Nominal Anti-Unification with Atom-Variables

Manfred Schmidt-Schauß
Goethe Universität, Frankfurt am Main, Germany

Daniele Nantes-Sobrinho
Department of Computing, Imperial College London, UK
Department of Mathematics, University of Brasília, Brazil

Abstract

Anti-unification is the task of generalizing a set of expressions in the most specific way. It was extended to the nominal framework by Baumgarter, Kutsia, Levy and Villaret, who defined an algorithm solving the nominal anti-unification problem, which runs in polynomial time. Unfortunately, when an infinite set of atoms are allowed in generalizations, a minimal complete set of solutions in nominal anti-unification does not exist, in general. In this paper, we present a more general approach to nominal anti-unification that uses *atom-variables* instead of explicit atoms, and two variants of freshness constraints: $\mathtt{NL}_A$-constraints (with atom-variables), and EQR-constraints based on EQuivalence Relations on atom-variables. The idea of atom-variables is that different atom-variables may be instantiated with identical or different atoms. Albeit simple, this freedom in the formulation increases its application potential: we provide an algorithm that is finitary for the $\mathtt{NL}_A$-freshness constraints, and for EQR-freshness constraints it computes a unique least general generalization. There is a price to pay in the general case: checking freshness constraints and other related logical questions will require exponential time. The setting of Baumgartner et al. is improved by the atom-only case, which runs in polynomial time and computes a unique least general generalization.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases Generalization, anti-unification, nominal algorithms, higher-order deduction

Digital Object Identifier 10.4230/LIPIcs.FSCD.2022.7

Funding *Daniele Nantes-Sobrinho*: partially funded by the EPSRC Fellowship 'VeTSpec: Verified Trustworthy Software Specification' (EP/R034567/1) and Edital DPI/DPG n. 03/2020.

1 Introduction

Anti-unification is the task of, given a set of expressions, to find a most specific (or least general) generalization of all the expressions in this set. In the first-order version, anti-unification simply looks for the largest common term structure and also takes care of equal variables. In this case, the problem can be solved in polynomial time and produces a unique solution [1]. A simple example, taken from Plotkin [21], is the expression $P(g(x), x)$ that generalizes the set $\{P(g(a), a), P(g(b), b)\}$. Notice that the expressions z, $P(y, x)$ and $P(g(y), x)$ also generalize the expressions in the set, however, they are not least general: there exist substitutions σ_i, such that $P(y, x)\sigma_1 = P(g(x), x)$, $P(g(y), x)\sigma_2 = P(g(x), x)$ and $z\sigma_3 = P(g(x), x)$, but not vice versa.

In this paper we are interested in a more complex variation of this problem in the context of a nominal language [11], which is a convenient alternative for expressing languages with binders with the benefit that nominal unification is decidable in quadratic time and unitary [25, 6, 18]. Thus, this work develops around the *nominal anti-unification problem*, i.e., the problem of finding a least general generalization of nominal expressions. Similarly to the relation between nominal unification and higher-order pattern unification [17], nominal anti-unification relates with higher-order pattern anti-unification, thus, developments in nominal

anti-unification promote new insights into the tractability and applicability of higher-order anti-unification problems, and consequently, it has a high potential for applications such as recursion scheme and clone detection [2, 3], learning with counter examples [16], etc.

The nominal anti-unification problem considered by Baumgartner et al. in [4] does not have a least general generalization (lgg), not even a minimal complete set of generalizers can be computed (i.e. it is nullary). A simple example illustrates the infinite set of generalizations: for $f(a_1)$ and $g(a_2)$, the generalization $(\emptyset, X)$ is appropriate, where the pair consists of an empty set of freshness constraints and the generalization variable X. However, there is a strictly decreasing chain $(\emptyset, X)$, $(\{a_3\#X\}, X)$,$(\{a_3\#X, a_4\#X\}, X), \ldots$ (where $a_i\#X$ means that atom a_i is fresh in the instances of X). These are more and more strictly specific generalizations, for an infinite set of atoms $\{a_3, a_4, \ldots\}$. The names $a_3, a_4, \ldots$ are irrelevant for the problem, but provide an argument that a *least* general generalization might not exist, not even a minimal complete set of lggs. Restricting the set of available atoms to a finite set as in [4] results in nice properties of the algorithm. Although it may suffice in practice, it is not satisfactory.

