Towards a Well-Founded $\preceq$ Relation for Permissive Nominal Terms

Alexander Baumgartner

Universidad de O'Higgins, Rancagua, Chile

Abstract

This work discusses the relation of more general terms ($\preceq$) in the permissive nominal language. First, we show that the permissive nominal language exhibits the same problem as the classical nominal language, namely, that the relation $\preceq$ is not well-founded. Second, we propose a modification in one of the original definitions that leads to a well-founded $\preceq$ relation. Third, we formulate an anti-unification algorithm that computes a unique least general generalization of two arbitrary input terms. The proposed modification yields a language that includes the original language from Dowek et al.. Since the original language is of generalization type zero, computed generalizations might be outside of it.

1 Introduction

This work discusses the relation that states that one term is more general than another one ($\preceq$), and the problem of finding a generalization of two input terms (called anti-unification problem), in the setting of permissive nominal terms [9]. Interesting generalizations are the least general ones (lgg). They represent parts of the input terms that they have in common. Finding an lgg has various real-world applications, e.g., in clone detection [5], analogy making [13], or parallel recursion scheme detection [1]. To be able to find an lgg, the relation $\preceq$ needs to be well-founded.

In the setting of nominal terms only atoms can be bounded. This yields some nice computational properties [14, 20] that are missing in λ-calculus where arbitrary variables may be bound [12, 15]. While the anti-unification problem has already been studied in various settings with binders [3, 6, 7, 8, 10, 17], including classical nominal terms [2, 4, 19], to the best of our knowledge, it has not yet been studied in the setting of permissive nominal terms.

It's a known fact that in the case of classical nominal terms $\preceq$ is not well-founded [2]. I.e., without restriction the anti-unification problem is of type zero (no lgg exists). One possible approach to overcome this issue is to introduce atom-variables [19]. Unfortunately, without any restrictions, that approach leads to intractable algorithms due to its intrinsic complexity.

First, we show that, like in the classical setting, $\preceq$ is not well-founded in the permissive case either. Second, we suggest a modification of the definition of permission sets which leads to a well-founded $\preceq$ relation. Permission sets define the atoms that are allowed to appear freely when instantiating a variable. The modification yields a term language that is a superset of the original one introduced by Dowek et al.. Third, we formulate an anti-unification algorithm.

2 Preliminaries

This work focuses on permissive nominal terms, introduced in [9]. We assume that the reader is familiar with them. In the following, main concepts and notions are being introduced.

Permissive Nominal Terms. Fix a countable infinite set of *atoms* $\mathbb{A} = \{a, b, c, \ldots\}$. Atoms are identified by their name, i.e., different names imply different elements from $\mathbb{A}$. In the permissive nominal setting, $\mathbb{A}$ is partitioned into two countably infinite sets $\mathbb{A}^<$ and $\mathbb{A}^>$. For

instance, $\mathbb{A}^<$ might be identified with even numbers while $\mathbb{A}^>$ corresponds to the odd ones. A *signature* $\Sigma = \{f, g, \ldots\}$ is a set of *symbols* of certain arity, such that $\Sigma \cap \mathbb{A} = \emptyset$. If the arity of a symbol is clear from the context, we won't mention it. A *permutation* π is a bijection on $\mathbb{A}$ that is identity almost everywhere. Permutations are represented by finite sequences of swappings.[1] *Swappings* are written as $(a\ b)$ where $a, b \in \mathbb{A}$. *Id* denotes the identity permutation. The composition of two permutations π and π', denoted by $\pi \circ \pi'$, is equivalent to the concatenation of their representations as sequences of swappings.

