}$ belong to the same sort. □

Example 5.4. Given the nominal term $\mathsf{t} = \mathsf{a.b.c.(c\,a)(a\,b)\cdot X}$, after applying the substitution $\sigma = [\mathsf{X} \mapsto \mathsf{f(a,b,c,Y)}]$ we get $\sigma(\mathsf{t}) = \mathsf{a.b.c.f(b,c,a,Y)}$. The translation of the term t w.r.t. $\mathsf{L} = \langle \mathsf{a,b,c}\rangle$ and $\nabla_1 = \emptyset$ results into $[\![\mathsf{t}]\!]_{\mathsf{L},\nabla_1} = \lambda a.\lambda b.\lambda c.X(b,c,a)$ and, the translation of the instantiation $\sigma(\mathsf{t})$ w.r.t. L and $\nabla_2 = \{\mathsf{a\#Y}\}$ results into $[\![\sigma(\mathsf{t})]\!]_{\mathsf{L},\nabla_2} = \lambda a.\lambda b.\lambda c.f(b,c,a,Y(c,a))$. There is a λ-substitution $[X \mapsto \lambda a.\lambda b.\lambda c.f(a,b,c,Y(b,c))]$ (described in Definition 5.7) that when applied to $[\![\mathsf{t}]\!]_{\mathsf{L},\nabla_1}$ results into $[\![\sigma(\mathsf{t})]\!]_{\mathsf{L},\nabla_2}$. Graphically this can be represented as the commutation of the following diagram (proved in Lemma 5.10).

[3]Notice that we say sublist, not subset, to emphasize that the relative order between $\mathsf{a_i}$ is preserved.
[4]Notice that $\mathsf{a_i}$ and $\pi\cdot\mathsf{a_i}$ are of the same sort, and that the type of X depends on the type of X, and the parameters ∇ and L.

$$\begin{tikzcd}
\mathsf{a.b.c.(c\,a)(a\,b)\cdot X} \arrow[r, "{[\mathsf{X} \mapsto \mathsf{f(a,b,c,Y)}]}"] \arrow[d, "{[\![\ ]\!]_{\mathsf{L},\emptyset}}"] & \mathsf{a.b.c.f(b,c,a,(c\,a)(a\,b)\cdot Y)} \arrow[d, "{[\![\ ]\!]_{\mathsf{L},\{\mathsf{a}\#\mathsf{Y}\}}}"] \\
\lambda a.\lambda b.\lambda c.X(b,c,a) \arrow[r, "{[X \mapsto \lambda a.\lambda b.\lambda c.f(a,b,c,Y(b,c))]}"] & \lambda a.\lambda b.\lambda c.f(b,c,a,Y(c,a))
\end{tikzcd}$$

Definition 5.5. Given a list of atoms $\mathsf{L} = \langle \mathsf{a}_1, \dots, \mathsf{a}_n \rangle$, the translation function is defined on equational nominal problems as follows

$$[\![\mathsf{P}]\!]_{\mathsf{L}} = \{\lambda a_1. \dots .\lambda a_n.[\![\mathsf{t}]\!]_{\mathsf{L},\emptyset} \stackrel{?}{=} \lambda a_1. \dots .\lambda a_n.[\![\mathsf{u}]\!]_{\mathsf{L},\emptyset} \mid t \stackrel{?}{\approx} u \in P\}$$

LEMMA 5.6. *Given an equational nominal unification problem* P*, let* L *be a list containing all atoms of* P *without repetitions. The translation* $[\![\mathsf{P}]\!]_{\mathsf{L}}$ *is a higher-order pattern unification problem.*
Moreover, the size and the time needed to compute $[\![\mathsf{P}]\!]_{\mathsf{L}}$ *is bounded by* $|\mathsf{P}| \cdot |\mathsf{L}|$.

PROOF. By Lemma 5.3, $\lambda a_1. \dots .\lambda a_n.[\![\mathsf{t}]\!]_{\mathsf{L},\emptyset} \stackrel{?}{=} \lambda a_1. \dots .\lambda a_n.[\![\mathsf{u}]\!]_{\mathsf{L},\emptyset}$ is an equation between λ-terms of the same type. Now notice that $[\![\pi \cdot \mathsf{X}]\!]_{\mathsf{L},\nabla} = X\left([\![\pi \cdot \mathsf{b}_1]\!]_{\mathsf{L},\nabla}, \dots, [\![\pi \cdot \mathsf{b}_m]\!]_{\mathsf{L},\nabla}\right)$ translate the variable X into an application of the free variable X to a list of pairwise distinct bound variables, because the list L contains pairwise distinct atoms, hence the b_i are all different, π is a permutation, and we ensure that all atoms are translated into bound variables by adding λ-bindings in front of both terms. Therefore, both sides of the equation are higher-order patterns.
Concerning the size of the translation, we obtain this bound due to the translation of these suspensions. □

Finally, we have to translate solutions of nominal unification problems into λ-substitutions.

Definition 5.7. Given a freshness environment ∇ and a list $\mathsf{L} = \langle \mathsf{a}_1, \dots, \mathsf{a}_n \rangle$ of atoms, we define the translation function on nominal substitutions σ as follows

$$[\![\sigma]\!]_{\mathsf{L},\nabla} = \bigcup_{\mathsf{X} \in \mathrm{Dom}(\sigma)} \left[X \mapsto \lambda a_1. \cdots \lambda a_n.[\![\sigma(\mathsf{X})]\!]_{\mathsf{L},\nabla}\right]$$

The following remark shows why in some places we require that solutions $\langle \nabla, \sigma \rangle$ of a nominal problem P satisfy $\mathrm{Dom}(\sigma) = \mathrm{Vars}(\mathsf{P})$.

Remark 5.8. Consider the nominal unification problem $\mathsf{P}_1 = \{\mathsf{a.X} \stackrel{?}{\approx} \mathsf{b.Y}\}$, and the list $\mathsf{L} = \langle \mathsf{a}, \mathsf{b} \rangle$ of its atoms. Its translations as a higher-order pattern unification problem is

$$[\![\mathsf{P}_1]\!]_{\mathsf{L}} = \left[\!\!\left[\{\mathsf{a.X} \stackrel{?}{\approx} \mathsf{b.Y}\}\right]\!\!\right]_{\mathsf{L}} = \{\lambda a.\lambda b.\lambda a.X(a,b) \stackrel{?}{=} \lambda a.\lambda b.\lambda b.Y(a,b)\}$$

The λ-substitution

$$\sigma_1 = [\![[\mathsf{X} \mapsto (\mathsf{a\,b})\cdot \mathsf{Y}]]\!]_{\mathsf{L},\{\mathsf{a}\#\mathsf{Y}\}} = [X \mapsto \lambda a.\lambda b.Y(a)]$$

does not solve $[\![\mathsf{P}_1]\!]_{\mathsf{L}}$. Whereas the λ-substitution

$$\sigma_2 = [\![[\mathsf{X} \mapsto (\mathsf{a\,b})\cdot \mathsf{Y}, \mathsf{Y} \mapsto \mathsf{Y}]]\!]_{\mathsf{L},\{\mathsf{a}\#\mathsf{Y}\}} = [X \mapsto \lambda a.\lambda b.Y(a),\ Y \mapsto \lambda a.\lambda b.Y(b)]$$

solves $[\![\mathsf{P}_1]\!]_{\mathsf{L}}$. Notice that in the first case the domain of the nominal unifier (as defined in Section 2) is $\{\mathsf{X}\}$, whereas in the second case it is $\{\mathsf{X}, \mathsf{Y}\} = \mathrm{Vars}(\mathsf{P}_1)$.

We will see in Theorem 5.12 that, if L contains all atoms of P and σ, $\mathrm{Vars}(\mathsf{P}) \subseteq \mathrm{Dom}(\sigma)$ and $\langle\nabla, \sigma\rangle$ solves P, then $[\![\sigma]\!]_{\mathsf{L},\nabla}$ solves $[\![\mathsf{P}]\!]_{\mathsf{L}}$. With this example we see that the second condition in the implication is necessary.

Now, consider the nominal unification problem $\mathsf{P}_2 = \{\mathsf{a.b.(a\,b)\cdot X} \overset{?}{\approx} \mathsf{b.b.(a\,b)\cdot X}\}$ and the same list $\mathsf{L} = \langle \mathsf{a}, \mathsf{b}\rangle$. Its translation is

$$[\![\mathsf{P}_2]\!]_{\mathsf{L}} = \{\lambda a.\lambda b.\lambda a.\lambda b.X(b,a) = \lambda a.\lambda b.\lambda b.\lambda b.X(b,a)\}$$

In this case, the pattern substitution σ_1 is a most general pattern unifier of $[\![\mathsf{P}_2]\!]_{\mathsf{L}}$, and σ_2 is a pattern unifier, but not a most general one.

As we will see, we have to require $\mathrm{Vars}(\mathsf{P}) \supseteq \mathrm{Dom}(\sigma)$, if we want to ensure that the translation not only preserves unifiability, but also most generality.

Notice that w.l.o.g. we can require most general nominal solutions to satisfy $\mathrm{Vars}(\mathsf{P}) = \mathrm{Dom}(\sigma)$, because most general solutions do not instantiate variables not belonging to $\mathrm{Vars}(\mathsf{P})$, and we can always add pairs $\mathsf{X} \mapsto \mathsf{X}$ for all variables occurring in P and not in $\mathrm{Dom}(\sigma)$.

Notice also that in σ_2 there are two free variables with the same name Y, but distinct types. Be aware that in $Y \mapsto \lambda a.\lambda b.Y(b)$ the *replaced* Y has two arguments, whereas the *introduced* Y has only one argument (they have distinct types). In λ-calculus this is not a problem. The reason of this duplicity is that the translation function is parametric on a freshness environment ∇. This is relevant in the case of a nominal variable. For instance, $[\![\mathsf{Y}]\!]_{\langle \mathsf{a},\mathsf{b}\rangle,\emptyset} = Y(a,b)$ where we use the replaced Y with two parameters, and $[\![\mathsf{Y}]\!]_{\langle \mathsf{a},\mathsf{b}\rangle,\{\mathsf{a\#Y}\}} = Y(b)$ where we use the introduced Y with one parameter. If we would like to avoid this duplicity we have to forbid the use of a variable of the problem in the right-hand side of a nominal solution. Then, in our example P_1, the most general nominal solution could be written as $\langle\{\mathsf{a\#Y'}\}, [\mathsf{X} \mapsto (\mathsf{a\,b})\cdot\mathsf{Y'}, \mathsf{Y} \mapsto \mathsf{Y'}]\rangle$.

To prove that the translation of the solution of a problem is a solution of the translation of the problem, we start by proving the following two technical lemmas.

LEMMA 5.9. *For any list of atoms* L*, freshness environment* ∇*, nominal terms* t*,* u*, and atom* a*, if all atoms of* t *and* u *belong to* L*, then we have*

(*1*) $\nabla \vdash \mathsf{a} \# \mathsf{t}$ *if, and only if,* $a \notin \mathrm{FV}([\![\mathsf{t}]\!]_{\mathsf{L},\nabla})$*, and*
(*2*) $\nabla \vdash \mathsf{t} \approx \mathsf{u}$ *if, and only if,* $[\![\mathsf{t}]\!]_{\mathsf{L},\nabla} =_\alpha [\![\mathsf{u}]\!]_{\mathsf{L},\nabla}$.

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 $\mathsf{t} = \pi\cdot\mathsf{X}$, in this case, $[\![\pi\cdot\mathsf{X}]\!]_{\mathsf{L},\nabla} = X\left([\![\pi\cdot\mathsf{b}_1]\!]_{\mathsf{L},\nabla}, \dots, [\![\pi\cdot\mathsf{b}_\mathsf{m}]\!]_{\mathsf{L},\nabla}\right)$, where $\langle \mathsf{b}_1, \dots, \mathsf{b}_\mathsf{m}\rangle$ is the sublist of atoms of L satisfying $\mathsf{b}_\mathsf{i}\#\mathsf{X} \notin \nabla$. Therefore, since all atoms permuted by π are in L, we can establish the following sequence of equivalences $\nabla \vdash \mathsf{a}\#\pi\cdot\mathsf{X}$ iff $\pi^{-1}\cdot\mathsf{a}\#\mathsf{X} \in \nabla$ iff $\pi^{-1}\cdot\mathsf{a} \notin \{\mathsf{b}_1, \dots, \mathsf{b}_\mathsf{m}\}$ iff $\mathsf{a} \notin \{\pi\cdot\mathsf{b}_1, \dots, \pi\cdot\mathsf{b}_\mathsf{m}\}$ iff $a \notin \mathrm{FV}(X([\![\pi\cdot\mathsf{b}_1]\!]_{\mathsf{L},\nabla}, \dots, [\![\pi\cdot\mathsf{b}_\mathsf{m}]\!]_{\mathsf{L},\nabla}))$ iff $a \notin \mathrm{FV}([\![\pi\cdot\mathsf{X}]\!])$.