Our approach is to employ *atom-variables* [24] in the grammar of nominal expressions, which are intended to represent atoms, and formulate the anti-unification problem in this extended nominal language. Atoms are only used in the semantics, whereas in the expression language, only atom-variables are used. Therefore, basic nominal syntactic notions, such as permutations, suspended variables and abstractions, are now generalized to contain both atom-variables $(A, B, C, \ldots)$ and generalization variables $(X, Y, \ldots)$. For instance, abstractions such as $\lambda C.f(C, X)$ and $\lambda(A\ C) \cdot B.f(C, X)$, suspended atom and generalization variables, such as $(A\ B) \cdot C$ and $(A\ B) \cdot X$, are allowed, and their meaning relies on the instantiation of such atom-variables by concrete atoms: mapping atom-variables A, B, C to concrete different atoms a, b, c, respectively, $(A\ C) \cdot B$ reduces to $(a\ c) \cdot b = b$ (the permutation $(a\ c)$ has no effect on b) and the abstraction $\lambda(A\ C) \cdot B.f(C, X)$ becomes $\lambda b.f(c, X)$. Another important feature is that alpha-equivalence ($\sim$) is defined semantically: deciding whether $(A\ B) \cdot C \sim (A\ C) \cdot B$ relies on the instantiation of the atoms A, B and C. In the case where B and C are mapped to the same atom, the equivalence holds, but it fails if A, B, C are mapped to different atoms.

Standard freshness constraints in our language, called $\mathtt{NL}_A$-constraints, include not only constraints of the form $A\#X$, but also of the forms $A\#B$, $B\#\lambda A.B$ etc. As expected, the meaning of these constraints depends on the instantiation of the atom-variables occurring in them. For instance, $A\#X$ means that instances of A cannot occur free in instances of X, and $A\#B$ means that instances of A are different from the instances of B. The expressivity gain is nicely observed with $C\#\lambda A.\lambda B.C$, which means that the instantiation of C is the same as that of A or of B. However, this is still not sufficient: For representing generalizations in a more expressive way we introduce the EQR-constraints, which permit, for every equivalence class of instantiations, an extra set of constraints. For example, $((A = B) \implies A\#X) \wedge ((A \neq B) \implies B\#X)$ with the informal meaning: if A, B are equally instantiated, then $A\#X$ must be satisfied, otherwise $B\#X$. This increase permits to have a smaller number of least general generalizations.

Moreover, the standard notion of a nominal expression being *more general* than another (which is relevant for generalizations) when atom-variables are involved is defined by a more complex requirement than just one being an instance of the other: one has to consider the instances of the atom-variables involved in both terms. Thus, the syntax of the terms is more powerful, which means that the syntax of the generalizations is more powerful, and this has to be taken into account by the search for a most specific generalization. This means

that these "extended" freshness constraints and permutations with atom-variables, which are now part of the syntax of nominal terms, also may appear in generalizations, which increases the representative power of this approach.

Contributions. We construct a sound and complete rule-based algorithm AtomAntiUnif for computing generalizations (see Section 3), which performs in exponential time, and relies on the subalgorithm Eqvm for equivariant matching with atom-variables. Our rules are inspired by [4], modified with semantic checks for equalities and disequalities of atom-variables, and extra rules dealing with suspended atom/generalization variables. Even though our algorithm AtomAntiUnif computes a good generalization, it is not least general, as it is shown in Example 3.4. Our approach significantly improves previous ones: there are no infinite chains of generalizations (Corollary 3.12). Thus, the unification type of generalization problems with atom-variables is finitary.

