Nominal Anti-Unification *

Alexander Baumgartner[1], Temur Kutsia[1], Jordi Levy[2], and Mateu Villaret[3]

1 **Research Institute for Symbolic Computation (RISC)**
Johannes Kepler University, Linz, Austria
{abaumgar,kutsia}@risc.jku.at
2 **Artificial Intelligence Research Institute (IIIA)**
Spanish Council for Scientific Research (CSIC), Barcelona, Spain
levy@iiia.csic.es
3 **Departament d'Informàtica i Matemàtica Aplicada (IMA)**
Universitat de Girona (UdG), Girona, Spain
villaret@ima.udg.edu

Abstract

We study nominal anti-unification, which is concerned with computing least general generalizations for given terms-in-context. In general, the problem does not have a least general solution, but if the set of atoms permitted in generalizations is finite, then there exists a least general generalization which is unique modulo variable renaming and α-equivalence. We present an algorithm that computes it. The algorithm relies on a subalgorithm that constructively decides equivariance between two terms-in-context. We prove soundness and completeness properties of both algorithms and analyze their complexity. Nominal anti-unification can be applied to problems were generalization of first-order terms is needed (inductive learning, clone detection, etc.), but bindings are involved.

1998 ACM Subject Classification F.4.1 [Theory of Computation]: Mathematical Logic and Formal Languages—Mathematical Logic, F.2.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity—Nonnumerical Algorithms and Problems.

Keywords and phrases Nominal Anti-Unification, Term-in-context, Equivariance.

Digital Object Identifier 10.4230/LIPIcs.RTA.2015.x

1 Introduction

Binders are very common in computer science, logic, mathematics, linguistics. Functional abstraction λ, universal quantifier $\forall$, limit lim, integral $\int$ are some well-known examples of binders. To formally represent and study systems with binding, Pitts and Gabbay [13–15] introduced nominal techniques, based on the idea to give explicit names to bound entities. It makes a syntactic distinction between *atoms,* which can be bound, and *variables,* which can be substituted. This approach led to the development of the theory of nominal sets, nominal logic, nominal algebra, nominal rewriting, nominal logic programming, etc.

Equation solving between nominal terms (maybe together with freshness constraints) has been investigated by several authors, who designed and analyzed algorithms for nominal unification [5, 6, 18–20, 30], nominal matching [7], equivariant unification [9], and permissive

* This research has been partially supported by the Spanish project HeLo (TIN2012-33042), by the Austrian Science Fund (FWF) with the project SToUT (P 24087-N18), and by the strategic program "Innovatives OÖ 2010plus" by the Upper Austrian Government.

26th International Conference on Rewriting Techniques and Applications (RTA'15).
Editor: Maribel Fernández; pp. 1–17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

nominal unification [10, 11]. However, in contrast to unification, its dual problem, anti-unification, has not been studied for nominal terms previously.

The anti-unification problem for two terms t_1 and t_2 is concerned with finding a term t that is more general than the original ones, i.e., t_1 and t_2 should be substitutive instances of t. The interesting generalizations are the least general ones, which retain the common structure of t_1 and t_2 as much as possible. Plotkin [23] and Reynolds [25] initiated research on anti-unification in the 1970s, developing generalization algorithms for first-order terms. Since then, anti-unification has been studied in various theories, including some of those with binding constructs: calculus of constructions [22], $M\lambda$ [12], second-order lambda calculus with type variables [21], simply-typed lambda calculus where generalizations are higher-order patterns [3], just to name a few.

The problem we address in this paper is to compute generalizations for nominal terms. More precisely, we consider this problem for nominal *terms-in-context*, which are pairs of a freshness context and a nominal term, aiming at computing their least general generalizations (lgg). However, it turned out that without a restriction, there is no lgg for terms-in-context, in general. Even more, a *minimal* complete set of generalizations does not exist. This is in sharp contrast with the related problem of anti-unification for higher-order patterns, which always have a single lgg [3]. The reason is one can make terms-in-context less and less general by adding freshness constraints for the available (infinitely many) atoms, see Example 2.7. Therefore, we restrict the set of atoms which are permitted in generalizations to be fixed and finite. In this case, there exists a single lgg (modulo α-equivalence and variable renaming) for terms-in-context and we design an algorithm to compute it in $O(n^5)$ time.

There is a close relation between nominal and higher-order pattern unification: One can be translated into the other by the solution-preserving translation defined in [8, 18, 20] or the translation defined for permisive terms in [10, 11]. We show that for anti-unification, this method, in general, is not applicable. Even if one finds conditions under which such a translation-based approach to anti-unification works, due to complexity reasons it is still better to use the direct nominal anti-unification algorithm developed in this paper.

Computation of nominal lgg's requires to solve the equivariance problem: Given two terms s_1 and s_2, find a permutation of atoms which, when applied to s_1, makes it α-equivalent to s_2 (under the given freshness context). This is necessary to guarantee that the computed generalization is *least* general. For instance, if the given terms are $s_1 = f(a, b)$ and $s_2 = f(b, a)$, where a, b are atoms, the freshness context is empty, and the atoms permitted in the generalization are a, b, and c, then the term-in-context $\langle\{c\#X, c\#Y\}, f(X, Y)\rangle$ generalizes $\langle\emptyset, s_1\rangle$ and $\langle\emptyset, s_2\rangle$, but it is not their lgg. To compute the latter, we need to reflect the fact that generalizations of the atoms are related to each other: One can be obtained from the other by swapping a and b. This leads to an lgg $\langle\{c\#X\}, f(X, (a\,b)\cdot X)\rangle$. To compute the permutation $(a\,b)$, an equivariance problem should be solved. Equivariance is already present in α-Prolog [28] and Isabelle [27]. We develop a rule-based algorithm for equivariance problems, which computes in quadratic time the justifying permutation if the input terms are equivariant, and fails otherwise.

Both anti-unification and equivariance algorithms are implemented in the anti-unification algorithm library [2] and can be accessed from `http://www.risc.jku.at/projects/stout/software/`.

Various variants of anti-unification, such as first-order, higher-order, or equational anti-unification have been used in inductive logic programming, logical and relational learning [24], reasoning by analogy [16], program synthesis [26], program verification [21], etc. Nominal anti-unification can, hopefully, contribute in solving similar problems in nominal

setting or in first-order settings where bindings play an important role.

In this paper, we mainly follow the notation from [20]. Long proofs can be found in the technical report [4].

2 Nominal Terms

In *nominal signatures* we have *sorts of atoms* (typically ν) and *sorts of data* (typically δ) as disjoint sets. *Atoms* (typically $a, b, \ldots$) have one of the sorts of atoms. *Variables* (typically $X, Y, \ldots$) have a sort of atom or a sort of data, i.e. of the form $\nu \mid \delta$. In nominal terms, variables can be instantiated and atoms can be bound. Nominal function symbols (typically $f, g, \ldots$) have an arity of the form $\tau_1 \times \cdots \times \tau_n \to \delta$, 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 $\langle\nu\rangle\tau$.

A *swapping* $(a\,b)$ is a pair of atoms of the same sort. A *permutation* is a (possibly empty) sequence of swappings. We use upright Greek letters (e.g., π, ρ) to denote permutations. *Nominal terms* (typically $t, s, u, r, q, \ldots$) are given by the grammar:

$$t ::= f(t_1, \ldots, t_n) \mid a \mid a.t \mid \pi{\cdot}X$$

where f is an n-ary function symbol, a is an atom, π is a permutation, and X is a variable. They are called respectively *application, atom, abstraction,* and *suspension.* The sorts of application and atomic terms are defined as usual, the sort of $a.t$ is $\langle\nu\rangle\tau$ where ν is the sort of a and τ is the sort of t, and the sort of $\pi{\cdot}X$ is the one of X.

The *inverse* of a permutation $\pi = (a_1\,b_1)\ldots(a_n\,b_n)$ is the permutation $(a_n\,b_n)\ldots(a_1\,b_1)$, denoted by π^{-1}. The empty permutation is denoted by Id. The *effect of a swapping* over an atom is defined by $(a\,b) \bullet a = b$, $(a\,b) \bullet b = a$ and $(a\,b) \bullet c = c$, when $c \notin \{a, b\}$. It is extended to the rest of terms: $(a\,b) \bullet f(t_1, \ldots, t_n) = f((a\,b) \bullet t_1, \ldots, (a\,b) \bullet t_n)$, $(a\,b) \bullet (c.t) = ((a\,b) \bullet c)\,.\,((a\,b) \bullet t)$, and $(a\,b) \bullet \pi{\cdot}X = (a\,b)\pi{\cdot}X$, where $(a\,b)\pi$ is the permutation obtained by concatenating $(a\,b)$ and π. The *effect of a permutation* is defined by $(a_1\,b_1)\ldots(a_n\,b_n) \bullet t = (a_1\,b_1) \bullet ((a_2\,b_2)\ldots(a_n\,b_n) \bullet t)$. The effect of the empty permutation is $Id \bullet t = t$. We extend it to suspensions and write X as the shortcut of $Id{\cdot}X$.