The proof of the second statement can be done by induction on the equivalence $\mathsf{t} \approx \mathsf{u}$. We only comment the equivalence between suspensions: $\pi\cdot\mathsf{X} \approx \pi'\cdot\mathsf{X}$. Notice that, $\pi\cdot\mathsf{X} \approx \pi'\cdot\mathsf{X}$ if, and only if, for all atoms a such that $\pi\cdot\mathsf{a} \neq \pi'\cdot\mathsf{a}$, we have $\mathsf{a\#X} \in \nabla$. Since all atoms permuted by π are in L, this condition is equivalent to: the bound variables $[\![\pi\cdot\mathsf{a}]\!]_{\mathsf{L},\nabla}$ and $[\![\pi'\cdot\mathsf{a}]\!]_{\mathsf{L},\nabla}$ are passed as a parameter to X in $[\![\pi\cdot\mathsf{X}]\!]_{\mathsf{L},\nabla}$ and $[\![\pi'\cdot\mathsf{X}]\!]_{\mathsf{L},\nabla}$ only when $\pi\cdot\mathsf{a} = \pi'\cdot\mathsf{a}$. Finally, this condition is equivalent to $[\![\pi\cdot\mathsf{X}]\!]_{\mathsf{L},\nabla} = [\![\pi'\cdot\mathsf{X}]\!]_{\mathsf{L},\nabla}$. □

The first statement of the previous lemma will not be necessary for our purposes because we have removed freshness equations.

LEMMA 5.10. *For any list of atoms* $\mathsf{L} = \langle \mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n} \rangle$*, freshness environment* ∇*, nominal substitution* σ*, and nominal term* t *satisfying that all atoms of* t *are in* L *and* $\mathrm{Vars}(\mathsf{t}) \subseteq \mathrm{Dom}(\sigma)$*, we have* $[\![\sigma]\!]_{\mathsf{L},\nabla}([\![\mathsf{t}]\!]_{\mathsf{L},\emptyset}) = [\![\sigma(\mathsf{t})]\!]_{\mathsf{L},\nabla}$.

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

$$\begin{aligned}
&[\![\sigma]\!]_{\mathsf{L},\nabla}([\![\pi\cdot\mathsf{X}]\!]_{\mathsf{L},\emptyset})\\
&\quad= \Big[\dots, X \mapsto \lambda a_1 \dots \lambda a_n.[\![\sigma(\mathsf{X})]\!]_{\mathsf{L},\nabla}, \dots\Big] \Big(X([\![\pi\cdot\mathsf{a}_1]\!]_{\mathsf{L},\nabla}, \dots, [\![\pi\cdot\mathsf{a}_\mathsf{n}]\!]_{\mathsf{L},\nabla})\Big)\\
&\quad= \Big(\lambda a_1 \dots \lambda a_n.[\![\sigma(\mathsf{X})]\!]_{\mathsf{L},\nabla}\Big) \Big([\![\pi\cdot\mathsf{a}_1]\!]_{\mathsf{L},\nabla}, \dots, [\![\pi\cdot\mathsf{a}_\mathsf{n}]\!]_{\mathsf{L},\nabla}\Big)\\
&\quad= \Big[a_1 \mapsto [\![\pi\cdot\mathsf{a}_1]\!]_{\mathsf{L},\nabla}, \dots, a_n \mapsto [\![\pi\cdot\mathsf{a}_\mathsf{n}]\!]_{\mathsf{L},\nabla}\Big] \Big([\![\sigma(\mathsf{X})]\!]_{\mathsf{L},\nabla}\Big)\\
&\quad= [\![\pi\cdot\sigma(\mathsf{X})]\!]_{\mathsf{L},\nabla}\\
&\quad= [\![\sigma(\pi\cdot\mathsf{X})]\!]_{\mathsf{L},\nabla}
\end{aligned}$$

Notice that in the first equality we use $\mathsf{X} \in \mathrm{Vars}(\mathsf{t}) \subseteq \mathrm{Dom}(\sigma)$, hence $X \in \mathrm{Dom}([\![\sigma]\!]_{\mathsf{L},\nabla})$. Notice also that in the fourth equality we use that all atoms of $\mathsf{t} = \pi\cdot\mathsf{X}$ are in L, therefore π only permutes atoms of L. □

In the proof of this lemma we do not require L to contain all atoms of σ.

Example 5.11. Let be $\mathsf{L} = \langle \mathsf{a}, \mathsf{b} \rangle$, $\nabla = \{\mathsf{b}\#\mathsf{Y}\}$, $\mathsf{t} = \mathsf{f}((\mathsf{a}\,\mathsf{b})\cdot\mathsf{X}, (\mathsf{a}\,\mathsf{b})\cdot\mathsf{Y})$ and $\sigma = [\mathsf{X} \mapsto \mathsf{b}.\mathsf{a}, \mathsf{Y} \mapsto \mathsf{Y}]$. We have

$$\begin{aligned}
[\![\sigma]\!]_{\mathsf{L},\nabla} &= [\![[\mathsf{X} \mapsto \mathsf{b}.\mathsf{a}, \mathsf{Y} \mapsto \mathsf{Y}]]\!]_{\mathsf{L},\{\mathsf{b}\#\mathsf{Y}\}}\\
&= [X \mapsto \lambda a.\lambda b.[\![\mathsf{b}.\mathsf{a}]\!]_{\mathsf{L},\{\mathsf{b}\#\mathsf{Y}\}},\ Y \mapsto \lambda a.\lambda b.[\![\mathsf{Y}]\!]_{\mathsf{L},\{\mathsf{b}\#\mathsf{Y}\}}]\\
&= [X \mapsto \lambda a.\lambda b.\lambda b.a,\ Y \mapsto \lambda a.\lambda b.Y(a)]\\
[\![\mathsf{t}]\!]_{\mathsf{L},\emptyset} &= \Big[\!\Big[\mathsf{f}\big((\mathsf{a}\,\mathsf{b})\cdot\mathsf{X}, (\mathsf{a}\,\mathsf{b})\cdot\mathsf{Y}\big)\Big]\!\Big]_{\mathsf{L},\emptyset}\\
&= f\big(X(b,a), Y(b,a)\big)\\
[\![\sigma(\mathsf{t})]\!]_{\mathsf{L},\nabla} &= \Big[\!\Big[[\mathsf{X} \mapsto \mathsf{b}.\mathsf{a}, \mathsf{Y} \mapsto \mathsf{Y}]\,\mathsf{f}\big((\mathsf{a}\,\mathsf{b})\cdot\mathsf{X}, (\mathsf{a}\,\mathsf{b})\cdot\mathsf{Y}\big)\Big]\!\Big]_{\mathsf{L},\{\mathsf{b}\#\mathsf{Y}\}}\\
&= [\![\mathsf{f}(\mathsf{a}.\mathsf{b}, (\mathsf{a}\,\mathsf{b})\cdot\mathsf{Y})]\!]_{\mathsf{L},\{\mathsf{b}\#\mathsf{Y}\}}\\
&= f(\lambda a.b, Y(b))
\end{aligned}$$

Now, we have

$$\begin{aligned}
[\![\sigma]\!]_{\mathsf{L},\nabla}\Big([\![\mathsf{t}]\!]_{\mathsf{L},\emptyset}\Big) &= f\big(\underline{(\lambda a.\lambda b.\lambda b.a)(b,a)}, (\lambda a.\lambda b.Y(a))(b,a)\big) = f(\lambda c.b, Y(b))\\
&= f(\lambda a.b, Y(b)) = [\![\sigma(\mathsf{t})]\!]_{\mathsf{L},\nabla}
\end{aligned}$$

Notice that the substitution resulting form the β-reduction of the underlined redex needs to avoid the capture of b. This is done replacing the bound variable b by c. In the following section we will see that, in pattern unification, we can do this without using new bound variable names. In this case, we could have used a instead of c.

From these two lemmas we can prove the following results.

THEOREM 5.12. *For any list of atoms* $\mathsf{L} = \langle \mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n} \rangle$*, freshness environment* ∇*, equational nominal unification problem* P*, and nominal substitution* σ *if* L *contains all atoms of* P *without repetitions, and* σ *and* $\mathrm{Vars}(\mathsf{P}) \subseteq \mathrm{Dom}(\sigma)$*, we have that* $\langle \nabla, \sigma \rangle$ *solves the equational nominal unification problem* P*, if, and only if,* $[\![\sigma]\!]_{\mathsf{L},\nabla}$ *solves the pattern unification problem* $[\![\mathsf{P}]\!]_{\mathsf{L}}$.

PROOF. By definition of nominal solution, the pair $\langle\nabla,\sigma\rangle$ solves P iff

$$\nabla\vdash\sigma(\mathsf{t})\approx\sigma(\mathsf{u})\qquad\text{for all } \mathsf{t}\overset{?}{\approx}\mathsf{u}\in\mathsf{P}$$

By Lemma 5.9 this is equivalent to:

$$[\![\sigma(\mathsf{t})]\!]_{\mathsf{L},\nabla}=_\alpha[\![\sigma(\mathsf{u})]\!]_{\mathsf{L},\nabla}\qquad\text{for all } \mathsf{t}\overset{?}{\approx}\mathsf{u}\in\mathsf{P}$$

and, by Lemma 5.10 this is equivalent to:

$$[\![\sigma]\!]_{\mathsf{L},\nabla}([\![\mathsf{t}]\!]_{\mathsf{L},\emptyset})=[\![\sigma]\!]_{\mathsf{L},\nabla}([\![\mathsf{u}]\!]_{\mathsf{L},\emptyset})\qquad\text{for all } \mathsf{t}\overset{?}{\approx}\mathsf{u}\in\mathsf{P}$$

Since the substitution $[\![\sigma]\!]_{\mathsf{L},\nabla}$ does not instantiate the variables $a_1,\dots,a_n$, this is equivalent to (see Remark 5.13):

$$[\![\sigma]\!]_{\mathsf{L},\nabla}\Big(\lambda a_1.\dots.\lambda a_n.[\![\mathsf{t}]\!]_{\mathsf{L},\emptyset}\Big)=[\![\sigma]\!]_{\mathsf{L},\nabla}\Big(\lambda a_1.\dots.\lambda a_n.[\![\mathsf{u}]\!]_{\mathsf{L},\emptyset}\Big)\qquad\text{for all } \mathsf{t}\overset{?}{\approx}\mathsf{u}\in\mathsf{P}$$

where $\langle\mathsf{a}_1,\dots,\mathsf{a}_\mathsf{n}\rangle$ is the list of atoms occurring in P.

Finally, since $\Big[\!\!\Big[\mathsf{t}\overset{?}{\approx}\mathsf{u}\Big]\!\!\Big]_\mathsf{L}=\lambda a_1.\dots.\lambda a_n.[\![\mathsf{t}]\!]_{\mathsf{L},\emptyset}\overset{?}{=}\lambda a_1.\dots.\lambda a_n.[\![\mathsf{u}]\!]_{\mathsf{L},\emptyset}$, this is equivalent to $[\![\sigma]\!]_{\mathsf{L},\nabla}$ solves $[\![\mathsf{P}]\!]_\mathsf{L}$. □

A variant of the proof of Theorem 5.12 would allow us to prove that $\langle\nabla,\sigma\rangle$ solves $\mathsf{t}\overset{?}{\approx}\mathsf{u}$, if, and only if, $[\![\sigma]\!]_{\mathsf{L},\nabla}$ solves $[\![\mathsf{t}]\!]_{\mathsf{L},\emptyset}\overset{?}{=}[\![\mathsf{u}]\!]_{\mathsf{L},\emptyset}$. Therefore, it seems unnecessary to add the λ-bindings $\lambda a_1.\cdots.\lambda a_n$ in front of both sides of the higher-order equations, as was suggested in Example 3.4. The following remark illustrates what would happen if we had defined translation of equations in this way.

Remark 5.13. Assume that we had defined $\Big[\!\!\Big[\mathsf{t}\overset{?}{\approx}\mathsf{u}\Big]\!\!\Big]_\mathsf{L}=[\![\mathsf{t}]\!]_{\mathsf{L},\emptyset}\overset{?}{=}[\![\mathsf{u}]\!]_{\mathsf{L},\emptyset}$, instead of the definition we have for $\Big[\!\!\Big[\mathsf{t}\overset{?}{\approx}\mathsf{u}\Big]\!\!\Big]_\mathsf{L}$ with the external lambda.

The translation of the unsolvable nominal equation $\mathsf{a}\overset{?}{\approx}\mathsf{b}$ would result into $a\overset{?}{=}b$ which is solvable by $[a\mapsto b]$ (notice that, in this case, atoms are translated into free variables). The example does not contradict the proposed variant of Theorem 5.12 because the substitution $[a\mapsto b]$ is not the translation of any nominal substitution, i.e. there does not exists any freshness environment ∇ and any nominal substitution σ such that $[\![\sigma]\!]_{\mathsf{L},\nabla}=[a\mapsto b]$.

Using the right translation function, if we introduce the external λ-bindings we get the unsolvable higher-order unification problem $\lambda a.\lambda b.a\overset{?}{=}\lambda a.\lambda b.b$.

On the other hand, with this variant the translation of the solvable nominal equation of Example 3.4 would be

$$\begin{aligned}[\![\mathsf{P}]\!]_\mathsf{L}&=\Big[\!\!\Big[\{\mathsf{a.b.f(b,X_6)}\overset{?}{\approx}\mathsf{a.a.f(a,X_7)}\}\Big]\!\!\Big]_{\langle\mathsf{a,b}\rangle}\\&=\{\lambda a.\lambda b.f(b,X_6(a,b))\overset{?}{=}\lambda a.\lambda a.f(a,X_7(a,b))\}\end{aligned}$$

that is not a higher-order pattern unification problem (notice that Lemma 5.6 does not hold if we do not introduce the external λ-bindings).