The set $\mathcal{P}$ of all *permission sets* is defined as

$$\mathcal{P} = \{A \cup B \mid A \subseteq \mathbb{A}^<, B \subset \mathbb{A}^>, \text{ and } B \text{ finite}\}.$$

In contrast to [9], our permission sets are not necessarily infinite. This decision is discussed in section 3. S, T denote arbitrary permission sets. Since B is restricted to be finite in the definition of $\mathcal{P}$, the subtraction of a (finite number of) permission set(s) from $\mathbb{A}$ yields a countably infinite set. Therefore, permission sets are *coinfinite* w.r.t. $\mathbb{A}$, i.e., from $S \in \mathcal{P}$ follows that $\mathbb{A} \setminus S$ is an infinite set. Informally, this means that there are always "fresh" atoms available. It's an important property that guarantees that bound atoms can always be renamed.

For each permission set S, fix a countable infinite set of *variables* $\mathbb{X}^S = \{X^S, Y^S, Z^S, \ldots\}$ of sort S, disjoint from $\mathbb{A}$ and Σ. Moreover, $\mathbb{X}^S$ and $\mathbb{X}^T$ are disjoint for any permission set $T \neq S$.

Example 1. $X^{\{a,b\}}$ is of finite sort $\{a, b\}$, while $X^{\mathbb{A}^< \cup \{b\}}$ is of infinite sort $\mathbb{A}^< \cup \{b\}$. $X^\emptyset$ is of sort $\emptyset$. Sorts (i.e., permission sets) define the atoms that are allowed to appear freely in instantiations (see Example 2).

Permissive nominal terms are built by the grammar:

$$t, t_i ::= a \mid \lambda a.t \mid \pi \cdot X^S \mid f(t_1, \ldots, t_n) \qquad \pi ::= Id \mid (a\ b) \circ \pi$$

where a, b are atoms, f is a symbol of arbitrary but fixed arity n, $\lambda a.t$ denotes the *abstraction* of atom a in the permissive nominal term t, $\pi \cdot X^S$ is a *suspension* of the permutation π on the variable X^S, and $f(t_1, \ldots, t_n)$ is a *function application*. In the following, the word *term* refers to permissive nominal terms, if not specified otherwise.

Permutation application to a term is defined recursively and gets suspended in front of variables, as usual. We overload the notation, writing, e.g., $\pi \cdot t$ where π is a permutation and t a term. A permutation π may be applied to a permission set S by $\pi \cdot S = \{\pi(a) \mid a \in S\}$. The set of *free atoms* of some terms $t_1, \ldots, t_n$, denoted by $\mathtt{fa}(t_1, \ldots, t_n)$, is defined as $\mathtt{fa}(t_1, t_2, \ldots, t_n) = \mathtt{fa}(t_1) \cup \mathtt{fa}(t_2, \ldots, t_n)$, $\mathtt{fa}(a) = \{a\}$, $\mathtt{fa}(\lambda a.t) = \mathtt{fa}(t) \setminus \{a\}$, $\mathtt{fa}(\pi \cdot X^S) = \pi \cdot S$, and $\mathtt{fa}(f(t_1, \ldots, t_n)) = \mathtt{fa}(t_1, \ldots, t_n)$.

Two terms are *α-equivalent* if they are equal up to renaming of bound atoms (Figure 1). The predicate $=_\alpha$ is used to denote α-equivalence.

$$\frac{}{a =_\alpha a} \qquad \frac{\forall a \in S : \pi(a) =_\alpha \pi'(a)}{\pi \cdot X^S =_\alpha \pi' \cdot X^S}$$

$$\frac{s_1 =_\alpha t_1 \quad \cdots \quad s_n =_\alpha t_n}{f(s_1, \ldots s_n) =_\alpha f(t_1, \ldots, t_n)} \qquad \frac{s =_\alpha t}{\lambda a.s =_\alpha \lambda a.t} \qquad \frac{s =_\alpha (a\ b) \cdot t}{\lambda a.s =_\alpha \lambda b.t \qquad a \notin \mathtt{fa}(t)}$$

Figure 1: α-equivalence rules as defined in [9].

[1] Representations are not unique. In any case, an arbitrary representation can be chosen.