The set of variables of a term t is denoted by $\mathrm{Vars}(t)$. A term t is called *ground* if $\mathrm{Vars}(t) = \emptyset$. The set of *atoms* of a term t or a permutation π is the set of all atoms which appear in it and is denoted by $\mathrm{Atoms}(t)$, $\mathrm{Atoms}(\pi)$ respectively. For instance, $\mathrm{Atoms}(f(a.g(a), (b\,c){\cdot}X, d) = \{a, b, c, d\}$. We write $\mathrm{Atoms}(t_1, \ldots, t_n)$ for the set $\mathrm{Atoms}(t_1) \cup \cdots \cup \mathrm{Atoms}(t_n)$.

Positions in terms are defined with respect to their tree representation in the usual way, as strings of integers. However, suspensions are put in a single leaf node. For instance, the tree form of the term $f(a.b.g((a\,b){\cdot}X, a), h(c))$, and the corresponding positions are shown in Fig. 1. The symbol f stands in the position ϵ (the empty sequence). The suspension is put in one node of the tree, at the position 1.1.1.1. The abstraction operator and the corresponding bound atom together occupy one node as well. For any term t, $t|_{\mathsf{p}}$ denotes the *subterm of t at position* p. For instance, $f(a.b.g((a\,b){\cdot}X, a), h(c))|_{1.1} = b.g((a\,b){\cdot}X, a)$.

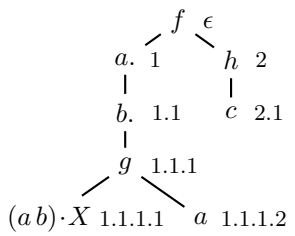

Figure 1 The tree form and positions of the term $f(a.b.g((a\,b){\cdot}X, a), h(c))$.

Every permutation π naturally defines a bijective function from the set of atoms to the sets of atoms, that we will also represent as π. Suspensions are uses of variables with a permutation of atoms waiting to be applied once the variable is instantiated. 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. We denote by $\mathrm{FA}(t)$ the set of all atoms which occur freely in t: $\mathrm{FA}(f(t_1,\ldots,t_n)) = \bigcup_{i=1}^{n} \mathrm{FA}(t_i)$, $\mathrm{FA}(a) = \{a\}$, $\mathrm{FA}(a.t) = \mathrm{FA}(t) \setminus \{a\}$, and $\mathrm{FA}(\pi\cdot X) = \mathrm{Atoms}(\pi)$. $\mathrm{FA}^{\text{-s}}(t)$ is the set of all atoms which occur freely in t ignoring suspensions: $\mathrm{FA}^{\text{-s}}(f(t_1,\ldots,t_n)) = \bigcup_{i=1}^{n} \mathrm{FA}^{\text{-s}}(t_i)$, $\mathrm{FA}^{\text{-s}}(a) = \{a\}$, $\mathrm{FA}^{\text{-s}}(a.t) = \mathrm{FA}^{\text{-s}}(t) \setminus \{a\}$, and $\mathrm{FA}^{\text{-s}}(\pi\cdot X) = \emptyset$.

The head of a term t, denoted $\mathrm{Head}(t)$, is defined as: $\mathrm{Head}(f(t_1,\ldots,t_n)) = f$, $\mathrm{Head}(a) = a$, $\mathrm{Head}(a.t) = .$, and $\mathrm{Head}(\pi\cdot X) = X$.

Substitutions are defined in the standard way, as a mapping from variables to terms of the same sort. We use Greek letters $\sigma, \vartheta, \varphi$ to denote substitutions. The identity substitution is denoted by ε. Furthermore, we use the postfix notation for substitution applications, i.e. $t\sigma$ denotes the application of a substitution σ to a term t, and similarly, the composition of two substitutions σ and ϑ is written as $\sigma\vartheta$. Composition of two substitutions is performed as usual. Their application allows atom capture, for instance, $a.X\{X \mapsto a\} = a.a$, and forces the permutation effect: $\pi\cdot X\{X \mapsto t\} = \pi \bullet t$, for instance, $(a\,b)\cdot X\{X \mapsto f(a,(a\,b)\cdot Y)\} = f(b,(a\,b)(a\,b)\cdot Y)$. The notions of substitution *domain* and *range* are also standard and are denoted, respectively, by Dom and Ran.

A *freshness constraint* is a pair of the form $a\#X$ stating that the instantiation of X cannot contain free occurrences of a. A *freshness context* is a finite set of freshness constraints. We will use ∇ and Γ to denote freshness contexts. $\mathrm{Vars}(\nabla)$ and $\mathrm{Atoms}(\nabla)$ denote respectively the set of variables and atoms of ∇.

We say that a substitution σ *respects* a freshness context ∇, if for all X, $\mathrm{FA}^{\text{-s}}(X\sigma) \cap \{a \mid a\#X \in \nabla\} = \emptyset$.

The predicate $\approx$, which stands for α-equivalence between terms, and the freshness predicate $\#$ were defined in [29,30] by the following theory:

$$\frac{}{\nabla \vdash a \approx a} \qquad \frac{\nabla \vdash t \approx t'}{\nabla \vdash a.t \approx a.t'} \qquad \frac{a \neq a' \quad \nabla \vdash t \approx (a\,a') \bullet t' \quad \nabla \vdash a\#t'}{\nabla \vdash a.t \approx a'.t'}$$

$$\frac{a\#X \in \nabla \text{ for all } a \text{ such that } \pi \bullet a \neq \pi' \bullet a}{\nabla \vdash \pi\cdot X \approx \pi'\cdot X} \qquad \frac{\nabla \vdash t_1 \approx t_1' \quad \cdots \quad \nabla \vdash t_n \approx t_n'}{\nabla \vdash f(t_1,\ldots t_n) \approx f(t_1',\ldots,t_n')}$$

where the freshness predicate $\#$ is defined by

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

$$\frac{(\pi^{-1} \bullet a\#X) \in \nabla}{\nabla \vdash a\#\pi\cdot X}(\#\text{-susp.}) \qquad \frac{\nabla \vdash a\#t_1 \quad \cdots \quad \nabla \vdash a\#t_n}{\nabla \vdash a\#f(t_1,\ldots t_n)}(\#\text{-application})$$

Their intended meanings are:

1. $\nabla \vdash a\#t$ holds, if for every substitution σ such that $t\sigma$ is a ground term and σ respects the freshness context ∇, we have a is not free in $t\sigma$;
2. $\nabla \vdash t \approx u$ holds, if for every substitution σ such that $t\sigma$ and $u\sigma$ are ground terms and σ respects the freshness context ∇, $t\sigma$ and $u\sigma$ are α-equivalent.

Based on the definition of the freshness predicate, we can design an algorithm, which we call FC, which solves the following problem: Given a set of freshness formulas $\{a_1\#t_1,\ldots, a_n\#t_n\}$, compute a *minimal* (with respect to $\subseteq$) freshness context ∇ such that $\nabla \vdash a_1\#t_1,\ldots,\nabla \vdash a_n\#t_n$. Such a ∇ may or may not exist, and the algorithm should detect it. The algorithm can be found in the technical report [4]. It is simply a bottom-up application of the rules of the freshness predicate, starting from each of the $\nabla \vdash a_1\#t_1,\ldots,\nabla \vdash a_n\#t_n$.

It succeeds if each branch of such a derivation tree is either closed (i.e., ends with the application of the #-atom or the #-abst-1 rule), or ends with an application of the #-susp. rule, producing a membership atom of the form $a\#X \in \nabla$ for some a and X. In this case we say that the desired ∇ is the set of all such $a\#X$ freshness atoms, and write $\mathsf{FC}(\{a_1\#t_1, \ldots, a_n\#t_n\}) = \nabla$. (Hence, ∇ is empty when all branches are closed.) It fails if at least one branch of the derivation tree produces $\nabla \vdash a\#a$ for some a, i.e., no rule applies to it. In this case we write $\mathsf{FC}(\{a_1\#t_1, \ldots, a_n\#t_n\}) = \bot$. The following theorem is easy to verify:

▶ **Theorem 2.1.** *Let F be a set of freshness formulas and ∇ be a freshness context. Then $\mathsf{FC}(F) \subseteq \nabla$ iff $\nabla \vdash a\#t$ for all $a\#t \in F$.*

▶ **Corollary 2.2.** *$\mathsf{FC}(F) = \bot$ iff there is no freshness context that would justify all formulas in F.*

Given a freshness context ∇ and a substitution σ, we define $\nabla\sigma = \mathsf{FC}(\{a\#X\sigma \mid a\#X \in \nabla\})$. The following lemma is straightforward:

▶ **Lemma 2.3.** *σ respects ∇ iff $\nabla\sigma \neq \bot$.*

When $\nabla\sigma \neq \bot$, we call $\nabla\sigma$ the *instance* of ∇ under σ.