The translation of its nominal most general solution is

$$[\![\sigma]\!]_{\mathsf{L},\nabla}=[\![[\mathsf{X_6}\mapsto(\mathsf{b\ a})\cdot\mathsf{X_7}]]\!]_{\langle\mathsf{a,b}\rangle,\{\mathsf{b}\#\mathsf{X_7}\}}=[X_6\mapsto\lambda a.\lambda b.X_7(b),X_7\mapsto\lambda a.\lambda b.X_7(a)]$$

In this case, $[\![\sigma]\!]_{\mathsf{L},\nabla}$ is a higher-order unifier of $[\![\mathsf{P}]\!]_\mathsf{L}$, as the variant of Theorem 5.12 predicts. However, it is not a most general unifier, and we are interested in translating most general solutions into most general solutions.

THEOREM 5.14. *If the equational nominal unification problem* P *is solvable, then, for any list* L *containing all atoms of* P *without repetitions,* $[\![\mathsf{P}]\!]_{\mathsf{L}}$ *is a solvable higher-order pattern unification problem.*

PROOF. The theorem is a direct consequence of Theorem 5.12. Notice that, if P is solvable, then there exists a solution $\langle\nabla,\sigma\rangle$. We can choose this solution satisfying the two requirements: all atoms used in σ are also used in P, and $\mathrm{Dom}(\sigma)$ is a superset of all variables of P. The inspection of Urban et al.'s algorithm [Urban et al. 2003; 2004] shows us that the solution that it computes does introduce new atoms not occurring in P. Moreover, we can extend the domain of σ with instantiations of the form $[\mathsf{X}\mapsto\mathsf{X}]$ to ensure the other restriction □

The opposite implication of Theorem 5.14 can not be directly proved from Theorem 5.12, because $[\![\mathsf{P}]\!]_{\mathsf{L}}$ could have solutions that are not of the form $[\![\sigma]\!]_{\mathsf{L},\nabla}$, for any solution $\langle\nabla,\sigma\rangle$ of P.

6. SOME PROPERTIES OF PATTERN UNIFICATION

In this section we prove some fundamental properties of Higher-Order Pattern Unification. In particular, we prove that we can express most general unifiers of pattern unification problems only using bound-variable names and types already used in the problem (Theorem 6.11). This property will be used in Section 7 to prove the existence of a translation of pattern unifiers into nominal unifiers, in Lemma 7.6. The proof of this property is based on a new pattern unification algorithm, described in Definition 6.6. This algorithm is based on the classical Nipkow's algorithm [Nipkow 1993]. However, this new algorithm is fully functional, in the sense that no new symbols need to be introduced. Its soundness and completeness are proved in Lemma 6.10.

In the following example we note that in the solution of pattern unification problems it is important to *save names* of bound variables. In the following we will distinguish between variables and variable names. For instance $\lambda x.\lambda x.x$ has three occurrences of variables, two distinct variables,[5] with one unique variable name. Notice that α-conversion preserves the number of variables, but may change the number of names.

Example 6.1. Consider the nominal problem $\mathsf{a.X} \overset{?}{\approx} \mathsf{a.f(b.Y)}$. Its translation using $\mathsf{L}=\langle\mathsf{a},\mathsf{b}\rangle$ is $\lambda a.\lambda b.\lambda a.X(a,b) \overset{?}{=} \lambda a.\lambda b.\lambda a.f(\lambda b.Y(a,b))$. An α-conversion results in $\lambda a.\lambda b.\lambda c.X(c,b) \overset{?}{=} \lambda a.\lambda b.\lambda c.f(\lambda d.Y(c,d))$ 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 would have to translate back $Y(c,d)$ as $(\mathsf{a\,c})(\mathsf{d\,b})\cdot\mathsf{Y}$. And, since substitutions like $[\mathsf{X}\mapsto\mathsf{t}]$ are translated as $[X\mapsto\lambda a.\lambda b.[\![\mathsf{t}]\!]_{\mathsf{L},\nabla}]$, we would have to translate back $[X\mapsto\lambda c.\lambda b.[\![\mathsf{t}]\!]_{\mathsf{L},\nabla}]$ as $[\mathsf{X}\mapsto(\mathsf{a\,c})\cdot\mathsf{t}]$. Therefore, our pattern unifier had to be translated back as $[\mathsf{X}\mapsto(\mathsf{a\,c})\cdot\mathsf{f}(\mathsf{d}.(\mathsf{a\,c})(\mathsf{d\,b})\cdot\mathsf{Y})]=[\mathsf{X}\mapsto\mathsf{f}(\mathsf{d}.(\mathsf{d\,b})\cdot\mathsf{Y})]$. The translation of this nominal unifier is $[\![\mathsf{X}\mapsto\mathsf{f}(\mathsf{d}.(\mathsf{d\,b})\cdot\mathsf{Y})]\!]_{\langle\mathsf{a},\mathsf{b}\rangle,\emptyset}=\lambda a.\lambda b.f(\lambda d.Y(a,d))$, that is α-equivalent to the original higher-order pattern unifier. Notice that we translate w.r.t. a list $\langle\mathsf{a},\mathsf{b}\rangle$ of atoms that does not contain the atom d that occurs in the nominal unifier.

Although this approximation to the definition of a back translation function seems to work, it relies in the change of new atoms by atoms contained in L. Hence, it depends on the availability of enough atoms in L. What would happen if we have to translate back a term of the form $Y(c_1,\dots,c_m)$ where m is greater than the length of L? We will prove that this situation never arises with most general unifiers. In fact, we will prove that we do not need to use new names of atoms to write such unifiers.

[5]This becomes clear if we use distinct names as in this equivalent λ-term $\lambda y.\lambda z.z$.

If we look at Nipkow's transformation rules described in Subsection 2.2, it seems that no new bound-variable names are introduced. However, this is not true. There are three places where their introduction is hidden. In the following we illustrate these cases.

(1) It is assumed that equations have the same most external λ-bindings, i.e. that they are of the form $\lambda\vec{x}.s \stackrel{?}{=} \lambda\vec{x}.t$. If this is not the case, we have to α-convert one of the sides. However, this is not always possible without introducing new bound-variable names. For instance, if we have the equation $\lambda x.\lambda y.\lambda y.X(x,y) \stackrel{?}{=} \lambda y.\lambda y.\lambda x.Y(x,y)$, after α-converting the two most external λ-binder, we get $\lambda x.\lambda y.\lambda y.X(x,y) \stackrel{?}{=} \lambda x.\lambda y.\lambda x.Y(x,y)$, that needs a new bound-variable name to obtain the same λ-binders in both sides, by means of α-conversion. Using a *new name* z we would get $\lambda x.\lambda y.\lambda z.X(x,z) \stackrel{?}{=} \lambda x.\lambda y.\lambda z.Y(z,y)$.

(2) In the flex-rigid rule the terms u_i may not be of first-order type. In this case, we need to η-expand some subterms. For instance, the rule transforms $\lambda x.X(x) \stackrel{?}{=} \lambda x.f(\lambda x.g(x))$ into the equation $\lambda x.X_1(x) \stackrel{?}{=} \lambda x.\lambda x.g(x)$ and the substitution $\big[X \mapsto \lambda x.f(X_1(x))\big]$. The left-hand side of the equation needs to be η-expanded, and we can not use the name x. Using a *new name* z, and α-converting we would get $\lambda x.\lambda z.X_1(x,z) \stackrel{?}{=} \lambda x.\lambda z.g(z)$.

(3) When we compute a substitution for a variable, it must be applied to all the occurrences of the variable, and this may involve a β-reduction. Some β-reductions need to introduce new names to avoid variable-captures. For instance, if we have the equations $\big\{\lambda x.\lambda y.X(x,y) \stackrel{?}{=} \lambda x.\lambda y.f(\lambda x.Y(x,y)),\ \lambda x.\lambda y.Z(x,y) \stackrel{?}{=} \lambda x.\lambda y.X(y,x)\big\}$, after solving the first one we get $\big[X \mapsto \lambda x.\lambda y.f(\lambda x.Y(x,y))\big]$ that must be substituted in the second equation. We get, $\lambda x.\lambda y.Z(x,y) \stackrel{?}{=} \lambda x.\lambda y.\big(\lambda x.\lambda y.f(\lambda x.Y(x,y))\big)(y,x)$. The β-reduction using the standard substitution algorithm introduces a *new name* z to avoid the capture of the variable x, giving $\lambda x.\lambda y.Z(x,y) \stackrel{?}{=} \lambda x.\lambda y.f(\lambda z.Y(z,x))$

In the following we show how we can overcome these problems. One of the ideas is using a kind of swapping for λ-calculus, instead of the usual substitution, like it is done in nominal terms. A similar notion of swapping is introduced in [Mendelzon et al. 2010] for the explicit substitution calculus.

Definition 6.2. Given two variables x, y, and a λ-term t, we define the swapping of x and y in t, noted by $(x\,y)\cdot t$ inductively as follows

$$
\begin{array}{ll}
(x\,y)\cdot x = y & \\
(x\,y)\cdot y = x & \\
(x\,y)\cdot z = z & \text{if } z \neq x, y \\
(x\,y)\cdot c = c & \\
(x\,y)\cdot\big(\lambda z.t\big) = \lambda\big((x\,y)\cdot z\big).\big((x\,y)\cdot t\big) & \\
(x\,y)\cdot\big(a(t_1,\dots,t_n)\big) = \big((x\,y)\cdot a\big)\big((x\,y)\cdot t_1,\dots,(x\,y)\cdot t_n\big) &
\end{array}
$$

where c is a constant and a is a constant or a variable.

Notice that this swapping is distinct from the swapping on nominal terms. In particular $(a\,b)\cdot X = X$, and we do not keep suspensions. In some cases its application results into an α-equivalent term, but in general the result is a different term.

Remark 6.3. In λ-calculus, following the Barendregt variable convention, operations are defined on classes of α-equivalent terms, rather than on particular terms. This, for instance, allows us to freely α-convert terms in substitutions in order to avoid variable capture. Therefore, (although it is often omitted) we have to prove that the operation is independent of the representative of the class that we take. The previous swapping operation is defined for particular terms. However, the following lemma ensures that it can be extended to α-equivalent classes of terms. Barendregt variable convention suggests to use distinct variable names for distinct variables. Here, since we try to avoid the introduction of new variable names, we do not use the convention, and work with particular terms.

LEMMA 6.4. *For any term t and variables x and y, we have*

$$(x\,y)\cdot t =_\alpha [x \mapsto y, y \mapsto x]t$$

where $[x \mapsto y, y \mapsto x]$ changes x by y and y by x in t, simultaneously.
In particular, if $x, y \notin \mathrm{FV}(t)$, then $(x\,y)\cdot t =_\alpha t$.

PROOF. By structural induction on t. For one of the cases of λ-abstraction, for instance, we have

$$\begin{array}{rll}
(x\,y)\cdot\lambda x.t & = \lambda y.(x\,y)\cdot t & \text{By ind. hyp.}\\
& = \lambda y.[x \mapsto y, y \mapsto x]t & \text{Let be } z \notin \mathrm{FV}(t) \cup \{x, y\}\\
& = \lambda y.[z \mapsto y][y \mapsto x][x \mapsto z]t & \text{Since } y \notin \mathrm{FV}([y \mapsto x][x \mapsto z]t)\\
& =_\alpha \lambda z.[y \mapsto x][x \mapsto z]t & \text{Since } z \neq x, y\\
& = [y \mapsto x]\lambda z.[x \mapsto z]t & \text{Since } z \notin \mathrm{FV}(t)\\
& =_\alpha [y \mapsto x]\lambda x.t & \text{Since } x \notin \mathrm{FV}(\lambda x.t)\\
& = [x \mapsto y, y \mapsto x]\lambda x.t &
\end{array}$$

□

LEMMA 6.5. *If $\vec{y}$ is a list of pairwise distinct variable names,*[6] *$|\vec{y}| = |\vec{x}| = n$ and $\{\vec{y}\} \cap \mathrm{FV}(\lambda\vec{x}.t) = \emptyset$, then*

$$(\lambda\vec{x}.t)(\vec{y}) = \Pi_n(\vec{x}, \vec{y})\cdot t$$

where $\Pi_n(\vec{x}, \vec{y})$ is a permutation on the names $\vec{x}, \vec{y}$ defined inductively as

$$\begin{array}{l}
\Pi_1(\langle x\rangle, \langle y\rangle) = (x\,y)\\
\Pi_n\big(\langle x_1, \dots, x_n\rangle, \langle y_1, \dots, y_n\rangle\big)\\
\qquad = \Pi_{n-1}\big(\langle (x_1\,y_1)\cdot x_2, \dots, (x_1\,y_1)\cdot x_n\rangle, \langle y_2, \dots, y_n\rangle\big)\cdot(x_1\,y_1)
\end{array}$$