More General Relation $\preceq$. A *substitution* σ is a function that maps variables to terms such that for any arbitrary variable X^S holds that $\mathtt{fa}(\sigma(X^S)) \subseteq S$. Substitutions are identity almost everywhere. The identity substitution is denoted by *id* and arbitrary substitutions are denoted by σ, ρ. The composition of two substitutions and the action of a substitution on a term are defined as usual. We use the postfix notation like $t\sigma$, where t is a term and σ a substitution. The application of a substitution to a term is called *instantiation* and yields another term.

Given two terms t and s. We say that t is *more general* than s, denoted as $t \preceq s$, if there exists a substitution σ such that $t\sigma =_\alpha s$. The relation $t \prec s$ denotes that t is *strictly more general* than s, i.e., $t \preceq s$ but not $s \preceq t$. The notions of *less general* and *strictly less general* are defined analogously. Moreover, $t \simeq s$ means that t and s are *equi-general*, i.e., $t \preceq s$ and $s \preceq t$. A term r is a *generalization* of t and s if $r \preceq t$ and $r \preceq s$. It is a *least general generalization* (lgg) of t and s if there is no generalization r' of t and s such that $r \prec r'$.

Example 2. $\{X^\emptyset \mapsto \lambda a.f(a)\}$ is a substitution, while $\{X^\emptyset \mapsto f(a)\}$ is not. Therefore, $X^\emptyset \preceq \lambda a.f(a)$, while $X^\emptyset \not\preceq f(a)$. The instantiation $g(Y^S)\{Y^S \mapsto f(a)\} = g(f(a))$ is valid if $a \in S$.

3 Generalization Type

To discuss the generalization type of permissive nominal terms and our motivation of deviating from the original approach, we recall the original definition of permission sets [9]. Dowek et al. permission sets are defined as

$$\mathcal{P}_D = \{(\mathbb{A}^< \setminus A) \cup B \mid A \subset \mathbb{A}^<, B \subset \mathbb{A}^>, \text{ and } A, B \text{ are finite}\}.$$

Dowek et al. permission sets are infinite and coinfinite w.r.t. $\mathbb{A}$. On the other hand, our definition yields permission sets that are only coinfinite w.r.t. $\mathbb{A}$, but not necessarily infinite. Note that $\mathcal{P}$ also includes permission sets of the form $(\mathbb{A}^< \setminus A) \cup B$, where $A \subset \mathbb{A}^<, B \subset \mathbb{A}^>$ and A, B are finite. Therefore, $\mathcal{P}_D \subset \mathcal{P}$, i.e., our setting is more general.

Theorem 1. When using Dowek et al. permission sets, the relation $\preceq$ is not well-founded.

Proof. Since $\mathbb{A}^<$ and $\mathbb{A}^>$ are disjoint, any permission set from $\mathcal{P}_D$ corresponds to a set of the form $S \setminus A$ where S consists of all atoms from $\mathbb{A}^<$ and finitely many atoms from $\mathbb{A}^>$, and $A \subset \mathbb{A}^<$ is finite. By definition, any substitution σ must satisfy $\mathtt{fa}(\sigma(X^{S\setminus A})) \subseteq S \setminus A$ for any A. It follows that $X^{S\setminus\emptyset} \prec X^{S\setminus\{a_1\}} \prec X^{S\setminus\{a_1,a_2\}} \prec \ldots$. Since A is finite, $\preceq$ is not well-founded. □

Corollary 1. Given two atoms a and b there is no lgg of a and b in the setting of $\mathcal{P}_D$.

This problem arises for any two arbitrary terms t and s that have a finite number of free atoms, i.e., where $\mathtt{fa}(t, s)$ is a finite set. For instance, the terms $f(a)$ and $g(b)$ do not have an lgg in the setting of Dowek et al.. Revising the proof of Theorem 1, it is easy to observe that the problem arises from the infinite nature of the Dowek et al. permission sets.