It is not hard to see that (a) if σ respects ∇, then σ respects any $\nabla' \subseteq \nabla$, and (b) if σ respects ∇ and ϑ respects $\nabla\sigma$, then $\sigma\vartheta$ respects ∇ and $(\nabla\sigma)\vartheta = \nabla(\sigma\vartheta)$.

▶ **Definition 2.4.** A *term-in-context* is a pair $\langle \nabla, t\rangle$ of a freshness context and a term. A term-in-context $\langle \nabla_1, t_1\rangle$ is *more general* than a term-in-context $\langle \nabla_2, t_2\rangle$, written $\langle \nabla_1, t_1\rangle \preceq \langle \nabla_2, t_2\rangle$, if there exists a substitution σ, which respects ∇_1, such that $\nabla_1\sigma \subseteq \nabla_2$ and $\nabla_2 \vdash t_1\sigma \approx t_2$.

We write $\nabla \vdash t_1 \preceq t_2$ if there exists a substitution σ such that $\nabla \vdash t_1\sigma \approx t_2$.

Two terms-in-context p_1 and p_2 are *equivalent* (or *equi-general*), written $p_1 \simeq p_2$, iff $p_1 \preceq p_2$ and $p_2 \preceq p_1$. The strict part of $\preceq$ is denoted by $\prec$, i.e., $p_1 \prec p_2$ iff $p_1 \preceq p_2$ and not $p_2 \preceq p_1$. We also write $\nabla \vdash t_1 \simeq t_2$ iff $\nabla \vdash t_1 \preceq t_2$ and $\nabla \vdash t_2 \preceq t_1$.

▶ **Example 2.5.** We give some examples to demonstrate the relations we have just defined:

- $\langle \{a\#X\}, f(a)\rangle \simeq \langle \emptyset, f(a)\rangle$. We can use $\{X \mapsto b\}$ for the substitution applied to the first pair.
- $\langle \emptyset, f(X)\rangle \preceq \langle \{a\#X\}, f(X)\rangle$ (with $\sigma = \varepsilon$), but not $\langle \{a\#X\}, f(X)\rangle \preceq \langle \emptyset, f(X)\rangle$.
- $\langle \emptyset, f(X)\rangle \preceq \langle \{a\#Y\}, f(Y)\rangle$ with $\sigma = \{X \mapsto Y\}$.
- $\langle \{a\#X\}, f(X)\rangle \not\preceq \langle \emptyset, f(Y)\rangle$, because in order to satisfy $\{a\#X\}\sigma \subseteq \emptyset$, the substitution σ should map X to a term t which contains neither a (freely) nor variables. But then $\emptyset \vdash f(t) \approx f(Y)$ does not hold. Hence, together with the previous example, we get $\langle \emptyset, f(Y)\rangle \prec \langle \{a\#X\}, f(X)\rangle$.
- $\langle \{a\#X\}, f(X)\rangle \not\preceq \langle \{a\#X\}, f(a)\rangle$. Notice that $\sigma = \{X \mapsto a\}$ does not respect $\{a\#X\}$.
- $\langle \{b\#X\}, (a\,b)\cdot X\rangle \preceq \langle \{c\#X\}, (a\,c)\cdot X\rangle$ with the substitution $\sigma = \{X \mapsto (a\,b)(a\,c)\cdot X\}$. Hence, we get $\langle \{b\#X\}, (a\,b)\cdot X\rangle \simeq \langle \{c\#X\}, (a\,c)\cdot X\rangle$, because the $\succeq$ part can be shown with the help of the substitution $\{X \mapsto (a\,c)(a\,b)\cdot X\}$.

▶ **Definition 2.6.** A term-in-context $\langle \Gamma, r\rangle$ is called a *generalization* of two terms-in-context $\langle \nabla_1, t\rangle$ and $\langle \nabla_2, s\rangle$ if $\langle \Gamma, r\rangle \preceq \langle \nabla_1, t\rangle$ and $\langle \Gamma, r\rangle \preceq \langle \nabla_2, s\rangle$. It is the *least general generalization,* (lgg in short) of $\langle \nabla_1, t\rangle$ and $\langle \nabla_2, s\rangle$ if there is no generalization $\langle \Gamma', r'\rangle$ of $\langle \nabla_1, t\rangle$ and $\langle \nabla_2, s\rangle$ which satisfies $\langle \Gamma, r\rangle \prec \langle \Gamma', r'\rangle$.

Note that if we have infinite number of atoms in the language, the relation $\prec$ is not well-founded: $\langle \emptyset, X\rangle \prec \langle \{a\#X\}, X\rangle \prec \langle \{a\#X, b\#X\}, X\rangle \prec \cdots$. As a consequence, two terms-in-context may not have an lgg and not even a minimal complete set of generalizations:[1]

▶ **Example 2.7.** Let $p_1 = \langle \emptyset, a_1\rangle$ and $p_2 = \langle \emptyset, a_2\rangle$ be two terms-in-context. Then in any complete set of generalizations of p_1 and p_2 there is an infinite chain $\langle \emptyset, X\rangle \prec \langle \{a_3\#X\}, X\rangle \prec \langle \{a_3\#X, a_4\#X\}, X\rangle \prec \cdots$, where $\{a_1, a_2, a_3, \ldots\}$ is the set of all atoms of the language. Hence, p_1 and p_2 do not have a minimal complete set of generalizations.

This example is a proof of the theorem, which characterizes the generalization type of nominal anti-unification:[2]

▶ **Theorem 2.8.** *The problem of anti-unification for terms-in-context is of nullary type.*

However, if we restrict the set of atoms which can be used in the generalizations to be fixed and finite, then the anti-unification problem becomes unitary. (We do not prove this property here, it will follow from the Theorems 6.2 and 6.3 in Sect. 6.)

▶ **Definition 2.9.** We say that a term t (resp., a freshness context ∇) is *based* on a set of atoms A iff $\mathrm{Atoms}(t) \subseteq A$ (resp., $\mathrm{Atoms}(\nabla) \subseteq A$). A term-in-context $\langle \nabla, t\rangle$ is based on A if both t and ∇ are based on it. We extend the notion of A-basedness to permutations, calling π A-based if it contains only atoms from A. Such a permutation defines a bijection, in particular, from A to A. If p_1 and p_2 are A-based terms-in-context, then their *A-based generalizations* are terms-in-context which are generalizations of p_1 and p_2 and are based on A. An *A-based lgg* of A-based terms-in-context p_1 and p_2 is a term-in-context p, which is an A-based generalization of p_1 and p_2 and there is no A-based generalization p' of p_1 and p_2 which satisfies $p \prec p'$.

The problem we would like to solve is the following:

Given: Two nominal terms t and s of the same sort, a freshness context ∇, and a *finite* set of atoms A such that t, s, and ∇ are based on A.

Find: A term r and a freshness context Γ, such that the term-in-context $\langle \Gamma, r\rangle$ is an A-based least general generalization of the terms-in-context $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$.

Our anti-unification problem is parametric on the set of atoms we consider as the base, and finiteness of this set is essential to ensure the existence of an lgg.

3 Motivation of Using a Direct Nominal Anti-Unification Algorithm

In [20], relation between nominal unification (NU) and higher-order pattern unification (HOPU) has been studied. In particular, it was shown how to translate NU problems into HOPU problems and how to obtain nominal unifiers back from higher-order pattern unifiers. It is tempting to use the same translation for nominal anti-unification (NAU), using the algorithm from [3] to solve higher-order anti-unification problems over patterns (HOPAU), but it turns out that the generalization computed in this way is not always based on the same set of the atoms as the input:

[1] Minimal complete sets of generalizations are defined in the standard way. For a precise definition, see, e.g., [1, 17].

[2] Generalization types are defined analogously to unification types: unary, finitary, infinitary, and nullary, see [17].

▶ **Example 3.1.** We consider the following problem: Let the set of atoms be $A_1 = \{a, b\}$. The terms to be generalized are $a.b$ and $b.a$, and the freshness context is $\nabla = \emptyset$. According to [20], translation to higher-order patterns gives the anti-unification problem $\lambda a, b, a.\, b \triangleq \lambda a, b, b.\, a$, whose lgg is $\lambda a, b, c.X(a, b)$. However, we can not translate this lgg back to an A_1-based term-in-context, because it contains more bound variables than there are atoms in A_1.

On the other hand, the translation would work for the set of atoms $A_2 = \{a, b, c\}$: Back-translating $\lambda a, b, c.X(a, b)$ gives the A_2-based lgg $\langle \{c\#X\}, c.X\rangle$.

The reason why the translation-based approach does not work for A-based NAU is that A is finite, while in higher-order anti-unification there is an infinite supply of fresh (bound) variables. If we assumed A to be infinite, there would still be a mismatch between NAU and the corresponding HOPAU: NAU, as we saw, is nullary in this case, while HOPAU is unitary. The reason of this contrast is that from infinitely many nominal generalizations, there is only one which is a well-typed higher-order generalization.