PROOF. By induction on the length n of both vectors. Obviously, the variable x_1 is not free in $\lambda x_1.\lambda x_2, \dots, x_n.t$. By assumption, the variable y_1 is neither free in this term.
From $\mathrm{FV}(\lambda x_2, \dots, x_n.t) \subseteq \mathrm{FV}(\lambda\vec{x}.t) \cup \{x_1\}$, and $x_1, y_1 \notin \mathrm{FV}(\lambda\vec{x}.t)$, we have $\mathrm{FV}((x_1\,y_1)\cdot(\lambda x_2, \dots, x_n.t)) \subseteq \mathrm{FV}(\lambda\vec{x}.t) \cup \{y_1\}$. Since $y_1 \notin \{y_2, \dots, y_n\}$ and $\{\vec{y}\} \cap \mathrm{FV}(\lambda\vec{x}.t) = \emptyset$, we have $\{y_2, \dots, y_n\} \cap \mathrm{FV}((x_1\,y_1)\cdot(\lambda x_2, \dots, x_n.t)) = \emptyset$. Therefore, we can apply the induction hypothesis to the term $(x_1\,y_1)\cdot(\lambda x_2, \dots, x_n.t)$ and the vector $(y_2, \dots, y_n)$, obtaining

$$\begin{array}{rll}
(\lambda\vec{x}.t)(\vec{y}) & =_\alpha (\lambda y_1.(x_1\,y_1)\cdot(\lambda x_2, \dots, x_n.t))(y_1, y_2, \dots, y_n) & \text{By Lemma 6.4}\\
& =_\beta ((x_1\,y_1)\cdot(\lambda x_2, \dots, x_n.t))(y_2, \dots, y_n) & \text{By } \beta\text{-reduction}\\
& = (\lambda(x_1\,y_1)\cdot x_2, \dots, (x_1\,y_1)\cdot x_n.(x_1\,y_1)\cdot t)(y_2, \dots, y_n) & \text{Def. of swapping}\\
& = \Pi_{n-1}\big(\langle (x_1\,y_1)\cdot x_2, \dots, (x_1\,y_1)\cdot x_n\rangle, \langle y_2, \dots, y_n\rangle\big)\cdot(x_1\,y_1)\cdot t & \text{By ind. hyp.}\\
& = \Pi_n(\vec{x}, \vec{y})\cdot t &
\end{array}$$

□

Now we will describe a variant of the higher-order pattern unification algorithm of Section 2.2. In this variant, external λ-binders are α-converted explicitly and the flex-rigid rule has been replaced by a new rule where η-expansion is made explicit, i.e. the

[6]Notice that we do not require $\vec{x}$ to be pairwise distinct. If they are also pairwise distinct, then $\Pi_n(\vec{x}, \vec{y}) = (x_n\,y_n)\dots(x_1\,y_1)$.

terms u_i are base-typed, thus the right-hand side does not need to be η-expanded, like in the original rule. Moreover, β-redexes are removed using swappings, according to Lemma 6.5, since we are dealing with patterns.

Definition 6.6. We assume unoriented equations and define the following set of transformation rules over higher-order pattern equations:

α-transformation:

$$\lambda\vec{w}.\lambda x.t \overset{?}{=} \lambda\vec{w}.\lambda y.u \to \left\langle \lambda\vec{w}.\lambda x.t \overset{?}{=} \lambda\vec{w}.(x\ y)\cdot(\lambda y.u), [\,] \right\rangle \quad \text{if } x \notin \mathrm{FV}(u)$$

$$\lambda\vec{w}.\lambda x.t \overset{?}{=} \lambda\vec{w}.\lambda x.u \to \left\langle \lambda\vec{w}.t \overset{?}{=} \lambda\vec{w}.u, [\,] \right\rangle \quad \text{if } x \notin \mathrm{FV}(t) \text{ and } x \notin \mathrm{FV}(u)$$

$$\lambda\vec{w}.\lambda x.t \overset{?}{=} \lambda\vec{w}.\lambda x.u \to \left\langle \lambda\vec{w}.\lambda x.t \overset{?}{=} \lambda\vec{w}.\lambda x.u, [X \mapsto \lambda\vec{y}.Z(\vec{z})] \right\rangle \quad \text{if } x \notin \mathrm{FV}(t),\ X(\vec{y}) \text{ is a subterm of } u,\ x \in \{\vec{y}\} \text{ and } \{\vec{z}\} = \{\vec{y}\} \setminus \{x\}$$

Rigid-rigid:

$$\lambda\vec{w}.a(t_1,\dots,t_n) \overset{?}{=} \lambda\vec{w}.a(u_1,\dots,u_n) \to \left\langle \{\lambda\vec{w}.t_1 \overset{?}{=} \lambda\vec{w}.u_1, \dots, \lambda\vec{w}.t_n \overset{?}{=} \lambda\vec{w}.u_n\}, [\,] \right\rangle$$

Flex-rigid:

$$\lambda\vec{w}.X(\vec{x}) \overset{?}{=} \lambda\vec{w}.a(\lambda\vec{y_1}.u_1,\dots,\lambda\vec{y_m}.u_m) \to \Big\langle \Big\{ \lambda\vec{w}.\lambda\vec{y_1}.X_1(\vec{z_1}) \overset{?}{=} \lambda\vec{w}.\lambda\vec{y_1}.u_1\ , \dots, \lambda\vec{w}.\lambda\vec{y_m}.X_m(\vec{z_m}) \overset{?}{=} \lambda\vec{w}.\lambda\vec{y_m}.u_m \Big\}, [X \mapsto \lambda\vec{x}.a(\lambda\vec{y_1}.X_1(\vec{z_1}),\dots,\lambda\vec{y_m}.X_m(\vec{z_m}))] \Big\rangle$$

if $X \notin \mathrm{FV}(u_i)$, a is a constant or $a \in \{\vec{x}\}$, and $\{\vec{z_i}\} = \{\vec{x}\} \cup \{\vec{y_i}\}$, for $i = 1,\dots,m$.

Flex-flex:

$$\lambda\vec{w}.X(\vec{x}) \overset{?}{=} \lambda\vec{w}.X(\vec{y}) \to \left\langle \emptyset, [X \mapsto \lambda\vec{x}.Z(\vec{z})] \right\rangle \quad \text{where } \{\vec{z}\} = \{x_i \mid x_i = y_i\}$$

$$\lambda\vec{w}.X(\vec{x}) \overset{?}{=} \lambda\vec{w}.Y(\vec{y}) \to \left\langle \emptyset, [X \mapsto \lambda\vec{x}.Z(\vec{z}), Y \mapsto \lambda\vec{y}.Z(\vec{z})] \right\rangle \quad \text{where } X \neq Y \text{ and } \{\vec{z}\} = \{\vec{x}\} \cap \{\vec{y}\}$$

These transformations are applied as follows. The equation on the left-hand side is replaced by the equations in the first component of the right-hand side, and then the substitution in the second component of the right-hand side is applied to all the equations. If this substitution introduces β-redexes, they are removed using swappings, according to Lemma 6.5. Moreover, all the substitutions are composed to compute the resulting unifier. In other words, the transformation is applied as follows $\langle\{e\} \cup E, \sigma\rangle \to \langle\sigma'(E' \cup E)\downarrow_\beta, \sigma' \circ \sigma\rangle$, if we have a transformation $e \to \langle E', \sigma'\rangle$.

With the following examples, we illustrate how these rules solve the problems concerning the introduction of new bound variable names described previously, at the beginning of this section.

Example 6.7. Given the equation $\lambda x.\lambda y.\lambda y.X(x,y) \overset{?}{=} \lambda y.\lambda y.\lambda x.Y(x,y)$ the application of the first α-transformation rule gives us $\lambda x.\lambda y.\lambda y.X(x,y) \overset{?}{=} \lambda x.\lambda x.\lambda y.Y(y,x)$. A second application of this α-transformation gives us $\lambda x.\lambda y.\lambda y.X(x,y) \overset{?}{=} \lambda x.\lambda y.\lambda x.Y(x,y)$. Now, the first α-transformation rule is no longer applicable. However, we can apply the third α-transformation rule, that instantiates $[X \mapsto \lambda x.\lambda y.X'(y)]$, and gives the equation $\lambda x.\lambda y.\lambda y.X'(y) \overset{?}{=} \lambda x.\lambda y.\lambda x.Y(x,y)$. Now, applying the second α-transformation rule, we obtain $\lambda y.\lambda y.X'(y) \stackrel{?}{=} \lambda y.\lambda x.Y(x,y)$. Again, we can apply the third α-transformation rule, that instantiates $[Y \mapsto \lambda x.\lambda y.Y'(x)]$, and gives $\lambda y.\lambda y.X'(y) \stackrel{?}{=} \lambda y.\lambda x.Y'(x)$. The first α-transformation rule gives $\lambda y.\lambda y.X'(y) \stackrel{?}{=} \lambda y.\lambda y.Y'(y)$. Finally, the second α-transformation rule gives $\lambda y.X'(y) \stackrel{?}{=} \lambda y.Y'(y)$.

This last equation can be solved applying the second flex-flex rule. The resulting unifier is

$$\begin{aligned}&\big[X' \mapsto \lambda y.Z(y),\ Y' \mapsto \lambda y.Z(y)\big] \circ \big[Y \mapsto \lambda x.\lambda y.Y'(x)\big] \circ \big[X \mapsto \lambda x.\lambda y.X'(y)\big]\Big|_{\{X,Y\}}\\ &= \big[X \mapsto \lambda x.\lambda y.Z(y),\, Y \mapsto \lambda x.\lambda y.Z(x)\big]\end{aligned}$$

Example 6.8. The new flex-rigid rule transforms $\lambda x.X(x) \stackrel{?}{=} \lambda x.f(\lambda y.a)$ into the equation $\lambda x.\lambda y.X_1(x,y) \stackrel{?}{=} \lambda x.\lambda y.a$ and the substitution $[X \mapsto \lambda x.f(\lambda y.X_1(x,y))]$. The original flex-rigid rule would give us $\lambda x.X_1(x) \stackrel{?}{=} \lambda x.\lambda y.a$, that conveniently η-expanded using the same variable name y, results into the same equation. A further application of the flex-rigid rule solves the equation by $[X_1 \mapsto \lambda x.\lambda y.a]$.

In other cases, the resulting equation may be different. The new rule transforms $\lambda x.X(x) \stackrel{?}{=} \lambda x.f(\lambda x.g(x))$ into the equation $\lambda x.\lambda x.X_1(x) \stackrel{?}{=} \lambda x.\lambda x.g(x)$ and the substitution $[X \mapsto \lambda x.f(\lambda x.X_1(x))]$. However, the original flex-rigid rule would give us $\lambda x.X_1(x) \stackrel{?}{=} \lambda x.\lambda x.g(x)$ and the substitution $[X \mapsto \lambda x.f(X_1(x))]$. In the subsequent η-expansion we can not use the name x, and we need a *new name* z, and α-conversion of the right-hand side getting $\lambda x.\lambda z.X_1(x,z) \stackrel{?}{=} \lambda x.\lambda z.g(z)$. Both equations are obviously distinct. However, to solve this second equation, X_1 can not use the first argument, because it is not used in the right-hand side. Therefore, we can instantiate $X_1 \mapsto \lambda x.\lambda y.X_1'(y)$, and α-convert the new variable name z, getting the same equation as with the new flex-rigid rule.

Example 6.9. Given the equations $\big\{\lambda x.\lambda y.X(x,y) \stackrel{?}{=} \lambda x.\lambda y.f(\lambda x.Y(x,y)),$ $\lambda x.\lambda y.Z(x,y) \stackrel{?}{=} \lambda x.\lambda y.X(y,x)\big\}$, after solving the first equation and replacing $\big[X \mapsto \lambda x.\lambda y.f(\lambda x.Y(x,y))\big]$ into the second one, we get $\lambda x.\lambda y.Z(x,y) \stackrel{?}{=} \lambda x.\lambda y.\big((\lambda x.\lambda y.f(\lambda x.Y(x,y)))(y,x)\big)$. By Lemma 6.5, we can β-reduce using swappings, instead of the usual standard substitution. The permutation will be $\Pi_2(\langle x,y\rangle,\langle y,x\rangle) = \Pi_1\big(\langle (x\,y)\cdot y\rangle,\langle x\rangle\big)\cdot(x\,y) = (x\,x)\cdot(x\,y) = (x\,y)$, and the result of the β-reduction will be

$$\big(\lambda x.\lambda y.f\big(\lambda x.Y(x,y)\big)\big)(y,x) =_\beta (x\,y)\cdot f\big(\lambda x.Y(x,y)\big) = f\big(\lambda y.Y(y,x)\big)$$

LEMMA 6.10. *The algorithm described in Definition 6.6 is sound and complete and computes a most-general higher-order pattern unifier whenever it exists, when names of free and bound variables are disjoint.*

PROOF. The algorithm computes basically the same most general unifiers than the Nipkow's algorithm.

The fact that we use swapping instead of substitution to remove β-redexes is not a problem according to Lemma 6.5. We will obtain a term that is α-equivalent to the one that we would obtain with the traditional capture-avoiding substitution. Notice that in the lemma we require arguments of free variables (the sequence $\vec{y}$) to be a list of distinct bound variables. This is ensured in the case of higher-order pattern unification, but it is not true in the general λ-calculus. The algorithm preserves the disjointness of bound and free variable names. Therefore, the other condition of the lemma $\{\vec{y}\} \cap \mathrm{FV}(\lambda\vec{x}.t)$ is also satisfied.