$\mathtt{NL}_A$- freshness constraints should be investigated with care: a more expressive form of freshness constraints based on EQuivalence Relations over atom-variables has to be considered. The Eqr-freshness constraints (Definition 4.2) are a more powerful version of freshness constraints that explicitly represent all possible in-/equality patterns of instantiations of atom-variables. To obtain a singleton solution set, a second algorithm AtomAntiUnifLgg refines the constraint part of the generalization computed by AtomAntiUnif, and returns a least general generalization (Definition 4.10). Finally, we conclude that the anti-unification problem with atom-variables and Eqr-freshness constraints is decidable and unitary (Theorem 4.13). With the restriction that different atom-variables can only be instantiated with different atoms (called the atoms-only case), we obtain a specialization of our algorithm that computes a singleton complete set of lggs in polynomial time for our variant of the nominal anti-unification problem investigated in [4] (see Theorem 4.16). This corresponds to the problem in [4] with infinitely many atoms, but using a more flexible semantics. The power of our semantics is illustrated in Example 2.14.

Related Work. Early works on generalization date back to the 70s, a well-known note is presented by Plotkin [21], who discussed the usefulness of generalization when looking for methods of induction. More recent works such as [7] and [8] investigate higher-order pattern anti-unification, the latter is an extension with equational theories. Applications of anti-unification were exploited in several directions, such as finding parallel recursion schemes [2], in program analysis [5], for analogy (or clone) detection [3, 14, 19], in description logics [13], checking inductive properties of term rewriting systems [10], among others. A library of anti-unification algorithms is available in [3].

Organization. Section 2 gives the necessary background on the nominal language $\mathtt{NL}_A$ with atom-variables and introduces the nominal anti-unification problem with atom-variables following a semantic approach. Section 3 contains the rules for the AtomAntiUnif algorithm as well as arguments for its soundness, completeness and run-time complexity. Section 4 introduces a more expressive version of freshness constraints with atom-variables, and the AtomAntiUnifLgg algorithm that refines generalizations output by AtomAntiUnif, and computes a unique lgg of two $\mathtt{NL}_A$ terms-in-context. Proofs are available in the appendix.

2 Preliminaries

The nominal language $\mathtt{NL}_{aX}$ of expressions[1] is built by the grammar

$$S ::= a \mid f(S_1, \ldots, S_n) \mid \lambda a.S \mid \pi{\cdot}X \qquad\qquad \pi ::= \emptyset \mid (a\ b){\cdot}\pi$$

where a, b are atoms in the infinite set *Atoms*, π is a nonterminal for permutations, $(a\ b)$ is a swapping, X is a nonterminal for generalization variables, f, g are function symbols and c, c_i are constant symbols in the function signature $\mathcal{F}$, where we assume that there is at least one (say c) of arity zero and one of arity 2. Compound expressions are function applications, and lambda-expressions which bind atoms. We also permit the tuple notation in examples. Applications in the lambda calculus can be represented using a binary function symbol.

The *ground language* $\mathtt{NL}_a$ is a sublanguage of $\mathtt{NL}_{aX}$ where variables X and permutations π are omitted. We consider only α-equivalence $\sim$, which in fact is only defined on $\mathtt{NL}_a$.

An $\mathtt{NL}_{aX}$*-freshness constraint* is an expression of the form $a\#S$, expressing that a is not free in (fresh for) S. We permit also $\bot$ as freshness constraint, which represents `False`. An $\mathtt{NL}_{aX}$*-freshness context* $(\nabla, \Delta, \ldots)$ is a set of $\mathtt{NL}_{aX}$-freshness constraints. We assume that permutation applications are homomorphically shifted inside expressions, where $\pi{\cdot}a$ is immediately computed. Every $\mathtt{NL}_{aX}$-freshness context can be transformed into a simpler one consisting only of constraints of the form $a\#X$ or $\bot$ by exhaustively using the rules:

$$\frac{a\#f(S_1,\ldots,S_n)}{a\#S_1,\ldots,a\#S_n} \qquad \frac{a\#b}{} \qquad \frac{a\#a}{\bot} \qquad \frac{a\#\lambda b.S}{a\#S} \qquad \frac{a\#\lambda a.S}{} \qquad \frac{a\#\pi{\cdot}X}{\pi{\cdot}a\#X}$$