One might think that the translation-based approach would still work, if one considers only nominal anti-unification problems where the set of atoms is large enough for the input terms-in-context. However, there is a reason that speaks against NAU-to-HOPAU translation: complexity. The translation approach leads to a quadratic increase of the input size (Lemma 5.6 in [20]). The HOPAU algorithm in [3] runs in cubic time with respect to the size of its input. Hence, the translation-based approach leads to an algorithm with runtime complexity $O(n^6)$. In contrast, the algorithm developed in this paper has runtime complexity $O(n^5)$, and requires no back and forth translations.

4 The Lattice of More General Terms-In-Context

The notion of *more general term* defines an order relation between classes of terms (modulo some notion of *variable renaming*). In most cases, we have actually a meet-semilattice, since, given two terms, there always exists a greatest lower bound (meet) that corresponds to their anti-unifier. On the contrary, the least upper bound (join) of two terms only exists if they are unifiable. For instance, the two first-order terms $f(a, X_1)$ and $f(X_2, b)$ have a meet $f(Y_1, Y_2)$, and, since they are unifiable, also a join $f(a, b)$. Notice that unifiability and existence of a join are equivalent if both terms do not share variables (for instance $f(a, X)$ and $f(X, b)$ are both smaller than $f(a, b)$, hence joinable, but they are not unifiable). With this restriction one do not loose generality: The unification problem $t_1 \approx^? t_2$ (sharing variables), can be reduced to $f(t_1, t_2) \approx^? f(X, X)$ (not sharing variables), where f is some binary symbol and X a fresh variable. Therefore, in the first-order case, the problem of searching a most general unifier is equivalent to the search of the join of two terms, and the search of a least general generalization to the search of the meet. Notice that meet and join are unique up to some notion of *variable renaming*. For instance, the join of $f(a, X, X')$ and $f(Y, b, Y')$ is $f(a, b, Z)$ for any renaming of Z by any variable.

In the nominal case, we consider the set of terms-in-context (modulo variable renaming) with the more general relation. The following lemma establishes a correspondence between joinability and unifiability.

▶ **Lemma 4.1.** *Given two terms-in-context $\langle \nabla_1, t_1\rangle$ and $\langle \nabla_2, t_2\rangle$ with disjoint sets of variables, $\langle \nabla_1, t_1\rangle$ and $\langle \nabla_2, t_2\rangle$ are joinable if, and only if, $\{t_1 \approx^? t_2\} \cup \nabla_1 \cup \nabla_2$ has a solution (is unifiable).*

Like in first-order unification, the previous lemma allows us to reduce any nominal unification problem $P = \{a_1\#u_1, \ldots, a_m\#u_m, t_1 \approx s_1, \ldots, t_n \approx s_n\}$ into the joinability of the

two terms-in-context $\langle \emptyset, f(X,X)\rangle$ and $\langle \mathsf{FC}(\{a_1\#u_1,\ldots,a_m\#u_m\}), f(g(t_1,\ldots,t_n), g(s_1,\ldots,s_n))\rangle$ where f and g are any appropriate function symbols, and X is a fresh variable.

The nominal anti-unification problem is already stated in terms of finding the meet of two terms-in-context, with the only proviso that all terms and contexts must be based on some finite set of atoms.

5 Nominal Anti-Unification Algorithm

The triple $X : t \triangleq s$, where X, t, s have the same sort, is called the *anti-unification triple*, shortly AUT, and the variable X is called a *generalization variable*. We say that a set of AUTs P is based on a finite set of atoms A, if for all $X : t \triangleq s \in P$, the terms t and s are based on A.

▶ **Definition 5.1.** The nominal anti-unification algorithm is formulated in a rule-based way working on tuples P; S; Γ; σ and two global parameters A and ∇, where

- P and S are sets of AUTs such that if $X : t \triangleq s \in P \cup S$, then this is the sole occurrence of X in $P \cup S$;
- P is the set of AUTs to be solved;
- A is a finite set of atoms;
- The freshness context ∇ does not constrain generalization variables;
- S is a set of already solved AUTs (the store);
- Γ is a freshness context (computed so far) which constrains generalization variables;
- σ is a substitution (computed so far) mapping generalization variables to nominal terms;
- P, S, ∇, and Γ are A-based.

We call such a tuple a *state*. The rules below operate on states.

Dec: **Decomposition**

$$\{X : \mathsf{h}(t_1,\ldots,t_m) \triangleq \mathsf{h}(s_1,\ldots,s_m)\} \uplus P;\ S;\ \Gamma;\ \sigma \\ \Longrightarrow \{Y_1 : t_1 \triangleq s_1, \ldots, Y_m : t_m \triangleq s_m\} \cup P;\ S;\ \Gamma;\ \sigma\{X \mapsto \mathsf{h}(Y_1,\ldots,Y_m)\},$$

where h is a function symbol or an atom, $Y_1,\ldots,Y_m$ are fresh variables of the corresponding sorts, $m \geq 0$.

Abs: **Abstraction**

$$\{X : a.t \triangleq b.s\} \uplus P;\ S;\ \Gamma;\ \sigma \Longrightarrow \{Y : (c\,a) \bullet t \triangleq (c\,b) \bullet s\} \cup P;\ S;\ \Gamma;\ \sigma\{X \mapsto c.Y\},$$

where Y is fresh, $c \in A$, $\nabla \vdash c\#a.t$ and $\nabla \vdash c\#b.s$.

Sol: **Solving**

$$\{X : t \triangleq s\} \uplus P;\ S;\ \Gamma;\ \sigma \Longrightarrow P;\ S \cup \{X : t \triangleq s\};\ \Gamma \cup \Gamma';\ \sigma,$$

if none of the previous rules is applicable, i.e. one of the following conditions hold:

(a) both terms have distinct heads: $\mathrm{Head}(t) \neq \mathrm{Head}(s)$, or
(b) both terms are suspensions: $t = \boldsymbol{\pi}_1 \cdot Y_1$ and $s = \boldsymbol{\pi}_2 \cdot Y_2$, where $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2$ and Y_1, Y_2 are not necessarily distinct, or
(c) both are abstractions and rule Abs is not applicable: $t = a.t'$, $s = b.s'$ and there is no atom $c \in A$ satisfying $\nabla \vdash c\#a.t'$ and $\nabla \vdash c\#b.s'$.

The set Γ' is defined as $\Gamma' := \{a\#X \mid a \in A \ \wedge\ \nabla \vdash a\#t \ \wedge\ \nabla \vdash a\#s\}$.

Mer: **Merging**

$P;\ \{X : t_1 \triangleq s_1, Y : t_2 \triangleq s_2\} \cup S;\ \Gamma;\ \sigma \Longrightarrow$
$P;\ \{X : t_1 \triangleq s_1\} \cup S;\ \Gamma\{Y \mapsto \pi \cdot X\};\ \sigma\{Y \mapsto \pi \cdot X\},$

where π is an Atoms(t_1, s_1, t_2, s_2)-based permutation such that $\nabla \vdash \pi \bullet t_1 \approx t_2$, and $\nabla \vdash \pi \bullet s_1 \approx s_2$.

The rules transform states to states. One can easily observe this by inspecting the rules.

Given a finite set of atoms A, two nominal A-based terms t and s, and an A-based freshness context ∇, to compute A-based generalizations for $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$, we start with $\{X : t \triangleq s\}; \emptyset; \emptyset; \varepsilon$, where X is a fresh variable, and apply the rules as long as possible. We denote this procedure by $\mathfrak{N}$. A *Derivation* is a sequence of state transformations by the rules. The state to which no rule applies has the form $\emptyset; S; \Gamma; \varphi$, where Mer does not apply to S. We call it the *final state*. When $\mathfrak{N}$ transforms $\{X : t \triangleq s\}; \emptyset; \emptyset; \varepsilon$ into a final state $\emptyset; S; \Gamma; \varphi$, we say that the *result computed* by $\mathfrak{N}$ is $\langle \Gamma, X\varphi\rangle$.

Note that the Dec rule works also for the AUTs of the form $X : a \triangleq a$. In the Abs rule, it is important to have the corresponding c in A. If we take $A = A_2$ in Example 3.1, then Abs can transform the AUT between t and s there, but if $A = A_1$ in the same example, then Abs is not applicable. In this case the Sol rule takes over, because the condition (c) of this rule is satisfied.

The condition (b) of Sol helps to compute, e,g, $\langle \emptyset, X\rangle$ for identical terms-in-context $\langle \emptyset, (a\,b)\cdot Y\rangle$ and $\langle \emptyset, (a\,b)\cdot Y\rangle$. Although one might expect that computing $\langle \emptyset, (a\,b)\cdot Y\rangle$ would be more natural, from the generalization point of view it does not matter, because $\langle \emptyset, X\rangle$ is as general as $\langle \emptyset, (a\,b)\cdot Y\rangle$.