Also note that the same problem arises in the classical nominal setting. Baumgartner et al. [2] showed that, when considering an infinite supply of atoms, there are infinite chains of the form $\langle\emptyset, X\rangle \prec \langle\{a_1\#X\}, X\rangle \prec \langle\{a_1\#X, a_2\#X\}, X\rangle \prec \ldots$. To overcome that issue, the supply of atoms may be restricted to a finite set $\mathcal{A}$ so that the statement $\langle\{a\#X \mid a \in \mathcal{A}\}, X\rangle$ becomes valid. For more details about the classical nominal setting we refer the reader to [2, 11, 18].

Our suggestion to allow finite sets in $\mathcal{P}$ was motivated by the goal of overcoming that issue.

Example 3. Considering finite permission sets, we get a variable $X^{\{a,b\}}$ as the lgg of two atoms a and b. I.e., only the atoms a and b are allowed to appear freely in instantiations of $X^{\{a,b\}}$. Note that $X^{\{a,b\}}$ is also an lgg of the two terms $f(a)$ and $g(b)$.

In our setting, there is still an infinite supply of atoms (e.g., to rename bound atoms), and, any valid term in the setting of Dowek et al. is also valid in our setting. Moreover, we can find an lgg of any two input terms without any restriction. Theorem 2 establishes that the generalization type becomes unitary in this setting.

Theorem 2. Given two arbitrary terms there exists a unique lgg up to $\simeq$.

The proof of Theorem 2 follows from section 4 and 5 where we construct an anti-unification algorithm that yields a unique output term of two arbitrary input terms and prove its properties.

4 Anti-Unification Algorithm NAU$_\mathcal{P}$

Given two terms t and s, an *anti-unification equation* is a triple $X^S : t \triangleq s$ where X^S is a variable of sort $\mathtt{fa}(t, s)$ that neither appears in t nor in s. X^S is called the *generalization variable*.

The anti-unification algorithm for permissive nominal terms, called NAU$_\mathcal{P}$, is formulated in terms of transformation rules that work on triples of the form $E;\ Q;\ \sigma$, where E and Q are sets of anti-unification equations, and σ is a substitution. Such triples are called the *states* of the algorithm. The transformation rules are given in Figure 2. A variable is called *fresh* if it didn't already appear in any of the former states of the transformation process. We use $\uplus$ to denote the disjoint union.

Atm: **Atom**

$$\{X^S : a \triangleq a\} \uplus E;\ Q;\ \sigma \Longrightarrow E;\ Q;\ \sigma\{X^S \mapsto a\}.$$

Dec: **Decomposition**

$$\{X^S : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_n)\} \uplus E;\ Q;\ \sigma \Longrightarrow \{Y_1^{S_1} : t_1 \triangleq s_1, \ldots, Y_n^{S_n} : t_n \triangleq s_n\} \cup E;\ Q;\ \sigma\{X^S \mapsto f(Y_1^{S_1}, \ldots, Y_n^{S_n})\},$$

where f is a symbol of arity $n \geq 0$, and $Y_i^{S_i}$ is a fresh variable of sort $S_i = \mathtt{fa}(t_i, s_i)$, for all $1 \leq i \leq n$.

Abs: **Abstraction**

$$\{X^S : \lambda a.t \triangleq \lambda b.s\} \uplus E;\ Q;\ \sigma \Longrightarrow \{Y^T : (c\ a) \cdot t \triangleq (c\ b) \cdot s\} \cup E;\ Q;\ \sigma\{X^S \mapsto \lambda c.Y^T\},$$

where $c \in \mathbb{A} \setminus S$ and Y^T is a fresh variable of sort $T = \mathtt{fa}((c\ a) \cdot t, (c\ b) \cdot s)$.[2]

Sol: **Solving**

$$\{X^S : t \triangleq s\} \uplus E;\ Q;\ \sigma \Longrightarrow E;\ Q \cup \{X^S : t \triangleq s\};\ \sigma,$$

if none of the previous rules is applicable.