In the third α-transformation rule, if $x \notin \mathrm{FV}(t)$ and $x \in \mathrm{FV}(u)$ and the equation is solvable, then x must occur in u just below a free variable, as one of its arguments, and this free variable must be instantiated by a term that does not use this argument. Notice also that the three α-transformation rules, when the equation is solvable, succeed in making the lists of most external λ-bindings equal in both sides of the equation.

In the case of the flex-rigid rule, we may obtain an equation $\lambda\vec{x}.X_i(x_1,\dots,x_n) \stackrel{?}{=} \lambda\vec{x}.\lambda\vec{y}.u'_i$ that needs to be η-expanded, and where $\{x_1,\dots,x_n\} \cap \{\vec{y}\} \neq \emptyset$. Let be $\{x'_1,\dots,x'_{n'}\} = \{x_1,\dots,x_n\} \setminus \vec{y}$, i.e. the sequence of variables x_i not in $\vec{y}$. In any solution of this equation X_i can not use the variables of the intersection of $\{x_1,\dots,x_n\} \cap \{\vec{y}\}$. Therefore, we can extend the solution with $X_i \mapsto \lambda x_1 \dots x_n.\lambda\vec{y}.X'_i(x'_1,\dots,x'_n,\vec{y})$, and get the equation $\lambda\vec{x}.\lambda\vec{y}.X'_i(x'_1,\dots,x'_{n'},\vec{y}) \stackrel{?}{=} \lambda\vec{x}.\lambda\vec{y}.u'_i$.

The flex-flex and rigid-rigid rules are the same as in Nipkow's algorithm. □

THEOREM 6.11. *Let P be a solvable pattern unification problem, where the set of free and bound variable names are disjoint, and let $\langle a_1,\dots,a_n\rangle$ be a list of the names of bound variables of the problem. Then, there exists a most general unifier σ such that*

(1) *σ does not use other bound-variable names than the ones already used in the problem, i.e than $\{a_1,\dots,a_n\}$.*

If in the original problem all bound variables with the same name have the same type, i.e. we have a type τ_i for every bound variable name a_i, then

(2) *the same applies to σ, i.e. any bound variable of σ with name a_i has type τ_i, and*
(3) *any free variable X occurring in σ has type $\nu_1 \to \dots \to \nu_m \to \nu$, where $\langle \nu_1,\dots,\nu_m\rangle$ is a sublist of $\langle \tau_1,\dots,\tau_n\rangle$.*

PROOF. By Lemma 6.10 with the new transformation rules we obtain most general unifiers for solvable pattern unification problems. Then, by simple inspection of the new transformation rules, where all bound variable names in the right-hand sides of the rules are already present in the left-hand sides, we have that new equations and substitutions do not introduce new names. In addition, since names of free and bound variables are distinct, β-reductions due to substitution applications satisfy conditions of Lemma 6.5, therefore we can conclude that we do not need new bound variable names due to β-reductions either.

Notice also that in these rules, when we introduce a new bound variable in the right-hand side, with a name already used in the left-hand side, both variables have the same type. When, we swap two variable names in an α-conversion or in a β-reduction, they have also the same type.

Finally, let σ' be any most general unifier not using other bound variable names than the ones used in P, i.e. $a_1,\dots,a_n$. For every variable X occurring free in σ, chose one of their occurrences. This will be of the form $X(b_1,\dots,b_m)$, where $\{b_1,\dots,b_m\} \subseteq \{a_1,\dots,a_n\}$ and the b_i are pairwise distinct. Let $\langle b_{\pi(1)},\dots,b_{\pi(m)}\rangle$ be a sublist of $\langle a_1,\dots,a_n\rangle$. Then composing σ' with $[X' \mapsto \lambda b_1.\cdots.\lambda b_m.X(b_{\pi(1)},\dots,b_{\pi(m)})]$, for every variable X, we get another most general unifier fulfilling the requirements of the third statement of the lemma. Notice that, although not all occurrences of X have the same parameters, it does not matter which one we chose because all them have the same type. □

7. THE REVERSE TRANSLATION

As we have shown, Theorem 5.12 is not enough to prove that, if $[\![\mathsf{P}]\!]_{\mathsf{L}}$ is solvable, then P is solvable. The proof of this implication (Theorem 7.7) is the main objective of this section. We define a back-translation function, parametric on a list of atoms and a freshness environment, for terms and substitutions (Definitions 7.1 and 7.2, respectively). This function is not always defined. When there exists a freshness environment ∇ such that $[\![t]\!]^{-1}_{\mathsf{L},\nabla}$ is defined (respectively for unifiers), then we say that t is ∇-compatible (Definition 7.3). In Lemma 7.6 we prove the ∇-compatibility of most general pattern unifiers, that is the base of the proof of Theorem 7.7. In Section 8, we will prove that the back-translation function preserves most generality.

Definition 7.1. Let L be a list of atoms, and ∇ be a freshness environment. The back-translation function is defined on λ-terms in η-long β-normal form as follows:

$$\begin{array}{ll}
[\![a]\!]^{-1}_{\mathsf{L},\nabla} = \mathsf{a} & \\
[\![f(t_1,\dots,t_n)]\!]^{-1}_{\mathsf{L},\nabla} = \mathsf{f}([\![t_1]\!]^{-1}_{\mathsf{L},\nabla},\dots,[\![t_n]\!]^{-1}_{\mathsf{L},\nabla}) & \\
[\![\lambda a.t]\!]^{-1}_{\mathsf{L},\nabla} = \mathsf{a}.\,[\![t]\!]^{-1}_{\mathsf{L},\nabla} & \\
[\![X(c_1,\dots,c_m)]\!]^{-1}_{\mathsf{L},\nabla} = \pi^{-1}\cdot\mathsf{X} & \text{where } \pi \text{ is a permutation on } \mathsf{L} \text{ satisfying} \\
& \langle \pi\cdot\mathsf{c_1},\dots,\pi\cdot\mathsf{c_m}\rangle \text{ is a sublist of } \mathsf{L} \text{ such that} \\
& \pi\cdot\mathsf{c_i}\#\mathsf{X}\notin\nabla \text{ and } \mathsf{c_i} \text{ and } \pi\cdot\mathsf{c_i} \text{ have the same sort}
\end{array}$$

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 nominal variable, and the permutation π^{-1} 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, $[\![\lambda x.t]\!]^{-1}_{\mathsf{L},\nabla}$ is not defined when x is not base typed, or $[\![x(t_1,\dots,t_n)]\!]^{-1}_{\mathsf{L},\nabla}$ is not defined when x is a bound variable. It also depends on the list L and environment ∇. For instance $[\![X(b,a)]\!]^{-1}_{\langle\mathsf{a}\rangle,\emptyset}$ and $[\![X(b,a)]\!]^{-1}_{\langle\mathsf{a},\mathsf{b}\rangle,\{\mathsf{a}\#\mathsf{X}\}}$ are undefined, whereas $[\![X(b,a)]\!]^{-1}_{\langle\mathsf{a},\mathsf{b}\rangle,\emptyset} = (\mathsf{a}\;\mathsf{b})\cdot\mathsf{X}$.

Notice also that the permutation π is not completely determined by the side condition of the fourth equation. For instance, given $\mathsf{L} = \langle\mathsf{a_1},\mathsf{a_2},\mathsf{a_3}\rangle$ as the list of atoms, all them of the same sort, to define $[\![X(a_1)]\!]^{-1}_{\mathsf{L},\{\mathsf{a_1}\#\mathsf{X},\mathsf{a_2}\#\mathsf{X}\}} = \pi^{-1}\cdot\mathsf{X}$ the condition requires $\pi\cdot\mathsf{a_1} = \mathsf{a_3}$, but then, we can choose $\pi\cdot\mathsf{a_2} = \mathsf{a_1}$ and $\pi\cdot\mathsf{a_3} = \mathsf{a_2}$, or vice versa $\pi\cdot\mathsf{a_2} = \mathsf{a_2}$ and $\pi\cdot\mathsf{a_3} = \mathsf{a_1}$. Therefore, $[\![t]\!]^{-1}_{\mathsf{L},\nabla}$ is nondeterministically defined.

For λ-substitutions the back-translation is defined as follows.

Definition 7.2. Let $\mathsf{L} = \langle\mathsf{a_1},\dots,\mathsf{a_n}\rangle$ be a list of atoms, and ∇ be a freshness environment. The back-translation function is defined on λ-substitutions as follows.

$$[\![\sigma]\!]^{-1}_{\mathsf{L},\nabla} = \bigcup_{X\in\mathrm{Dom}(\sigma)} \left[\mathsf{X}\mapsto[\![\sigma(X)(a_1,\dots,a_n)]\!]^{-1}_{\mathsf{L},\nabla}\right]$$

Notice that if $\sigma(X)(a_1,\dots,a_n)$ is not a well-typed λ-term, or $[\![\sigma(X)(a_1,\dots,a_n)]\!]^{-1}_{\mathsf{L},\nabla}$ is not defined for some $X\in\mathrm{Dom}(\sigma)$, then $[\![\sigma]\!]^{-1}_{\mathsf{L},\nabla}$ is not defined.

We introduce the following notion to describe which λ-terms and substitutions have reverse translation w.r.t. a freshness environment.

Definition 7.3. Given a λ-term t (resp. λ-substitution σ), a list of atoms L and a freshness environment ∇, we say that t (resp. σ) is $\langle\mathsf{L},\nabla\rangle$-compatible if $[\![t]\!]^{-1}_{\mathsf{L},\nabla}$ (resp. $[\![\sigma]\!]^{-1}_{\mathsf{L},\nabla}$) is defined.

LEMMA 7.4. *For any λ-term t, list of atoms L and freshness environment ∇, if t is $\langle\mathsf{L},\nabla\rangle$-compatible, then* $\left[\!\!\left[[\![t]\!]^{-1}_{\mathsf{L},\nabla}\right]\!\!\right]_{\mathsf{L},\nabla} = t$.

For every λ-substitution σ, list of atoms L and freshness environment ∇, if σ is $\langle \mathsf{L}, \nabla\rangle$-compatible, then $\left[\!\left[[\![\sigma]\!]^{-1}_{\mathsf{L},\nabla} \right]\!\right]_{\mathsf{L},\nabla} = \sigma$.

PROOF. Let $\mathsf{L} = \langle \mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n}\rangle$ be a list of atoms. The existence of $[\![t]\!]^{-1}_{\mathsf{L},\nabla}$ restricts the form of t to five cases. For the first four, the proof is trivial. In the case $t = X(c_1, \cdots, c_m)$, we have

$$\begin{aligned} \left[\!\left[[\![X(c_1,\cdots,c_m)]\!]^{-1}_{\mathsf{L},\nabla} \right]\!\right]_{\mathsf{L},\nabla} &= [\![\pi^{-1}\cdot \mathsf{X}]\!]_{\mathsf{L},\nabla} \\ &= X\left([\![\pi^{-1}\cdot\pi\cdot \mathsf{c}_1]\!]_{\mathsf{L},\nabla}, \cdots, [\![\pi^{-1}\cdot\pi\cdot \mathsf{c}_\mathsf{m}]\!]_{\mathsf{L},\nabla} \right) \\ &= X(c_1,\cdots,c_m) \end{aligned}$$

where π is a permutation on L satisfying $\langle \pi\cdot \mathsf{c}_1, \dots, \pi\cdot\mathsf{c}_\mathsf{m}\rangle$ is a sublist of L such that $\pi\cdot\mathsf{c}_\mathsf{i}\#\mathsf{X} \notin \nabla$ and c_i and $\pi\cdot\mathsf{c}_\mathsf{i}$ have the same sort.

For the second statement, by Definitions 7.2 and 5.7 we have

$$\begin{aligned} \left[\!\left[[\![\sigma]\!]^{-1}_{\mathsf{L},\nabla} \right]\!\right]_{\mathsf{L},\nabla} &= \left[\!\left[\bigcup_{\mathsf{X}\in\mathrm{Dom}(\sigma)} \left[\mathsf{X} \mapsto [\![\sigma(X)(a_1,\cdots,a_n)]\!]^{-1}_{\mathsf{L},\nabla}\right] \right]\!\right]_{\mathsf{L},\nabla} \\ &= \bigcup_{X\in\mathrm{Dom}(\sigma)} \left[X \mapsto \lambda a_1\cdots a_n. \left[\!\left[[\![\sigma(X)(a_1,\cdots,a_n)]\!]^{-1}_{\mathsf{L},\nabla} \right]\!\right]_{\mathsf{L},\nabla} \right] \\ &= \bigcup_{X\in\mathrm{Dom}(\sigma)} \left[X \mapsto \lambda a_1\cdots a_n.\sigma(X)(a_1,\cdots,a_n)\right] \\ &= \bigcup_{X\in\mathrm{Dom}(\sigma)} \left[X \mapsto \sigma(X)\right] = \sigma \end{aligned}$$

Where we make use of the first statement to prove $\left[\!\left[[\![\sigma(X)(a_1,\cdots,a_n)]\!]^{-1}_{\mathsf{L},\nabla} \right]\!\right]_{\mathsf{L},\nabla} = \sigma(X)(a_1,\cdots,a_n)$. Notice that, if σ is $\langle \mathsf{L},\nabla\rangle$-compatible, then $\sigma(X)(a_1,\cdots,a_n)$ is also $\langle \mathsf{L},\nabla\rangle$-compatible. □

Given a pattern unifier, in order to reconstruct the corresponding nominal unifier, we have several degrees of freedom. We start with higher-order pattern unifier σ with a restricted use of names of bound variables. Then, we will construct a freshness environment ∇ such that σ is $\langle \mathsf{L},\nabla\rangle$-compatible. This construction is described in the proof of Lemma 7.6, and it is nondeterministic. The corresponding nominal solution is then $\langle \nabla, [\![\sigma]\!]^{-1}_{\mathsf{L},\nabla}\rangle$. Moreover, $[\![\sigma]\!]^{-1}_{\mathsf{L},\nabla}$ is not uniquely defined. The following example illustrates these degrees of freedom in this back-translation.