▶ **Example 5.2.** We illustrate $\mathfrak{N}$ with the help of some examples:

- Let $t = f(a, b)$, $s = f(b, c)$, $\nabla = \emptyset$, and $A = \{a, b, c, d\}$. Then $\mathfrak{N}$ performs the following transformations:

$$\begin{aligned}
&\{X : f(a,b) \triangleq f(b,c)\}; \emptyset; \emptyset; \varepsilon \Longrightarrow_{\mathsf{Dec}}\\
&\{Y : a \triangleq b, Z : b \triangleq c\}; \emptyset; \emptyset; \{X \mapsto f(Y,Z)\} \Longrightarrow^{2}_{\mathsf{Sol}}\\
&\emptyset; \{Y : a \triangleq b, Z : b \triangleq c\}; \{c\#Y, d\#Y, a\#Z, d\#Z\};\ \{X \mapsto f(Y,Z)\} \Longrightarrow_{\mathsf{Mer}}\\
&\emptyset; \{Y : a \triangleq b\}; \{c\#Y, d\#Y\}; \{X \mapsto f(Y, (a\,b)(b\,c)\cdot Y)\}
\end{aligned}$$

  Hence, $p = \langle \{c\#Y, d\#Y\}, f(Y, (a\,b)(b\,c)\cdot Y)\rangle$ is the computed result. It generalizes the input pairs: $p\{Y \mapsto a\} \preceq \langle \nabla, t\rangle$ and $p\{Y \mapsto b\} \preceq \langle \nabla, s\rangle$. The substitutions $\{Y \mapsto a\}$ and $\{Y \mapsto b\}$ can be read from the final store. Note that $\langle \{c\#Y\}, f(Y, (a\,b)(b\,c)\cdot Y)\rangle$ would be also an A-based generalization of $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$, but it is strictly more general than p.
- Let $t = f(b, a)$, $s = f(Y, (a\,b)\cdot Y)$, $\nabla = \{b\#Y\}$, and $A = \{a, b\}$. Then $\mathfrak{N}$ computes the term-in-context $\langle \emptyset, f(Z, (a\,b)\cdot Z)\rangle$. It generalizes the input pairs.
- Let $t = f(g(X), X)$, $s = f(g(Y), Y)$, $\nabla = \emptyset$, and $A = \emptyset$. It is a first-order anti-unification problem. $\mathfrak{N}$ computes $\langle \emptyset, f(g(Z), Z)\rangle$. It generalizes the input pairs.
- Let $t = f(a.b, X)$, $s = f(b.a, Y)$, $\nabla = \{c\#X\}$, $A = \{a, b, c, d\}$. Then $\mathfrak{N}$ computes the term-in-context $p = \langle \{c\#Z_1, d\#Z_1\}, f(c.Z_1, Z_2)\rangle$. It generalizes the input pairs: $p\{Z_1 \mapsto b, Z_2 \mapsto X\} = \langle \emptyset, f(c.b, X)\rangle \preceq \langle \nabla, t\rangle$ and $p\{Z_1 \mapsto a, Z_2 \mapsto Y\} = \langle \emptyset, f(c.a, Y)\rangle \preceq \langle \nabla, s\rangle$.

6 Properties of the Nominal Anti-Unification Algorithm

The Soundness Theorem states that the result computed by $\mathfrak{N}$ is indeed an A-based generalization of the input terms-in-context:

▶ **Theorem 6.1** (Soundness of $\mathfrak{N}$). *Given terms t and s and a freshness context ∇, all based on a finite set of atoms A, if $\{X : t \triangleq s\}; \emptyset; \emptyset;\ \varepsilon \Longrightarrow^{+} \emptyset; S; \Gamma; \sigma$ is a derivation obtained by an execution of $\mathfrak{N}$, then $\langle \Gamma, X\sigma\rangle$ is an A-based generalization of $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$.*

The Completeness Theorem states that for any given A-based generalization of two input terms-in-context, $\mathfrak{N}$ can compute one which is at most as general than the given one.

▶ **Theorem 6.2** (Completeness of $\mathfrak{N}$). *Given terms t and s and freshness contexts ∇ and Γ, all based on a finite set of atoms A. If $\langle \Gamma, r\rangle$ is an A-based generalization of $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$, then there exists a derivation $\{X : t \triangleq s\}; \emptyset; \emptyset;\ \varepsilon \Longrightarrow^{+} \emptyset; S; \Gamma'; \sigma$ obtained by an execution of $\mathfrak{N}$, such that $\langle \Gamma, r\rangle \preceq \langle \Gamma', X\sigma\rangle$.*

Depending on the selection of AUTs to perform a step, there can be different derivations in $\mathfrak{N}$ starting from the same AUT, leading to different generalizations. The next theorem states that all those generalizations are the same modulo variable renaming and α-equivalence.

▶ **Theorem 6.3** (Uniqueness Modulo $\simeq$). *Let t and s be terms and ∇ be a freshness context that are based on the same finite set of atoms. Let $\{X : t \triangleq s\}; \emptyset; \emptyset;\ \varepsilon \Longrightarrow^{+} \emptyset; S_1; \Gamma_1; \sigma_1$ and $\{X : t \triangleq s\}; \emptyset; \emptyset;\ \varepsilon \Longrightarrow^{+} \emptyset; S_2; \Gamma_2; \sigma_2$ be two maximal derivations in $\mathfrak{N}$. Then $\langle \Gamma_1, X\sigma_1\rangle \simeq \langle \Gamma_2, X\sigma_2\rangle$.*

Theorems 6.1, 6.2, and 6.3 imply that nominal anti-unification is unitary: For any A-based ∇, t, and s, there exists an A-based lgg of $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$, which is unique modulo $\simeq$ and can be computed by the algorithm $\mathfrak{N}$.

Now we study how lgg's of terms-in-context depend on the set of atoms the terms-in-context are based on. The following lemma states the precise dependence.

▶ **Lemma 6.4.** *Let A_1 and A_2 be two finite sets of atoms with $A_1 \subseteq A_2$ such that the A_1-based terms-in-context $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$ have an A_1-based lgg $\langle \Gamma_1, r_1\rangle$ and an A_2-based lgg $\langle \Gamma_2, r_2\rangle$. Then $\Gamma_2 \vdash r_1 \preceq r_2$.*

Proof. $\langle \Gamma_1, r_1\rangle$ and $\langle \Gamma_2, r_2\rangle$ are unique modulo $\simeq$. Let D_i be the derivation in $\mathfrak{N}$ that computes $\langle \Gamma_i, r_i\rangle$, $i = 1, 2$. The number of atoms in A_1 and A_2 makes a difference in the rule Abs: If there are not enough atoms in A_1, an Abs step in D_2 is replaced by a Sol step in D_1. It means that for all positions p of r_1, $r_2|_{\mathsf{p}}$ is also defined. Moreover, there might exist a subterm $r_1|_{\mathsf{p}}$, which has a form of suspension, while $r_2|_{\mathsf{p}}$ is an abstraction. For such positions, $r_1|_{\mathsf{p}} \preceq r_2|_{\mathsf{p}}$. For the other positions p' of r_1, $r_1|_{\mathsf{p}'}$ and $r_2|_{\mathsf{p}'}$ may differ only by names of generalization variables or by names of bound atoms.

Another difference might be in the application of Sol in both derivations: It can happen that this rule produces a larger Γ' in D_2 than in D_1, when transforming the same AUT.

Hence, if there are positions $\mathsf{p}_1, \ldots, \mathsf{p}_n$ in r_1 such that $r_1|_{\mathsf{p}_i} = \pi_i \cdot X$, then there exists a substitution φ_X such that $\Gamma_2 \vdash \pi_i \cdot X\varphi \approx r_2|_{\mathsf{p}_i}$, $1 \le i \le n$. Taking the union of all φ_X's where $X \in \mathrm{Vars}(r_1)$, we get φ with the property $\Gamma_2 \vdash r_1\varphi \approx r_2$. ◀

Note that, in general, we can not replace $\Gamma_2 \vdash r_1 \preceq r_2$ with $\Gamma_2 \vdash r_1 \simeq r_2$ in Lemma 6.4. The following example illustrates this:

▶ **Example 6.5.** Let $t = a.b$, $s = b.a$, $\nabla = \emptyset$, $A_1 = \{a, b\}$, and $A_2 = \{a, b, c\}$. Then for $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$, $\langle \emptyset, X\rangle$ is an A_1-based lgg and $\langle \{c\#X\}, c.X\rangle$ is an A_2-based lgg. Obviously, $\{c\#X\} \vdash X \preceq c.X$ but not $\{c\#X\} \vdash c.X \preceq X$.

This example naturally leads to a question: Under which additional conditions can we have $\Gamma_2 \vdash r_1 \simeq r_2$ instead of $\Gamma_2 \vdash r_1 \preceq r_2$ in Lemma 6.4? To formalize a possible answer to it, we need some notation.

Let the terms t, s and the freshness context ∇ be based on the same set of atoms A. The maximal subset of A, *fresh* for t, s, and ∇, denoted $\mathit{fresh}(A, t, s, \nabla)$, is defined as $A \setminus (\mathrm{Atoms}(t, s) \cup \mathrm{Atoms}(\nabla))$.