Mer: **Merging**

$$E;\ \{X^S : t_1 \triangleq s_1, Y^T : t_2 \triangleq s_2\} \uplus Q;\ \sigma \Longrightarrow E;\ \{X^S : t_1 \triangleq s_1\} \cup Q;\ \sigma\rho,$$

where ρ is a substitution defined by $\{Y^T \mapsto \pi \cdot X^S\}$ and π is a permutation such that $\pi \cdot t_1 =_\alpha t_2$ and $\pi \cdot s_1 =_\alpha s_2$.[3]

Figure 2: Transformation rules of the anti-unification algorithm.

[2] Since $\mathbb{A} \setminus S$ isn't empty and $\mathtt{fa}(\lambda a.t, \lambda b.s) \subseteq S$, we can take $c \in \mathbb{A} \setminus S$ to rename the bound atoms (Figure 1).

[3] Since the sort of generalization variables is minimal w.r.t. the represented terms, ρ always exists if π exists.

Computing The Lgg. Given two terms t and s, $\text{NAU}_\mathcal{P}$ works in the following manner:

1. Create the initial state $\{X^{\mathtt{fa}(t,s)} : t \triangleq s\};\ \emptyset;\ id$.
2. Apply the rules of Figure 2 exhaustively, that is $\{X^{\mathtt{fa}(t,s)} : t \triangleq s\};\ \emptyset;\ id \Longrightarrow^* \emptyset;\ Q;\ \sigma$.
3. Apply the computed substitution σ to the generalization variable of the initial state $X^{\mathtt{fa}(t,s)}$, that is $X^{\mathtt{fa}(t,s)}\sigma$, to obtain the generalization of the input terms t and s.

We write $\text{NAU}_\mathcal{P}(t,s)$ to denote the result of that process, i.e., it denotes the generalization.

Since, from the computed substitution only one mapping is needed to obtain the generalization, we will omit all the other mappings in the following derivation examples.

Example 4. Consider the input terms $f(a,b,a)$ and $f(b,a,c)$. The initial state is $\{X^{\{a,b,c\}} : f(a,b,a) \triangleq f(b,a,c)\};\ \emptyset;\ id$. Now we apply the rules of Figure 2 exhaustively.

$$\begin{aligned}
&\{X^{\{a,b,c\}} : f(a,b,a) \triangleq f(b,a,c)\};\ \emptyset;\ id && \Longrightarrow_{\text{Dec}}\\
&\{Y_1^{\{a,b\}} : a \triangleq b,\ Y_2^{\{a,b\}} : b \triangleq a,\ Y_3^{\{a,c\}} : a \triangleq c\};\ \emptyset;\ \{X^{\{a,b,c\}} \mapsto f(Y_1^{\{a,b\}}, Y_2^{\{a,b\}}, Y_3^{\{a,c\}})\} && \Longrightarrow^3_{\text{Sol}}\\
&\emptyset;\ \{Y_1^{\{a,b\}} : a \triangleq b,\ Y_2^{\{a,b\}} : b \triangleq a,\ Y_3^{\{a,c\}} : a \triangleq c\};\ \{X^{\{a,b,c\}} \mapsto f(Y_1^{\{a,b\}}, Y_2^{\{a,b\}}, Y_3^{\{a,c\}})\} && \Longrightarrow_{\text{Mer}}\\
&\emptyset;\ \{Y_1^{\{a,b\}} : a \triangleq b,\ Y_3^{\{a,c\}} : a \triangleq c\};\ \{X^{\{a,b,c\}} \mapsto f(Y_1^{\{a,b\}}, (a\ b) \cdot Y_1^{\{a,b\}}, Y_3^{\{a,c\}})\}
\end{aligned}$$

The computed generalization is $f(Y_1^{\{a,b\}}, (a\ b) \cdot Y_1^{\{a,b\}}, Y_3^{\{a,c\}})$. Applying the substitution $\{Y_1^{\{a,b\}} \mapsto a, Y_3^{\{a,c\}} \mapsto a\}$ gives $f(a,b,a)$, and applying $\{Y_1^{\{a,b\}} \mapsto b, Y_3^{\{a,c\}} \mapsto c\}$ gives $f(b,a,c)$.