Example 7.5. Consider the nominal unification problem

$$\mathsf{P} = \{\mathsf{a}.\mathsf{a}.\mathsf{X} \overset{?}{\approx} \mathsf{c}.\mathsf{a}.\mathsf{X}\ ,\ \mathsf{a}.\mathsf{b}.\mathsf{X} \overset{?}{\approx} \mathsf{b}.\mathsf{a}.(\mathsf{a}\,\mathsf{b})\cdot\mathsf{X}\}$$

where all atoms and variables are of the same sort. Let $\mathsf{L} = \langle \mathsf{a},\mathsf{b},\mathsf{c}\rangle$ be a list of the atoms of the problem. It is translated as

$$\begin{aligned} [\![\mathsf{P}]\!]_{\mathsf{L}} = \{\ & \lambda a.\lambda b.\lambda c.\lambda a.\lambda a.X(a,b,c) \overset{?}{=} \lambda a.\lambda b.\lambda c.\lambda c.\lambda a.X(a,b,c)\ , \\ & \lambda a.\lambda b.\lambda c.\lambda a.\lambda b.X(a,b,c) \overset{?}{=} \lambda a.\lambda b.\lambda c.\lambda b.\lambda a.X(b,a,c)\ \} \end{aligned}$$

Most general higher-order pattern unifiers are

$$\sigma_1 = [X \mapsto \lambda a.\lambda b.\lambda c.Z(a,b)]$$

and

$$\sigma_2 = [X \mapsto \lambda a.\lambda b.\lambda c.Z(b,a)]$$

which are equivalent.

Following the construction described in the forthcoming proof of Lemma 7.6, for every variable $Z : \tau_1 \to \cdots \tau_m \to \tau_0$ occurring in σ, we construct a sublist of atoms $\mathsf{L_Z} = \langle \mathsf{b_1}, \dots, \mathsf{b_m} \rangle$ satisfying $\mathsf{b_j} : [\![\tau_j]\!]^{-1}$, for every $j = 1, \dots, m$. In our example, we can choose among three possibilities $\mathsf{L}^1_\mathsf{Z} = \langle \mathsf{a}, \mathsf{b} \rangle$, $\mathsf{L}^2_\mathsf{Z} = \langle \mathsf{a}, \mathsf{c} \rangle$ or $\mathsf{L}^3_\mathsf{Z} = \langle \mathsf{b}, \mathsf{c} \rangle$. We construct $\nabla = \{\mathsf{a}\#\mathsf{Z} \mid Z \text{ occurs in } \sigma \wedge \mathsf{a} \in \mathsf{L} \setminus \mathsf{L_Z}\}$.

From the two pattern unifiers σ_i, and the three lists $\mathsf{L}^\mathsf{j}_\mathsf{Z}$ we can construct six possible nominal unifiers:

	σ_1	σ_2
L^1_Z	$\langle \{c\#Z\}\ ,\ [X \mapsto \begin{pmatrix} a & b & c \\ a & b & c \end{pmatrix}^{-1} \cdot Z] \rangle$	$\langle \{c\#Z\}\ ,\ [X \mapsto \begin{pmatrix} a & b & c \\ b & a & c \end{pmatrix}^{-1} \cdot Z] \rangle$
L^2_Z	$\langle \{b\#Z\}\ ,\ [X \mapsto \begin{pmatrix} a & b & c \\ a & c & b \end{pmatrix}^{-1} \cdot Z] \rangle$	$\langle \{b\#Z\}\ ,\ [X \mapsto \begin{pmatrix} a & b & c \\ c & a & b \end{pmatrix}^{-1} \cdot Z] \rangle$
L^3_Z	$\langle \{a\#Z\}\ ,\ [X \mapsto \begin{pmatrix} a & b & c \\ b & c & a \end{pmatrix}^{-1} \cdot Z] \rangle$	$\langle \{a\#Z\}\ ,\ [X \mapsto \begin{pmatrix} a & b & c \\ c & b & a \end{pmatrix}^{-1} \cdot Z] \rangle$

The permutations can be written as swappings obtaining:

	σ_1	σ_2
L^1_Z	$\langle \{c\#Z\}\ ,\ [X \mapsto Z] \rangle$	$\langle \{c\#Z\}\ ,\ [X \mapsto (a\,b)\cdot Z] \rangle$
L^2_Z	$\langle \{b\#Z\}\ ,\ [X \mapsto (b\,c)\cdot Z] \rangle$	$\langle \{b\#Z\}\ ,\ [X \mapsto (a\,b)(b\,c)\cdot Z] \rangle$
L^3_Z	$\langle \{a\#Z\}\ ,\ [X \mapsto (a\,c)(b\,c)\cdot Z] \rangle$	$\langle \{a\#Z\}\ ,\ [X \mapsto (a\,c)\cdot Z] \rangle$

All these nominal unifiers are most general and equivalent. Notice that these are *all* the most general nominal unifiers.

LEMMA 7.6. *For every equational nominal unification problem* P*, let* L *be a list containing all atoms of* P *without repetitions. If the pattern unification problem* $[\![\mathsf{P}]\!]_\mathsf{L}$ *is solvable, then there exists a freshness environment* ∇*, and a most general pattern unifier* σ*, such that* σ *is* $\langle \mathsf{L}, \nabla \rangle$*-compatible.*

PROOF. Let $\mathsf{L} = \langle \mathsf{a_1}, \dots, \mathsf{a_n} \rangle$, and let τ_i be the sort of a_i, for $i = 1, \dots, n$. The most general unifier σ is chosen, accordingly to Theorem 6.11, as a unifier not using other bound variable names than the ones used in $[\![\mathsf{P}]\!]_\mathsf{L}$. Moreover, since all bound variables of $[\![\mathsf{P}]\!]_\mathsf{L}$ with the same name a_i have the same type $[\![\tau_\mathsf{i}]\!]$, the same happens in σ, and all free variables Z occurring in σ have a type of the form $Z : [\![\tau_{\mathsf{i}_1}]\!] \to \dots \to [\![\tau_{\mathsf{i}_\mathsf{m}}]\!] \to [\![\delta]\!]$, for some indexes satisfying $1 \le i_1 < \cdots < i_m \le n$. Notice that there could be more than one set of indexes satisfying this condition.

The freshness environment ∇ is constructed as follows. For any variable $Z : [\![\tau_{\mathsf{i}_1}]\!] \to \dots [\![\tau_{\mathsf{i}_\mathsf{m}}]\!] \to [\![\delta]\!]$ occurring[7] in σ , let be $\mathsf{L_Z} = \langle \mathsf{a}_{\mathsf{i}_1}, \dots, \mathsf{a}_{\mathsf{i}_\mathsf{m}} \rangle$. This will be a sublist of the atoms of L. Then,

$$\nabla = \{\mathsf{a}\#\mathsf{Z} \mid Z \text{ occurs in } \sigma \wedge \mathsf{a} \in \mathsf{L} \setminus \mathsf{L_Z}\}$$

We prove that $\sigma(X)(a_1, \dots, a_n)$ is $\langle \mathsf{L}, \nabla \rangle$-compatible, for any $X \in \mathrm{Dom}(\sigma)$.

Since σ is most general $\mathrm{Dom}(\sigma)$ only contains variables X of $[\![\mathsf{P}]\!]_\mathsf{L}$. All these variables have type $[\![\tau_1]\!] \to \cdots \to [\![\tau_\mathsf{n}]\!] \to [\![\delta]\!]$, where δ is the sort of X. Therefore, $\sigma(X)(a_1, \dots, a_n)$

[7]We say that X occurs in σ, if X occurs free in $\sigma(Y)$, for some $Y \in \mathrm{Dom}(\sigma)$.

is a well-typed λ-term. Now we prove that this term is $\langle \mathsf{L}, \nabla\rangle$-compatible by structural induction.

By Theorem 6.11, $\sigma(X)$ 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. Terms formed by a constant or free variable are particular cases of applications with $m = 0$, studied bellow.

All λ-abstractions will be of the form $\lambda a_i.t$, where $a_i = [\![\mathsf{a_i}]\!]_{\mathsf{L},\nabla}$. This ensure that its translation back is possible, if the body of the λ-abstractions is $\langle \mathsf{L}, \nabla\rangle$-compatible.

All applications are of the form $f(t_1, \dots, t_m)$ where f is a constant of the original nominal problem (since σ is most general), or of the form $X(a_{i_1}, \dots, a_{i_m})$ where X is a free variable and $a_{i_1}, \dots, a_{i_m}$ are distinct bound variables. Notice that we can no have terms of the form $a_i(t_1, \dots, t_n)$ where a_i is a bound variable, because all these bound variables have basic types. In the first case, the application is $\langle \mathsf{L}, \nabla\rangle$-compatible if arguments are. In the second case, let $X : [\![\tau_{\mathsf{j_1}}]\!] \to \dots \to [\![\tau_{\mathsf{j_m}}]\!] \to [\![\delta]\!]$, for some indexes satisfying $1 \leq j_1 < \dots < j_m \leq n$. using the ∇ constructed before, we can translate back $X(a_{i_1}, \dots, a_{i_m})$ as $\pi^{-1}{\cdot}\mathsf{X}$, for some π satisfying $\pi(\mathsf{a_{i_k}}) = \mathsf{a_{j_k}}$, for all $k = 1, \dots, m$. Notice that $\mathsf{a_{j_k}}$ and $\mathsf{a_{i_k}}$ have the same sort τ_{j_k}. Hence, this second kind of applications is also $\langle \mathsf{L}, \nabla\rangle$-compatible. $\square$

THEOREM 7.7. *For every equational nominal unification problem* P*, and any list* L *containing all atoms of* P *without repetitions, if the pattern unification problem* $[\![\mathsf{P}]\!]_{\mathsf{L}}$ *is solvable, then* P *is also solvable.*

PROOF. By Lemma 7.6, if $[\![\mathsf{P}]\!]_{\mathsf{L}}$ is solvable then there exist a most general unifier σ of $[\![\mathsf{P}]\!]_{\mathsf{L}}$, and a freshness environment ∇ such that $\langle \nabla, [\![\sigma]\!]^{-1}_{\mathsf{L},\nabla}\rangle$ is defined. W.l.o.g. assume that $\mathrm{Dom}(\sigma) = \mathrm{Vars}([\![\mathsf{P}]\!]_{\mathsf{L}})$ and hence, according to Definition 7.2, $\mathrm{Dom}([\![\sigma]\!]^{-1}_{\mathsf{L},\nabla}) = \mathrm{Vars}(\mathsf{P})$. By Lemma 7.4, we have $\left[\!\!\left[[\![\sigma]\!]^{-1}_{\mathsf{L},\nabla} \right]\!\!\right]_{\mathsf{L},\nabla} = \sigma$, which solves $[\![\mathsf{P}]\!]_{\mathsf{L}}$. Hence, by Theorem 5.12, $\langle \nabla, [\![\sigma]\!]^{-1}_{\mathsf{L},\nabla}\rangle$ solves P. $\square$

From Corollary 4.5, Lemma 5.6 and Theorems 5.14 and 7.7 we conclude the following result.

COROLLARY 7.8. *Nominal Unification is quadratic reducible to Higher-Order Pattern Unification.*

8. CORRESPONDENCE BETWEEN UNIFIERS

In this section we establish a correspondence between the solutions of a nominal unification problem and their translations. We prove that the translation and the reverse translation functions are monotone, in the sense that they translate more general solutions into more general solutions (Lemma 8.5). Therefore, both translate most general solutions into most general solutions (Theorem 8.7).

We start by generalizing the translation of a nominal substitution w.r.t. a freshness environment, to respect the translation of a nominal substitution w.r.t. *two* freshness environments, and similarly for the reverse translation.

Definition 8.1. Given a list of atoms L, a nominal substitution σ, and two freshness environments ∇_1 and ∇_2, satisfying $\nabla_2 \vdash \sigma(\nabla_1)$, we define

$$[\![\sigma]\!]^{\nabla_1}_{\mathsf{L},\nabla_2} = \bigcup_{\mathsf{X} \in \mathrm{Dom}(\sigma)} [X \mapsto \lambda a_1, \dots, a_n.[\![\sigma(\mathsf{X})]\!]_{\mathsf{L},\nabla_2}]$$

where $\langle \mathsf{a_1}, \dots, \mathsf{a_n}\rangle = \langle \mathsf{a} \in \mathsf{L} \mid \mathsf{a}\#\mathsf{X} \notin \nabla_1\rangle$.