If $A_1 \subseteq A_2$ are two sets of atoms such that t, s, ∇ are at the same time based on both A_1 and A_2, then $\mathit{fresh}(A_1, t, s, \nabla) \subseteq \mathit{fresh}(A_2, t, s, \nabla)$.

Let $\|t\|_{\mathsf{Abs}}$ stand for the number of abstraction occurrences in t. $|A|$ stands for the cardinality of the set of atoms A. We say that a set of atoms A is *saturated* for A-based t, s and ∇, if $|\mathit{fresh}(A, t, s, \nabla)| \geq \min\{\|t\|_{\mathsf{Abs}}, \|s\|_{\mathsf{Abs}}\}$.

The following lemma answers the question posed above:

▶ **Lemma 6.6.** *Under the conditions of Lemma 6.4, if A_1 is saturated for t, s, ∇, then $\Gamma_2 \vdash r_1 \simeq r_2$.*

Proof. Let D_i be the derivation in $\mathfrak{N}$ that computes $\langle \Gamma_i, r_i\rangle$, $i = 1, 2$. Note that in each of these derivations, the number of Abs steps does not exceed $\min\{\|t\|_{\mathsf{Abs}}, \|s\|_{\mathsf{Abs}}\}$. Since A_1 is saturated for t, s, ∇ and $A_1 \subseteq A_2$, A_2 is also saturated for t, s, ∇. Hence, whenever an AUT between two abstractions is encountered in the derivation D_i, there is always $c \in A_1$ available which satisfies the condition of the Abs rule. Therefore, such AU-E's are never transformed by Sol. We can assume without loss of generality that the sequence of steps in D_1 and D_2 are the same. we may also assume that we take the same fresh variables, and the same atoms from $\mathit{fresh}(A_1, t, s, \nabla)$ in the corresponding steps in D_1 and D_2. Then the only difference between these derivations is in the Γ's, caused by the Sol rule which might eventually make Γ_2 larger than Γ_1. The σ's computed by the derivations are the same and, therefore, r_1 and r_2 are the same (modulo the assumptions on the variable and fresh atom names). Hence, $\Gamma_2 \vdash r_1 \simeq r_2$. ◀

In other words, this lemma answers the following pragmatic question: Given t, s and ∇, how to choose a set of atoms A so that (a) t, s, ∇ are A-based and (b) in the A-based lgg $\langle \Gamma, r\rangle$ of $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$, the term r generalizes s and t in the "best way", maximally preserving similarities and uniformly abstracting differences between s and t. The answer is: Besides all the atoms occurring in t, s, or ∇, A should contain at least m more atoms, where $m = \min\{\|t\|_{\mathsf{Abs}}, \|s\|_{\mathsf{Abs}}\}$.

Besides that, the lemma also gives the condition when the NAU-to-HOPAU translation can be used for solving NAU problems: The set of permitted atoms should be saturated.

7 Deciding Equivariance

Computation of π in the condition of the rule Mer above requires an algorithm that solves the following problem: Given nominal terms t, s and a freshness context ∇, find an $\mathrm{Atoms}(t, s)$-based permutation π such that $\nabla \vdash \pi \bullet t \approx s$. This is the problem of deciding whether t and s are equivariant with respect to ∇. In this Section we describe a rule-based algorithm for this problem, called $\mathfrak{E}$.

Note that our problem differs from the problem of equivariant unification considered in [9]: We do not solve unification problems, since we do not allow variable substitution.

We only look for permutations to *decide equivariance constructively* and provide a dedicated algorithm for that.

The algorithm $\mathfrak{E}$ works on tuples of the form $E;\ \nabla;\ A;\ \pi$ (also called states). E is a set of equivariance equations of the form $t \approx s$ where t, s are nominal terms, ∇ is a freshness context, and A is a finite set of atoms which are available for computing π. The latter holds the permutation to be returned in case of success.

The algorithm is split into two phases. The first one is a simplification phase where function applications, abstractions and suspensions are decomposed as long as possible. The second phase is the permutation computation, where given a set of equivariance equations between atoms of the form $a \approx b$ we compute the permutation which will be returned in case of success. The rules of the first phase are the following:

Dec-E: **Decomposition**

$$\{f(t_1, \ldots, t_m) \approx f(s_1, \ldots, s_m)\} \uplus E;\ \nabla;\ A;\ Id \Longrightarrow \{t_1 \approx s_1, \ldots, t_m \approx s_m\} \cup E; \nabla; A; Id.$$

Alp-E: **Alpha Equivalence**

$$\{a.t \approx b.s\} \uplus E;\ \nabla;\ A;\ Id \Longrightarrow \{(\acute{c}\ a) \bullet t \approx (\acute{c}\ b) \bullet s\} \cup E;\ \nabla;\ A;\ Id,$$

where $\acute{c}$ is a fresh atom of the same sort as a and b.

Sus-E: **Suspension**

$$\{\pi_1 \cdot X \approx \pi_2 \cdot X\} \uplus E;\ \nabla;\ A;\ Id \Longrightarrow \{\pi_1 \bullet a \approx \pi_2 \bullet a \mid a \in A \wedge a\#X \notin \nabla\} \cup E;\ \nabla;\ A;\ Id.$$

The rules of the second phase are the following:

Rem-E: **Remove**

$$\{a \approx b\} \uplus E;\ \nabla;\ A;\ \pi \Longrightarrow E;\ \nabla;\ A \setminus \{b\};\ \pi, \qquad \text{if } \pi \bullet a = b.$$

Sol-E: **Solve**

$$\{a \approx b\} \uplus E;\ \nabla;\ A;\ \pi \Longrightarrow E;\ \nabla;\ A \setminus \{b\};\ (\pi \bullet a\ \ b)\pi, \qquad \text{if } \pi \bullet a,\ b \in A \text{ and } \pi \bullet a \neq b.$$

Note that in Alp-E, $\acute{c}$ is fresh means that $\acute{c} \notin A$ and, therefore, $\acute{c}$ will not appear in π. These atoms are an auxiliary means which play a role during the computation but do not appear in the final result.

Given nominal terms t, s, freshness context ∇, we construct a state $\{t \approx s\};\ \nabla;\ \text{Atoms}(t, s);\ Id$. We will prove that when the rules transform this state into $\emptyset;\ \nabla;\ A;\ \pi$, then π is an $\text{Atoms}(t, s)$-based permutation such that $\nabla \vdash \pi \bullet t \approx s$. When no rule is applicable, and the set of equations is not empty, we will also prove that there is no solution, hence we fail and return $\bot$.

▶ **Example 7.1.** We illustrate the algorithm $\mathfrak{E}$ on examples:

- Consider the equivariance problem $E = \{a \approx a,\ a.(a\,b)(c\,d)\cdot X \approx b.X\}$ and $\nabla = \{a\#X\}$:

  $\{a \approx a, a.(a\,b)(c\,d)\cdot X \approx b.X\}; \{a\#X\};\ \{a, b, c, d\};\ Id \Longrightarrow_{\text{Alp-E}}$
  $\{a \approx a, (\acute{e}\ a)(a\,b)(c\,d)\cdot X \approx (\acute{e}\ b)\cdot X\}; \{a\#X\};\ \{a, b, c, d\};\ Id \Longrightarrow_{\text{Sus-E}}$
  $\{a \approx a, \acute{e} \approx e, c \approx d, d \approx c\}; \{a\#X\};\ \{a, b, c, d\};\ Id \Longrightarrow_{\text{Rem-E}}$
  $\{\acute{e} \approx \acute{e}, c \approx d, d \approx c\}; \{a\#X\};\ \{b, c, d\};\ Id \Longrightarrow_{\text{Rem-E}}$
  $\{c \approx d, d \approx c\}; \{a\#X\};\ \{b, c, d\};\ Id \Longrightarrow_{\text{Sol-E}}$
  $\{d \approx c\};\ \{a\#X\};\ \{b, c\};\ (c\,d) \Longrightarrow_{\text{Rem-E}}$
  $\emptyset;\ \{a\#X\};\ \{b\};\ (c\,d).$

- For $E = \{a.f(b, X) \approx b.f(a, X)\}$ and $\nabla = \{a\#X\}$, $\mathfrak{E}$ returns $\bot$.
- For $E = \{a.f(b, (a\,b)\cdot X) \approx b.f(a, X)\}$ and $\nabla = \{a\#X\}$, $\mathfrak{E}$ returns $(b\,a)$.
- For $E = \{a.b.(a\,b)(a\,c)\cdot X = b.a.(a\,c)\cdot X\}$ and $\nabla = \emptyset$, $\mathfrak{E}$ returns Id.
- For $E = \{a.b.(a\,b)(a\,c)\cdot X = a.b.(b\,c)\cdot X\}$ and $\nabla = \emptyset$, $\mathfrak{E}$ returns $\bot$.