Example 5. Consider the input terms $f(\lambda b.b, a)$ and $f(\lambda a.X^S, X^S)$ where $S \subseteq \mathbb{A}^<$ and $a \in S$. The initial state is $\{Y^S : f(\lambda b.b, a) \triangleq f(\lambda a.X^S, X^S)\};\ \emptyset;\ id$.

$$\begin{aligned}
&\{Y^S : f(\lambda b.b, a) \triangleq f(\lambda a.X^S, X^S)\};\ \emptyset;\ id && \Longrightarrow_{\text{Dec}}\\
&\{Z_1^{S\setminus\{a\}} : \lambda b.b \triangleq \lambda a.X^S,\ Z_2^S : a \triangleq X^S\};\ \emptyset;\ \{Y^S \mapsto f(Z_1^{S\setminus\{a\}}, Z_2^S)\} && \Longrightarrow_{\text{Abs}}\\
&\{Z_3^{(S\setminus\{a\})\cup\{c\}} : c \triangleq (a\ c) \cdot X^S,\ Z_2^S : a \triangleq X^S\};\ \emptyset;\ \{Y^S \mapsto f(\lambda c.Z_3^{(S\setminus\{a\})\cup\{c\}}, Z_2^S)\} && \Longrightarrow^2_{\text{Sol}}\\
&\emptyset;\ \{Z_3^{(S\setminus\{a\})\cup\{c\}} : c \triangleq (a\ c) \cdot X^S,\ Z_2^S : a \triangleq X^S\};\ \{Y^S \mapsto f(\lambda c.Z_3^{(S\setminus\{a\})\cup\{c\}}, Z_2^S)\} && \Longrightarrow_{\text{Mer}}\\
&\emptyset;\ \{Z_2^S : a \triangleq X^S\};\ \{Y^S \mapsto f(\lambda c.(a\ c) \cdot Z_2^S, Z_2^S)\}
\end{aligned}$$

The computed result is $f(\lambda c.(a\ c) \cdot Z_2^S, Z_2^S)$. It is equi-general to $f(\lambda a.X^S, X^S)$, and, instantiating it by $\{Z_2^S \mapsto a\}$ results in $f(\lambda c.c, a)$ which is α-equivalent to $f(\lambda b.b, a)$.

Note that in the merge step of Example 5, we could have chosen to keep $Z_3^{(S\setminus\{a\})\cup\{c\}}$ instead of Z_2^S. There might be various possible rule applications to a certain state but the choice doesn't matter. The derivations are confluent and lead to equi-general results.

5 Properties of $\text{NAU}_\mathcal{P}$

Lemma 1 (Termination)**.** $\text{NAU}_\mathcal{P}$ generates $\mathcal{O}(n)$ states for any input of size n and terminates.

Proof. The size $\|t\|$ of a term t is defined as $\|a\| = 1,\ \|f(t_1,\ldots,t_n)\| = 1+\sum_{i=1}^n \|t_i\|,\ \|\lambda a.s\| = 1 + \|s\|$, and $\|\pi \cdot X^S\| = 1$. The size of a set of anti-unification equations E is defined as $\|E\| = \sum_{X^S : t \triangleq s \in E} \|t\| + \|s\|$. Finally, the size of a state $E;\ Q;\ \sigma$ is defined by $2\|E\| + \|Q\|$.

Using that definition, we get that the initial state created by $\text{NAU}_\mathcal{P}(t,s)$ is of size $2(\|t\| + \|s\|)$, i.e., linear by the size of the input terms t and s. Since every rule application strictly decreases the size of the state, $\text{NAU}_\mathcal{P}$ generates at most $\mathcal{O}(n)$ states until no more rule is applicable, for any input of size n. □

Lemma 2 (Soundness)**.** Given terms t, s. Any term $\mathrm{NAU}_{\mathcal{P}}(t, s)$ is a generalization of t and s.