Given a list of atoms L, a pattern substitution σ, and two freshness environments ∇ and ∇', we define

$$[\![\sigma]\!]^{\text{-1}\,\nabla_1}_{\mathsf{L},\nabla_2} = \bigcup_{X\in\mathrm{Dom}(\sigma)} \Big[\mathsf{X} \mapsto [\![\sigma(X)(a_1,\dots,a_n)]\!]^{\text{-1}}_{\mathsf{L},\nabla_2}\Big]$$

where $\langle \mathsf{a}_1,\dots,\mathsf{a}_n\rangle = \langle \mathsf{a}\in\mathsf{L} \mid \mathsf{a}\#\mathsf{X}\notin\nabla_1\rangle$.

We say that σ is $\langle\mathsf{L},\nabla_1\to\nabla_2\rangle$-compatible if $[\![\sigma]\!]^{\text{-1}\,\nabla_1}_{\mathsf{L},\nabla_2}$ is defined.

Notice that this definition generalizes Definition 5.7 because $[\![\sigma]\!]_{\mathsf{L},\nabla} = [\![\sigma]\!]^{\emptyset}_{\mathsf{L},\nabla}$, and Definition 7.2 because, $[\![\sigma]\!]^{\text{-1}}_{\mathsf{L},\nabla} = [\![\sigma]\!]^{\text{-1}\,\emptyset}_{\mathsf{L},\nabla}$.

The following lemmas are generalizations of Lemmas 5.10 and 7.4, respectively. Their proofs are also straightforward generalizations.

LEMMA 8.2. *For any list of atoms* L*, nominal substitution* σ*, freshness environments* ∇_1 *and* ∇_2*, and nominal term* t*, satisfying that all atoms of* t *are in* L*,* $\nabla_2\vdash\sigma(\nabla_1)$ *and* $\mathrm{Vars}(\mathsf{t})\subseteq\mathrm{Dom}(\sigma)$*, we have*

$$[\![\sigma]\!]^{\nabla_1}_{\mathsf{L},\nabla_2}([\![\mathsf{t}]\!]_{\mathsf{L},\nabla_1}) = [\![\sigma(\mathsf{t})]\!]_{\mathsf{L},\nabla_2}$$

LEMMA 8.3. *For any list of atoms* L*,* λ*-substitution* σ *and freshness environment* ∇_1 *and* ∇_2*, if* σ *is* $\langle\mathsf{L},\nabla_1\to\nabla_2\rangle$*-compatible, then*

$$\Big[\!\!\Big[[\![\sigma]\!]^{\text{-1}\,\nabla_1}_{\mathsf{L},\nabla_2}\Big]\!\!\Big]^{\nabla_1}_{\mathsf{L},\nabla_2} = \sigma$$

If a λ-substitution σ_1 is more general than another σ_2, then there exists a substitution σ_3 that satisfies $\sigma_2 = \sigma_3\circ\sigma_1$. The following lemma states that this substitution can be used to construct a nominal substitution as the reverse translation of σ_3 that we will use, in Lemma 8.5, to prove that reverse translation of σ_1 is more general than the reverse translation of σ_2.

LEMMA 8.4. *For any list of atoms* L*, pair of* λ*-substitutions* σ_1 *and* σ_2 *and pair of freshness environments* ∇_1 *and* ∇_2*, if* σ_1 *is* $\langle\mathsf{L},\nabla_1\rangle$*-compatible,* σ_2 *is* $\langle\mathsf{L},\nabla_2\rangle$*-compatible, and* σ_1 *is more general than* σ_2*, then there exists a* λ*-substitution* σ_3 *such that*

(*1*) $\sigma_2 = \sigma_3\circ\sigma_1|_{\mathrm{Dom}(\sigma_1)\cup\mathrm{Dom}(\sigma_2)}$
(*2*) σ_3 *is* $\langle\mathsf{L},\nabla_1\to\nabla_2\rangle$*-compatible, and*
(*3*) $\nabla_2\vdash[\![\sigma_3]\!]^{\text{-1}\,\nabla_1}_{\mathsf{L},\nabla_2}(\nabla_1)$.

PROOF. (1) The first statement follows from σ_1 being more general than σ_2. However, w.l.o.g. we take a σ_3 that only instantiates variables occurring in σ_1 or belonging to $\mathrm{Dom}(\sigma_2)$.
(2) For all $X\in\mathrm{Dom}(\sigma_3)$, let $\langle\mathsf{a}_1,\dots,\mathsf{a}_n\rangle = \langle\mathsf{a}\in\mathsf{L}\mid\mathsf{a}\#\mathsf{X}\notin\nabla_1\rangle$. Now, X occurs in σ_1 or $X\in\mathrm{Dom}(\sigma_2)$. In the first case, since σ_1 is $\langle\mathsf{L},\nabla_1\rangle$-compatible and we are dealing with higher-order pattern substitutions, X occurs in σ_1 in (at least one) subterm of the form $X(b_1,\dots,b_n)$, where b_i are distinct bound variables with names in $\langle\mathsf{a}_1,\dots,\mathsf{a}_n\rangle$, and a_i and b_i have the same type. Moreover, $\sigma_3(X)(b_1,\dots,b_n)$, conveniently β-reduced, is a subterm of some $\sigma_2(Y)$, for some $Y\in\mathrm{Dom}(\sigma_2)$. In the second case, if $X\in\mathrm{Dom}(\sigma_2)$, we also have this property. Therefore, since σ_2 is $\langle\mathsf{L},\nabla_2\rangle$-compatible, we have that $\sigma_3(X)(b_1,\dots,b_n)$, and hence $\sigma_3(X)(a_1,\dots,a_n)$ are $\langle\mathsf{L},\nabla_2\rangle$-compatible. Therefore, $[\![\sigma_3]\!]^{\text{-1}\,\nabla_1}_{\mathsf{L},\nabla_2} = \bigcup_{X\in\mathrm{Dom}(\sigma_3)}[\mathsf{X}\mapsto[\![\sigma_3(X)(a_1,\dots,a_n)]\!]^{\text{-1}}_{\mathsf{L},\nabla_2}]$ exists, and σ_3 is $\langle\mathsf{L},\nabla_1\to\nabla_2\rangle$-compatible.

(3) Let be $\mathsf{a}\#\mathsf{X} \in \nabla_1$. The free variable names of $\sigma_3(X)$ and L are disjoint. Therefore, $a \notin \mathrm{FV}(\sigma_3(X)(a_1,\dots,a_n))$, where as before $\langle \mathsf{a}_1,\dots,\mathsf{a}_\mathsf{n}\rangle = \langle \mathsf{a} \in \mathsf{L} \mid \mathsf{a}\#\mathsf{X} \notin \nabla_1\rangle$. By Lemma 7.4, since $\sigma_3(X)(a_1,\dots,a_n)$ is $\langle \mathsf{L},\nabla_2\rangle$-compatible, we have $a \notin \mathrm{FV}\left(\left[\!\left[[\![\sigma_3(X)(a_1,\dots,a_n)]\!]^{-1}_{\mathsf{L},\nabla_2} \right]\!\right]_{\mathsf{L},\nabla_2}\right)$. By Lemma 5.9, $\nabla_2 \vdash \mathsf{a}\#\,[\![\sigma_3(X)(a_1,\dots,a_n)]\!]^{-1}_{\mathsf{L},\nabla_2}$. By Definition 8.1, $\nabla_2 \vdash \mathsf{a}\#[\![\sigma_3]\!]^{-1\,\nabla_1}_{\mathsf{L},\nabla_2}(\mathsf{X})$. Therefore, we have $\nabla_2 \vdash [\![\sigma_3]\!]^{-1\,\nabla_1}_{\mathsf{L},\nabla_2}(\nabla_1)$. □

The following lemma ensures that the translation and reverse translation of substitutions is monotone w.r.t. the more generality relation.

LEMMA 8.5. *For every nominal unification problem* P*, any list* L *containing all atoms of* P *without repetitions, and pair of unifiers* $\langle\nabla_1,\sigma_1\rangle$ *and* $\langle\nabla_2,\sigma_2\rangle$*, satisfying* $\mathrm{Vars}(\mathsf{P}) \subseteq \mathrm{Dom}(\sigma_1) \subseteq \mathrm{Dom}(\sigma_2)$*, we have* $\langle\nabla_1,\sigma_1\rangle$ *is more general than* $\langle\nabla_2,\sigma_2\rangle$*, if, and only if,* $[\![\sigma_1]\!]_{\mathsf{L},\nabla_1}$ *is more general than* $[\![\sigma_2]\!]_{\mathsf{L},\nabla_2}$*.*

PROOF. The reduction of nominal unification to equational nominal unification preserves the set of solutions of a problem (see Lemma 4.4), hence we assume that P is a equational nominal unification problem.

$\Rightarrow$) By Theorem 5.12, both $[\![\sigma_1]\!]_{\mathsf{L},\nabla_1}$ and $[\![\sigma_2]\!]_{\mathsf{L},\nabla_2}$ are solutions of $[\![\mathsf{P}]\!]_\mathsf{L}$. If $\langle\nabla_1,\sigma_1\rangle$ is more general than $\langle\nabla_2,\sigma_2\rangle$, then there exists a nominal substitution σ' such that $\nabla_2 \vdash \sigma'(\nabla_1)$ and $\nabla_2 \vdash \sigma' \circ \sigma_1|_{\mathrm{Dom}(\sigma_1)\cup\mathrm{Dom}(\sigma_2)} \approx \sigma_2$.

For all $\mathsf{X} \in \mathrm{Dom}(\sigma_2)$, we have

$$\nabla_2 \vdash \sigma'(\sigma_1(X)) \approx \sigma_2(X)$$

By Lemma 5.9,

$$[\![\sigma'(\sigma_1(\mathsf{X}))]\!]_{\mathsf{L},\nabla_2} =_\alpha [\![\sigma_2(\mathsf{X})]\!]_{\mathsf{L},\nabla_2}$$

By Lemma 8.2,

$$[\![\sigma']\!]^{\nabla_1}_{\mathsf{L},\nabla_2}([\![\sigma_1(\mathsf{X})]\!]_{\mathsf{L},\nabla_1}) =_\alpha [\![\sigma_2(\mathsf{X})]\!]_{\mathsf{L},\nabla_2}$$

By Lemma 5.10,

$$[\![\sigma']\!]^{\nabla_1}_{\mathsf{L},\nabla_2}([\![\sigma_1]\!]_{\mathsf{L},\nabla_1}([\![\mathsf{X}]\!]_{\mathsf{L},\emptyset})) =_\alpha [\![\sigma_2]\!]_{\mathsf{L},\nabla_2}([\![\mathsf{X}]\!]_{\mathsf{L},\emptyset})$$

Since $[\![\mathsf{X}]\!]_{\mathsf{L},\emptyset} = X(a_1,\dots,a_n)$, where $\mathsf{L} = \langle \mathsf{a}_1,\dots,\mathsf{a}_\mathsf{n}\rangle$, and a_i are distinct variables, we have

$$[\![\sigma_2]\!]_{\mathsf{L},\nabla_2}(X) = [\![\sigma']\!]^{\nabla_1}_{\mathsf{L},\nabla_2} \circ [\![\sigma_1]\!]_{\mathsf{L},\nabla_1}(X), \qquad \text{for all } X \in \mathrm{Dom}([\![\sigma_2]\!]_{\mathsf{L},\nabla_2})$$

Therefore, $[\![\sigma_1]\!]_{\mathsf{L},\nabla_1}$ is more general than $[\![\sigma_2]\!]_{\mathsf{L},\nabla_2}$.

$\Leftarrow$) There exists a λ-substitution σ' such that

$$[\![\sigma_2]\!]_{\mathsf{L},\nabla_2} = \sigma' \circ [\![\sigma_1]\!]_{\mathsf{L},\nabla_1}|_{\mathrm{Dom}(\sigma_1)\cup\mathrm{Dom}(\sigma_2)}$$

By Lemma 8.4, σ' is $\langle \mathsf{L},\nabla_1 \to \nabla_2\rangle$-compatible. Hence, it exists the nominal substitution $\sigma'' = [\![\sigma']\!]^{-1\,\nabla_1}_{\mathsf{L},\nabla_2}$. For any $\mathsf{X} \in \mathrm{Dom}(\sigma_2)$, by Lemmas 8.2 and 8.3, we have

$$\begin{aligned}
\left[\!\left[\sigma''(\sigma_1(\mathsf{X}))\right]\!\right]_{\mathsf{L},\nabla_2} &= [\![\sigma'']\!]^{\nabla_1}_{\mathsf{L},\nabla_2}\big([\![\sigma_1]\!]^{\emptyset}_{\mathsf{L},\nabla_1}([\![\mathsf{X}]\!]_{\mathsf{L},\emptyset})\big)\\
&= \sigma'\big([\![\sigma_1]\!]^{\emptyset}_{\mathsf{L},\nabla_1}([\![\mathsf{X}]\!]_{\mathsf{L},\emptyset})\big)\\
&= [\![\sigma_2]\!]^{\emptyset}_{\mathsf{L},\nabla_2}([\![\mathsf{X}]\!]_{\mathsf{L},\emptyset})\\
&= [\![\sigma_2(\mathsf{X})]\!]_{\mathsf{L},\nabla_2}
\end{aligned}$$

By Lemma 5.9, we have $\nabla_2 \vdash \sigma''\big(\sigma_1(\mathsf{X})\big) \approx \sigma_2(\mathsf{X})$. Therefore, $\nabla_2 \vdash \sigma'' \circ \sigma_1|_{\mathrm{Dom}(\sigma_1)\cup\mathrm{Dom}(\sigma_2)} \approx \sigma_2$. By Lemma 8.4, we also have $\nabla_2 \vdash \sigma''(\nabla_1)$. From both facts, we conclude that σ_1 is more general than σ_2. □

COROLLARY 8.6. *Most general nominal unifiers of nominal unification problems are unique.*

PROOF. It is a direct consequence of uniqueness of most general higher-order pattern unifiers and Lemma 8.5. □

Finally we can conclude that the translations preserve most generality.