The Soundness Theorem for $\mathfrak{E}$ states that, indeed, the permutation the algorithm computes shows that the input terms are equivariant:

▶ **Theorem 7.2** (Soundness of $\mathfrak{E}$). *Let* $\{t \approx s\}; \nabla; A; Id \Longrightarrow^* \emptyset; \nabla; B; \pi$ *be a derivation in* $\mathfrak{E}$*, then* π *is an* A*-based permutation such that* $\nabla \vdash \pi \bullet t \approx s$.

We now prove an invariant lemma that is used in the proof of completeness Theorem 7.4.

▶ **Lemma 7.3** (Invariant Lemma). *Let* A *be a finite set of atoms,* E_1 *be a set of equivariance equations for terms based on* A*,* π_1 *be an* A*-based permutation and* $A_1 \subseteq A$*. Let* $E_1; \nabla; A_1; \pi_1 \Longrightarrow E_2; \nabla; A_2; \pi_2$ *be any step performed by a rule in* $\mathfrak{E}$*. Let* $\Gamma = \{\acute{c}\#X \mid X \in \mathrm{Vars}(E_1), \acute{c}$ *is a fresh variable*$\}$*. Let* μ *be an* A*-based permutation such that* $\nabla \cup \Gamma \vdash \mu \bullet t \approx s$*, for all* $t \approx s \in E_1$*. Then*

1. $\nabla \cup \Gamma \vdash \mu \bullet t' \approx s'$*, for all* $t' \approx s' \in E_2$.
2. *If* $\mu^{-1} \bullet b = \pi_1^{-1} \bullet b$*, for all* $b \in A \setminus A_1$*, then* $\mu^{-1} \bullet b = \pi_2^{-1} \bullet b$*, for all* $b \in A \setminus A_2$.

Proof. By case distinction on the applied rule.

Dec-E: The proposition is obvious.

Alp-E: In this case it follows from the definitions of $\approx$ and permutation application.

Sus-E: In this case $t = \tau_1 \cdot X$, $s = \tau_2 \cdot X$, and by the assumption we have $\nabla \vdash \mu\tau_1 \cdot X \approx \tau_2 \cdot X$. By the definition of $\approx$, it means that we have $a\#X \in \nabla$, for all atoms a such that $\mu\tau_1 \bullet a \neq \tau_2 \bullet a$. Hence, for all $a \in A$ with $a\#X \notin \nabla$ we have $\nabla \vdash \mu\tau_1 \bullet a \approx \tau_2 \bullet a$. This implies that μ also solves the equations in E_2, hence item 1 of the lemma.

Item 2 of the lemma is trivial for these three rules, since $A_1 = A_2$ and $\pi_1 = \pi_2 = Id$.

Rem-E: The item 1 is trivial. To prove the item 2, note that $t = a$, $s = b$, $\pi_1 = \pi_2$ and we only need to show $\mu^{-1} \bullet b = \pi_2^{-1} \bullet b$. By the assumption we have $\nabla \vdash \mu \bullet a \approx b$. Since a and b are atoms, the latter simply means that $\mu \bullet a = b$. From the rule condition we also know that $\pi_1 \bullet a = b$. From these two equalities we get $\mu^{-1} \bullet b = a = \pi_2^{-1} b$.

Sol-E: The item 1 is trivial also in this case. To prove the item 2, note that $t = a$, $s = b$, $\pi_2 = (\pi_1 \bullet a\ b)\pi_1$ and we only need to show $\mu^{-1} \bullet b = \pi_2^{-1} \bullet b$. By the assumption we have $\nabla \vdash \mu \bullet a \approx b$, which means that $\mu \bullet a = b$ and, hence, $a = \mu^{-1} \bullet b$. As for $\pi_2^{-1} \bullet b$, we have $\pi_2^{-1} \bullet b = \pi_1^{-1}(\pi_1 \bullet a\ b) \bullet b = \pi_1^{-1} \bullet (\pi_1 \bullet a) = a$. Hence, we get $\mu^{-1} \bullet b = a = \pi_2^{-1} \bullet b$. ◀

▶ **Theorem 7.4** (Completeness of $\mathfrak{E}$). *Let* A *be a finite set of atoms,* t, s *be* A*-based terms, and* ∇ *be a freshness context. If* $\nabla \vdash \mu \bullet t \approx s$ *holds for some* A*-based permutation* μ*, then there exists a derivation* $\{t \approx s\}; \nabla; A; Id \Longrightarrow^* \emptyset; \Gamma; B; \pi$*, obtained by an execution of* $\mathfrak{E}$*, such that* $\pi \bullet a = \mu \bullet a$ *for any atom* $a \in \mathrm{FA}(t)$.

8 Complexity Analysis

We represent a permutations π as two hash tables. One for the permutation itself, we call it T_π, and one for the inverse of the permutation, called $T_{\pi^{-1}}$. The key of a hash tables is an atom and we associate another atom, the mapping, with it. For instance the permutation $\pi = (a\,b)(a\,c)$ is represented as $T_\pi = \{a \mapsto c, b \mapsto a, c \mapsto b\}$ and $T_{\pi^{-1}} = \{a \mapsto b, b \mapsto c, c \mapsto a\}$. We write $T_\pi(a)$ to obtain from the hash table T_π the atom which is associated with the key

a. If no atom is associated with the key a then $T_\pi(a)$ returns a. We write $T_\pi(a \mapsto b)$, to set the mapping such that $T_\pi(a) = b$. As the set of atoms is small, we can assume a perfect hash function. It follows, that both defined operations are done in constant time, leading to constant time application of a permutation. Swapping application to a permutation $(a\,b)\pi$ is also done in constant time in the following way: Obtain $c = T_{\pi^{-1}}(a)$ and $d = T_{\pi^{-1}}(b)$ and perform the following updates:

(a) $T_\pi(c \mapsto b)$ and $T_\pi(d \mapsto a)$,
(b) $T_{\pi^{-1}}(b \mapsto c)$ and $T_{\pi^{-1}}(a \mapsto d)$.

We also represent set membership of atoms to a set of atoms A with a hash table $\in_A$ from atoms to Booleans such that $\in_A (a) = \mathit{true}$ iff $a \in A$. We also have a list L_A of the atoms representing the entries of the table such that $\in_A (a) = \mathit{true}$ to easily know all atoms in A.

Finally we also represent set membership of freshness constraints to a freshness environment ∇ with a hash table $\in_\nabla$.

▶ **Theorem 8.1.** *Given a set of equivariance equations E, and a freshness context ∇. Let m be the size of ∇, and let n be the size of E. The algorithm $\mathfrak{E}$ has $O(n^2+m)$ time complexity.*

Proof. Collecting the atoms from E in a separate set A does not affect the space complexity and can be done in time $O(n)$. The freshness environment ∇ will not be modified by rule applications and membership test in the rule Sus-E can be done in constant time. We only have to construct the corresponding hash tables in time $O(m)$. We analyze complexity of both phases.

For the first phase, notice that all rules can be applied only $O(n)$ many times, since Dec-E removes two function symbols and Alp-E two abstraction, and Sus-E two suspensions. The resulting equations after this phase only contain atoms. However, notice that the size of these equations is not necessarily linear. Every time we apply Alp-E a new swapping is applied to both subterms. This swappings may increase the size of suspensions occurring bellow the abstraction. Since there are $O(n)$ many suspensions and $O(n)$ many abstractions, the final size of suspensions is $O(n^2)$. This is the size of the atom equations at the beginning of the second phase. We can see that the application of Dec-E rule has $O(1)$ time complexity (with the appropriate representation of equations).

The application of Alp-E rule requires to find a fresh atom not in A, this can be done in constant time. Later, a swapping has to be applied twice. Swapping application requires traversing the term hence has $O(n)$ time complexity. The application of Sus-E requires to traverse L_A ($O(n)$) and check for freshness membership in $\in_\nabla$ ($O(1)$). Finally it has to add equations like $(\pi_1 \bullet a \approx \pi_2)$, this requires to build T_{π_1} and T_{π_2} that can be done in $O(n)$ time complexity and allow us to build each equation in $O(1)$ time. Summing up, this phase has $O(n^2)$ time complexity.

For the second phase, notice that both rules Rem-E and Sol-E remove an equation and do not introduce any other one. Hence, potentially having $O(n^2)$ many equations in this phase, these equations can be applied $O(n^2)$ may times. We construct a hash table T_π for π that will be maintained and used by both rules. Each application has time complexity $O(1)$. Rem-E uses T_π to check for applicability and if it is applied, it only removes b from A, hence updating $\in_A$ (notice that we do not care about L_A in this second phase of the algorithm). Sol-E uses $\in_A$ and T_π to check for applicability and if it is applied, it only removes b from A (hence updating $\in_A$), and updates T_π. Summing up, this phase maintains the overall $O(n^2)$ time complexity. ◀

▶ **Theorem 8.2.** *The nominal anti-unification algorithm* $\mathfrak{N}$ *has* $O(n^5)$ *time complexity and* $O(n^4)$ *space complexity, where* n *is the input size.*

Proof. By design of the rules and theorem 6.3 we can arrange a maximal derivation like $\{X_0 : t_0 \triangleq s_0\}; \emptyset; \emptyset; \varepsilon \Longrightarrow^*_{\mathsf{Dec,Abs,Sol}} \emptyset; S_l; \Gamma_l; \sigma_l \Longrightarrow^*_{\mathsf{Mer}} \emptyset; S_m; \Gamma_m; \sigma_m$, postponing the application of Mer until the end. Rules Dec, Abs and Sol can be applied $O(n)$ many times. However, notice that every application of Abs may increase the size of every suspension below. Hence, the size of the store S_l is $O(n^2)$, although it only contain $O(n)$ equations, after an exhaustive derivation $\{X_0 : t_0 \triangleq s_0\}; \emptyset; \emptyset; \varepsilon \Longrightarrow^*_{\mathsf{Dec,Abs,Sol}} \emptyset; S_l; \Gamma_l; \sigma_l$.