Proof. Given two input terms t and s, $\mathrm{NAU}_{\mathcal{P}}(t, s)$ creates an initial state E; Q; σ where $E = \{X^{\mathtt{fa}(t,s)} : t \triangleq s\}$, $Q = \emptyset$, and $\sigma = id$. It trivially holds that $X^{\mathtt{fa}(t,s)}\sigma$ is a generalization of t and s. By induction on the derivation process we will show that this is actually an invariant that is maintained by rule applications. More precisely, we show that, after any rule application E'; Q'; $\sigma' \Longrightarrow E''$; Q''; σ'', the term $X^{\mathtt{fa}(t,s)}\sigma''$ is a generalization of t and s, given that $X^{\mathtt{fa}(t,s)}\sigma'$ is a generalization of t and s. In order to prove that, two substitutions, that can be obtained from an arbitrary set of anti-unification equations F, are needed:

$$\rho_l^F := \{X^S \mapsto t \mid X^S : t \triangleq s \in F\} \qquad \rho_r^F := \{X^S \mapsto s \mid X^S : t \triangleq s \in F\}$$

For the initial state, it is trivial that $X^{\mathtt{fa}(t,s)}\sigma\rho_l^{E\cup Q} =_\alpha t$ and $X^{\mathtt{fa}(t,s)}\sigma\rho_r^{E\cup Q} =_\alpha s$. By case distinction on the rules of Figure 2 it can also easily be verified that $X^{\mathtt{fa}(t,s)}\sigma''\rho_l^{E''\cup Q''} =_\alpha t$ and $X^{\mathtt{fa}(t,s)}\sigma''\rho_r^{E''\cup Q''} =_\alpha s$ holds, given that $X^{\mathtt{fa}(t,s)}\sigma'\rho_l^{E'\cup Q'} =_\alpha t$ and $X^{\mathtt{fa}(t,s)}\sigma'\rho_r^{E'\cup Q'} =_\alpha s$. □

Lemma 3 (Completeness)**.** For any generalization r of some terms t, s holds $r \preceq \mathrm{NAU}_{\mathcal{P}}(t, s)$.

Proof. By structural induction on r we identify common parts of t and s. 4 possible cases are given by the term grammar. In the two cases where we encounter either an atom or a function application in r, it is easy to conclude that one of the rules Atm or Dec is applicable. Therefore, the same symbol will also appear in the substitution of the transformed state of $\mathrm{NAU}_{\mathcal{P}}$.

The third case treats abstractions in r. I.e., sub-terms like $\lambda a.t'$ and $\lambda b.s'$ in t, s. It corresponds to the case where Abs applies. Abs performs α-renaming and generalizes the abstraction.

The last case considers the appearance of a suspension $\pi \cdot X^S$ in r. It represents sub-terms t', s' of t and s, respectively. $\mathrm{NAU}_{\mathcal{P}}$ keeps the sort of generalization variables minimal, always. For t', s' it is $\mathtt{fa}(t', s')$. Therefore $\pi \cdot X^S$ is more general than the generalization variable used by $\mathrm{NAU}_{\mathcal{P}}$ to represent t' and s'. The rule Mer ensures that variables are shared, whenever possible. It follows that $\mathrm{NAU}_{\mathcal{P}}(t, s)$ is an lgg of t and s. □

6 Conclusion

The language of permissive nominal terms introduced by Dowek et al. exhibits the same problem, w.r.t. the generalization type, as classical nominal terms. In order to address that issue, we suggest a modification of the definition of permission sets. This leads to a term language that is a superset of the one introduced by Dowek et al.