THEOREM 8.7. *For any nominal problem* P*, any list* L *containing all atoms of* P *without repetitions, and nominal solution* $\langle\nabla,\sigma\rangle$*, satisfying* $\mathrm{Vars}(\mathsf{P}) \subseteq \mathrm{Dom}(\sigma)$*, we have* $\langle\nabla,\sigma\rangle$ *is a most general unifier of* P *if, and only if,* $[\![\sigma]\!]_{\mathsf{L},\nabla}$ *is a most general unifier of* $[\![\mathsf{P}]\!]_{\mathsf{L}}$.

PROOF. Like in Lemma 8.5, we can assume that P is an equational nominal unification problem.

$\Rightarrow$) Suppose that $\langle\nabla,\sigma\rangle$ is a most general nominal unifier of P, but $[\![\sigma]\!]_{\mathsf{L},\nabla}$ is not a most general pattern unifier of $[\![\mathsf{P}]\!]$. By Theorem 5.12, $[\![\sigma]\!]_{\mathsf{L},\nabla}$ is a solution of $[\![\mathsf{P}]\!]_{\mathsf{L}}$. Since most general higher-order pattern unifiers are unique, and by Lemma 7.6, there exists a most general pattern unifier σ' of $[\![\mathsf{P}]\!]_{\mathsf{L}}$ *strictly* more general than $[\![\sigma]\!]_{\mathsf{L},\nabla}$ and such that $[\![\sigma']\!]^{-1}_{\mathsf{L},\nabla}$ exists. By Lemma 7.4, $\Big[\!\!\Big[[\![\sigma']\!]^{-1}_{\mathsf{L},\nabla}\Big]\!\!\Big]_{\mathsf{L},\nabla} = \sigma'$. Since we assume that $\langle\nabla,\sigma\rangle$ is most general and nominal most general unifiers are also unique, we have that $\langle\nabla,\sigma\rangle$ is more general than $[\![\sigma']\!]^{-1}_{\mathsf{L},\nabla}$. Hence, by Lemma 8.5, $[\![\sigma]\!]_{\mathsf{L},\nabla}$ is more general than $\Big[\!\!\Big[[\![\sigma']\!]^{-1}_{\mathsf{L},\nabla}\Big]\!\!\Big]_{\mathsf{L},\nabla} = \sigma'$, which contradicts that σ' is strictly more general than $[\![\sigma]\!]_{\mathsf{L},\nabla}$.

$\Leftarrow$) Suppose that $[\![\sigma]\!]_{\mathsf{L},\nabla}$ is most general, and $\langle\nabla,\sigma\rangle$ is not. Then, there exists a most general unifier $\langle\nabla',\sigma'\rangle$ such that $\langle\nabla,\sigma\rangle$ is not more general than $\langle\nabla',\sigma'\rangle$. On the other hand, since $[\![\sigma]\!]_{\mathsf{L},\nabla}$ is most general, it is more general than $[\![\sigma']\!]_{\mathsf{L},\nabla'}$. Hence, by Lemma 8.5, $\langle\nabla,\sigma\rangle$ is more general than $\langle\nabla',\sigma'\rangle$. This contradicts the initial assumption. Therefore, if $[\![\sigma]\!]_{\mathsf{L},\nabla}$ is most general, then $\langle\nabla,\sigma\rangle$ must be most general. □

9. CONCLUSIONS

The paper describes a precise quadratic reduction from Nominal Unification to Higher-Order Pattern Unification. The main idea of this reduction is to translate nominal suspensions $\pi{\cdot}\mathsf{X}$ as λ-terms of the form $X(\pi(a_1),\dots,\pi(a_n))$, where $\mathsf{a_i}$ are the atoms that X may capture. These atoms are translated as bound variables, by adding λ-bindings in front of translated terms. Therefore, the translation results on higher-order patterns. This reduction helps to better understand the semantics of the nominal binding and permutations in comparison with λ-binding and α-conversion. We also describe a new algorithm for Higher-Order Pattern Unification where no new names of bound variables are introduced. A similar property holds for the Urban et al. [2003] algorithm, where no new atom names are introduced.

This paper is closely related to Permissive Nominal Unification [Dowek et al. 2009; 2010]. In these works, there is also a reduction of permissive unification (basically nominal unification) to higher-order pattern unification. In fact, both reductions share the same basic ideas. In permissive terms, atoms are divided into two infinite sets $\mathbb{A}^< \cup \mathbb{A}^>$, and variables X^S are tagged with a *permission set* S of the form $(\mathbb{A}^< \setminus \mathsf{A}) \cup \mathsf{B}$

where $\mathsf{A} \subseteq \mathbb{A}^<$ and $\mathsf{B} \subseteq \mathbb{A}^>$ are both finite. Roughly, S is the (infinite) set of capturable atoms in instantiations of X^S. Obviously, Dowek et al. [2010] do not pass all these atoms as arguments when they translate these suspensions into λ-terms. Their translation function (like ours) is parametric in a finite list of atoms D and translate $[\![\pi{\cdot}\mathsf{X}^\mathsf{S}]\!]^\mathsf{D} = X^S(\pi(a_1), \dots \pi(a_n))$ where $\{\mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n}\} = \mathsf{D} \cap \mathsf{S}$ (see Definition 8.3 in Dowek et al. [2010]). Therefore, their D plays a similar role to our L, and their S to our ∇. In our case, in Theorem 5.12 we pass as L all atoms occurring in the problem. Recall that the fact that nominal solutions could be expressed using only atoms of the problems is crucial in our case. In the corresponding Theorem 9.16 in [Dowek et al. 2010], they pass a more refined list of atoms, called *capturable atoms* of the problem (see Definition 8.7 in [Dowek et al. 2010]). In fact, they go further and prove that this list of atoms is minimal (see Theorem 8.14 in [Dowek et al. 2010]). Therefore, their reduction is optimal, and if the set of capturable atoms remains bounded, then the reduction is linear.

Qian [1996] proved that Higher-Order Pattern Unification can be decided in linear time and space. Some difficulties to verify the proof in detail has lead some researchers to doubt about its correctness [Urban 2008]. If this result turns to be correct, we would get that Nominal Unification can be decided in quadratic time using our reduction. Recently, Levy and Villaret [2010], and Calvès [2010] have found a quadratic algorithm for Nominal Unification not based on higher-order patterns. As further work it would be interesting to look for a linear or quasi-linear algorithm for Nominal Unification and Higher-Order Pattern Unification.

Acknowledgements

We are grateful to Murdoch J. Gabbay, Christian Urban and the anonymous referees of this paper for their helpful comments.

REFERENCES

CALVÈS, C. 2010. Complexity and implementation of nominal algorithms. Ph.D. thesis, King's College London.

CALVÈS, C. AND FERNÁNDEZ, M. 2007. Implementing nominal unification. *ENTCS 176,* 1, 25–37.

CALVÈS, C. AND FERNÁNDEZ, M. 2008. A polynomial nominal unification algorithm. *Theoretical Computer Science 403,* 2-3, 285–306.

CHENEY, J. 2005a. Equivariant unification. In *Proc. of the 16th Int. Conf. on Rewritting Techniques and Applications, RTA'05*. Lecture Notes in Computer Science Series, vol. 3467. Springer, 74–89.

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

CHENEY, J. AND URBAN, C. 2004. α-prolog: A logic programming language with names, binding and α-equivalence. In *Proc. of the 20th Int. Conf. on Logic Programming,ICLP'04*. LNCS Series, vol. 3132. 269–283.

CLOUSTON, R. A. AND PITTS, A. M. 2007. Nominal equational logic. *ENTCS 1496*, 223–257.

DOWEK, G. 2001. Higher-order unification and matching. In *Handbook of automated reasoning*. 1009–1062.

DOWEK, G., GABBAY, M., AND MULLIGAN, D. 2010. Permissive nominal terms and their unification. *Logic Journal of the IGPL*.

DOWEK, G., GABBAY, M. J., AND MULLIGAN, D. 2009. Permissive nominal terms and their unification. In *Proc. of the 24th Italian Conf. on Computational Logic, CILC'09*.

FERNÁNDEZ, M. AND GABBAY, M. 2005. Nominal rewriting with name generation: abstraction vs. locality. In *Proc. of the 7th Int. Conf. on Principles and Practice of Declarative Programming, PPDP'05*. 47–58.

FERNÁNDEZ, M. AND GABBAY, M. 2007. Nominal rewriting. *Information and Computation 205,* 6, 917–965.

GABBAY, M. AND MATHIJSSEN, A. 2006. Nominal algebra. In *Proc. of the $18^t h$ Nordic Workshop on Programming Theory, NWPT'06*.

GABBAY, M. AND MATHIJSSEN, A. 2007. A formal calculus for informal equality with binding. In *Logic, Language, Information and Computation*. LNCS Series, vol. 4576. Springer, 162–176.

GABBAY, M. AND MATHIJSSEN, A. 2009. Nominal (universal) algebra: equational logic with names and binding. *Journal of Logic and Computation 19,* 6, 1455–1508.

GABBAY, M. AND PITTS, A. 2001. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing 13,* 3–5, 341–363.

GABBAY, M. AND PITTS, A. M. 1999. A new approach to abstract syntax involving binders. In *Proc. of the 14th Symp. on Logic in Computer Science, LICS'99*. 214–224.

GOLDFARB, W. D. 1981. The undecidability of the second-order unification problem. *Theoretical Computer Science 13*, 225–230.

LEVY, J. 1998. Decidable and undecidable second-order unification problems. In *Proc. of the 9th Int. Conf. on Rewriting Techniques and Applications, RTA'98*. LNCS Series, vol. 1379. 47–60.

LEVY, J. AND VEANES, M. 2000. On the undecidability of second-order unification. *Information and Computation 159*, 125–150.

LEVY, J. AND VILLARET, M. 2008. Nominal unification from a higher-order perspective. In *Proc. of the 19th Int. Conf on Rewriting Techniques and Applications, RTA'08*. LNCS Series, vol. 5117. Springer, 246–260.

LEVY, J. AND VILLARET, M. 2009. Simplifying the signature in second-order unification. *Appl. Algebra Eng. Commun. Comput. 20,* 5-6, 427–445.

LEVY, J. AND VILLARET, M. 2010. An efficient nominal unification algorithm. In *Proc. of the 21th Int. Conf on Rewriting Techniques and Applications, RTA'10*. LNCS. Springer.

LUCCHESI, C. L. 1972. The undecidability of the unification problem for third-order languages. Tech. Rep. CSRR 2059, Dept. of Applied Analysis and Computer Science, Univ. of Waterloo.

MENDELZON, A., RÍOS, A., AND ZILIANI, B. 2010. Swapping: a natural bridge between named and indexed explicit substitution calculi. In *Proc. of the 5th Int. Workshop on Higher-Order Rewriting, HOR'10*. 41–46.

MILLER, D. 1991. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. of Logic and Computation 1,* 4, 497–536.

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

PATERSON, M. AND WEGMAN, M. N. 1978. Linear unification. *J. Comput. Syst. Sci. 16,* 2, 158–167.

PITTS, A. M. 2001. 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*. LNCS Series, vol. 2215. 219–242.

PITTS, A. M. 2003. Nominal logic, a first order theory of names and binding. *Information and Computation 186*, 165–193.

QIAN, Z. 1996. Unification of higher-order patterns in linear time and space. *J. of Logic and Computation 6,* 3, 315–341.

URBAN, C. 2008. Personal communication.

URBAN, C. AND CHENEY, J. 2005. Avoiding equivariance in alpha-prolog. In *Proc. of the Int. Conf. on Typed Lambda Calculus and Applications, TLCA'05*. LNCS Series, vol. 3461. 401–416.

URBAN, C., PITTS, A. M., AND GABBAY, M. J. 2003. Nominal unification. In *Proc. of the 17th Int. Work. on Computer Science Logic, CSL'03*. LNCS Series, vol. 2803. 513–527.

URBAN, C., PITTS, A. M., AND GABBAY, M. J. 2004. Nominal unification. *Theoretical Computer Science 323*, 473–497.

Received May 2010; revised January 2011; accepted January 2011