Now we turn to analyzing the transformation phase $\emptyset; S_l; \Gamma_l; \sigma_l \Longrightarrow^*_{\mathsf{Mer}} \emptyset; S_m; \Gamma_m; \sigma_m$. Let $S_l = \{X_1 : t_1 \triangleq s_1, \ldots, X_k : t_k \triangleq s_k\}$ and n_i be the size of $X_i : t_i \triangleq s_i$, $1 \leq i \leq k$, then $\sum_{i=1}^k n_i = O(n^2)$ and $k = O(n)$. From theorem 8.1 we know that solving the equivariance problem for two AUPs $X_i : t_i \triangleq s_i$ and $X_j : t_j \triangleq s_j$ and an arbitrary freshness context ∇ requires $O((n_i + n_j)^2 + m)$ time and space, where m is the size of ∇ with $m = O(n)$.

Merging requires to solve this problem for each pair of AUPs. This leads to the time complexity $\sum_{i=1}^k \sum_{j=i+1}^k O((n_i+n_j)^2+m) \leq O(\sum_{i=1}^k \sum_{j=1}^k (n_i+n_j)^2) + O(\sum_{i=1}^k \sum_{j=1}^k m)$. The second sum is $\sum_{i=1}^k \sum_{j=1}^k m = k^2 m = O(n^3)$. Now we estimate an upper bound for the sum $\sum_{i=1}^k \sum_{j=1}^k (n_i + n_j)^2 = \sum_{i=1}^k \sum_{j=1}^k n_i^2 + \sum_{i=1}^k \sum_{j=1}^k 2n_i n_j + \sum_{i=1}^k \sum_{j=1}^k n_j^2 \leq \sum_{i=1}^k k n_i^2 + 2\left(\sum_{i=1}^k n_i\right)\left(\sum_{j=1}^k n_j\right) + \sum_{i=1}^k (\sum_{j=1}^k n_j)^2 \leq k(\sum_{i=1}^k n_i)^2 + 2O(n^2)O(n^2) + \sum_{i=1}^k O(n^2) = kO(n^2)^2 + 2O(n^2)^2 + kO(n^2)^2 = O(n^5)$, resulting into the stated bounds.

The space is bounded by the space required by a single call to the equivariance algorithm with an imput of size $O(n^2)$, hence $O(n^4)$. ◀

9 Conclusion

The problem of anti-unification for nominal terms-in-context is sensitive to the set of atoms permitted in generalizations: If this set is infinite, there is no least general generalization. Otherwise there exists a unique lgg. If this set is finite and satisfies the notion of being saturated, defined in the paper, then the lgg retains the common structure of the input nominal terms maximally.

We illustrated that, similar to some other theories where unification, generalization, and the subsumption relation are defined, the nominal terms-in-contexts form a join-meet lattice with respect to the subsumption relation, where the existence of join is unifiability, and the meet corresponds to least general generalization.

We designed an anti-unification algorithm for nominal terms-in-context. It contains a subalgorithm that constructively decides whether two terms are equivariant with respect to the given freshness context. We proved termination, soundness, and completeness of these algorithms, investigated their complexities, and implemented them. Given a fixed set of atoms A, the nominal anti-unification algorithm computes a least general A-based term-in-context generalization of the given A-based terms-in-context, and requires $O(n^5)$ time and $O(n^4)$ space for that, where n is the size of the input. The computed lgg is unique modulo α-equivalence and variable renaming.

References

1 María Alpuente, Santiago Escobar, José Meseguer, and Pedro Ojeda. A modular equational generalization algorithm. In Michael Hanus, editor, *LOPSTR*, volume 5438 of *LNCS*, pages 24–39. Springer, 2008.

2 Alexander Baumgartner and Temur Kutsia. A library of anti-unification algorithms. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 543–557. Springer, 2014.

3 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. A variant of higher-order anti-unification. In Femke van Raamsdonk, editor, *RTA*, volume 21 of *LIPIcs*, pages 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

4 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal Anti-Unification. Technical Report 15-03, RISC, JKU Linz, April 2015.

5 Christophe Calvès. *Complexity and Implementation of Nominal Algorithms*. PhD thesis, King's College London, 2010.

6 Christophe Calvès and Maribel Fernández. A polynomial nominal unification algorithm. *Theor. Comput. Sci.*, 403(2-3):285–306, 2008.

7 Christophe Calvès and Maribel Fernández. Matching and alpha-equivalence check for nominal terms. *J. Comput. Syst. Sci.*, 76(5):283–301, 2010.

8 James Cheney. Relating nominal and higher-order pattern unification. In Laurent Vigneron, editor, *UNIF'05*, LORIA A05-R-022, pages 105–119, 2005.

9 James Cheney. Equivariant unification. *JAR*, 45(3):267–300, 2010.

10 Gilles Dowek, Murdoch J. Gabbay, and Dominic P. Mulligan. Permissive nominal terms and their unification. In *24th Italian Conference on Computational Logic, CILC'09*, 2009.

11 Gilles Dowek, Murdoch James Gabbay, and Dominic P. Mulligan. Permissive nominal terms and their unification: an infinite, co-infinite approach to nominal techniques. *Logic Journal of the IGPL*, 18(6):769–822, 2010.

12 Cao Feng and Stephen Muggleton. Towards inductive generalization in higher order logic. In Derek H. Sleeman and Peter Edwards, editors, *ML*, pages 154–162. Morgan Kaufmann, 1992.

13 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

14 Murdoch J. Gabbay. *A Theory of Inductive Definitions with alpha-Equivalence*. PhD thesis, University of Cambridge, UK, 2000.

15 Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *LICS*, pages 214–224. IEEE Computer Society, 1999.

16 Ulf Krumnack, Angela Schwering, Helmar Gust, and Kai-Uwe Kühnberger. Restricted higher-order anti-unification for analogy making. In Mehmet A. Orgun and John Thornton, editors, *Australian Conference on Artificial Intelligence*, volume 4830 of *LNCS*, pages 273–282. Springer, 2007.

17 Temur Kutsia, Jordi Levy, and Mateu Villaret. Anti-unification for unranked terms and hedges. In Manfred Schmidt-Schauß, editor, *RTA*, volume 10 of *LIPIcs*, pages 219–234. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

18 Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. In Andrei Voronkov, editor, *RTA*, volume 5117 of *LNCS*, pages 246–260. Springer, 2008.

19 Jordi Levy and Mateu Villaret. An efficient nominal unification algorithm. In Christopher Lynch, editor, *Proceedings of the 21st International Conference on Rewriting Techniques and Applications, RTA 2010, July 11-13, 2010, Edinburgh, Scottland, UK*, volume 6 of *LIPIcs*, pages 209–226. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

20 Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. *ACM Trans. Comput. Log.*, 13(2):10, 2012.

21 Jianguo Lu, John Mylopoulos, Masateru Harao, and Masami Hagiya. Higher order generalization and its application in program verification. *Ann. Math. Artif. Intell.*, 28(1-4):107–126, 2000.

22 Frank Pfenning. Unification and anti-unification in the calculus of constructions. In *LICS*, pages 74–85. IEEE Computer Society, 1991.

23 Gordon D. Plotkin. A note on inductive generalization. *Machine Intel.*, 5(1):153–163, 1970.

24 Luc De Raedt. *Logical and Relational Learning*. Springer, 2008.

25 John C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intel.*, 5(1):135–151, 1970.

26 Ute Schmid. *Inductive Synthesis of Functional Programs, Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning*, volume 2654 of *LNCS*. Springer, 2003.

27 Christian Urban. Nominal techniques in Isabelle/HOL. *J. Autom. Reasoning*, 40(4):327–356, 2008.

28 Christian Urban and James Cheney. Avoiding equivariance in alpha-prolog. In Paweł Urzyczyn, editor, *Typed Lambda Calculi and Applications*, volume 3461 of *LNCS*, pages 401–416. Springer Berlin Heidelberg, 2005.

29 Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. In Matthias Baaz and Johann A. Makowsky, editors, *CSL*, volume 2803 of *LNCS*, pages 513–527. Springer, 2003.

30 Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1–3):473–497, 2004.