Our work might be seen as a starting point for a broader revision of the (permissive) nominal setting. The modified definition probably leads to implications w.r.t. unification, computational complexity, and so on. It might be practical to restrict the permission sets so that they have a simple and finite representation. An interesting restriction could be to only consider finite permission sets and the original ones from Dowek et al., i.e., sets of the form $\{a_1, \ldots, a_n\}$ where $a_i \in \mathbb{A}$ and the ones in $\mathcal{P}_D$. Those are exactly the ones needed to get a well-founded $\preceq$ relation. Note that $\mathrm{NAU}_{\mathcal{P}}$ computes generalizations within that restricted setting if the input terms satisfy the restriction, e.g., if the input is from $\mathcal{P}_D$.

Due to the well-founded $\preceq$ relation, our setting might be closer to higher-order pattern [16] (HOP) than other nominal settings [14]. Note that HOP have a well-founded $\preceq$ relation too [3].

Questions like, implications of our setting w.r.t. the results from [9], computational complexity of $\mathrm{NAU}_{\mathcal{P}}$, connections to other languages (e.g., HOP), etc. remain open. They are possible directions of future work that continues to investigate on the proposed setting, with and without imposing restrictions on the permission sets.

References

[1] Adam D. Barwell, Christopher Brown, and Kevin Hammond. Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification. *Future Generation Comp. Syst.*, 79:669–686, 2018.

[2] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal anti-unification. In Maribel Fernández, editor, *26th International Conference on Rewriting Techniques and Applications, RTA 2015, June 29 to July 1, Warsaw, Poland*, volume 36 of *LIPIcs*, pages 57–73. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[3] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-order pattern anti-unification in linear time. *J. Autom. Reasoning*, 58(2):293–310, 2017.

[4] Alexander Baumgartner and Daniele Nantes-Sobrinho. A, C, and AC nominal anti-unification. In Temur Kutsia and Andrew M. Marshall, editors, *Proceedings of the 34th International Workshop on Unification, UNIF 2020, Linz, Austria, June 29, 2020*, pages 5:1–5:6, 2020.

[5] Petr Bulychev and Marius Minea. An evaluation of duplicate code detection using anti-unification. In *Proc. 3rd International Workshop on Software Clones*, 2009.

[6] David M. Cerna and Temur Kutsia. A generic framework for higher-order generalizations. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 10:1–10:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[7] David M. Cerna and Temur Kutsia. Higher-order pattern generalization modulo equational theories. *Math. Struct. Comput. Sci.*, 30(6):627–663, 2020.

[8] David M. Cerna and Temur Kutsia. Anti-unification and generalization: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 6563–6573. ijcai.org, 2023.

[9] Gilles Dowek, Murdoch Gabbay, and Dominic Mulligan. Permissive nominal terms and their unification: An infinite, co-infinite approach to nominal techniques. *Logic Journal of the IGPL*, 18:769–822, 10 2010.

[10] Cao Feng and Stephen Muggleton. Towards inductive generalisation in higher order logic. In Derek Sleeman and Peter Edwards, editors, *Machine Learning Proceedings 1992*, pages 154–162. Morgan Kaufmann, San Francisco (CA), 1992.

[11] Maribel Fernández and Murdoch Gabbay. Nominal rewriting. *Inf. Comput.*, 205(6):917–965, 2007.

[12] Warren D Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[13] 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.

[14] Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. In Andrei Voronkov, editor, *Rewriting Techniques and Applications*, pages 246–260, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[15] Claudio L Lucchesi. The undecidability of the unification problem for third order languages. *Report CSRR*, 2059:129–198, 1972.

[16] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.

[17] Frank Pfenning. Unification and anti-unification in the calculus of constructions. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 74–85. IEEE Computer Society, 1991.

[18] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.

[19] Manfred Schmidt-Schauß and Daniele Nantes-Sobrinho. Nominal anti-unification with atom-variables. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel*, volume 228 of *LIPIcs*, pages 7:1–7:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[20] Christian Urban, Andrew M Pitts, and Murdoch J Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1-3):473–497, 2004.