An $\mathtt{NL}_{aX}$-freshness context ∇ is in *flattened form*, denoted by $\langle\nabla\rangle_{ff}$, when ∇ is decomposed using the rules above, and permutations are eliminated such that only $a\#X$ and $\bot$ remain. A $\mathtt{NL}_{aX}$-freshness context is *consistent* if its flattened form does not contain $\bot$.

▶ **Definition 2.1** (Explanation of $\vDash$). *Let ∇ be a consistent $\mathtt{NL}_{aX}$-freshness context, and $a\#S$ be a constraint. Then $\nabla \vDash a\#S$* holds *iff $\langle\{a\#S\}\rangle_{ff} \subseteq \langle\nabla\rangle_{ff}$.*

An $\mathtt{NL}_{aX}$*-substitution* ρ is a finite mapping from generalization variables to $\mathtt{NL}_{aX}$-expressions, extended to expressions. We will denote the domain of substitutions by $dom(.)$. A substitution is *ground* if it maps variables to $\mathtt{NL}_a$-expressions. For a ground substitution ρ: $\nabla\rho$ is called *valid* iff $\langle\nabla\rho\rangle_{ff}$ is consistent.

▶ **Lemma 2.2.** *Let ∇ be an $\mathtt{NL}_{aX}$-freshness context, and $a\#S$ be an $\mathtt{NL}_{aX}$-freshness constraint. Then, $\nabla \vDash a\#S$ iff for all ground substitutions ρ: if $\nabla\rho$ is valid, then also $(\{a\#S\})\rho$ is valid.*

2.1 Nominal language $\mathtt{NL}_A$ with Atom-Variables

Let *AtomVars* be a set of atom-variables ranging over $A, B, A_1, B_1, \ldots$. The $\mathtt{NL}_A$-expression language is related to the nominal languages $\mathtt{NL}_a$ and $\mathtt{NL}_{aX}$, but includes *atom-variables* instead of atoms (see [24]). The grammar for the expression language $\mathtt{NL}_A$ of expressions s and permutations π with atom-variables is as follows:

$$s ::= W \mid \pi{\cdot}X \mid f(s_1, \ldots, s_n) \mid \lambda W.s \qquad W ::= \pi{\cdot}A \qquad \pi ::= \emptyset \mid (W_1\ W_2){\circ}\pi$$

Note that $\mathtt{NL}_A$-expressions do not contain atoms. There are two kinds of suspensions: of atom-variables, as in $\pi{\cdot}A$, and of generalization variables, as in $\pi{\cdot}X$. Composition of (atom-variable) permutations π_1 and π_2 is denoted $\pi_1 \circ \pi_2$ and can be flattened by concatenating

[1] The nominal language is equivalent to other nominal languages with different binding constructs.

the atom-variable swappings in π_1, π_2. For a permutation $\pi = \pi_1 \circ \ldots \circ \pi_n$, we write π^{-1} for the inverse of π, i.e., for the permutation $\pi_n^{-1} \circ \ldots \circ \pi_1^{-1}$. Note that $\pi^{-1} = \pi$ holds for swappings π.

Substitutions are extended by a mapping of atom-variables.

The following notation will be used: $Head(s)$ is defined as f, if $s = f(\ldots)$; and λ, if $s = \lambda a.s'$; if s is a suspension $\pi{\cdot}X$, then X; and if s is $\pi{\cdot}A$ then A. We use the usual conventions for dealing with permutations and suspensions, for example, to move permutations homomorphically inside terms and viewing $\emptyset{\cdot}s$ as the same term as s. Suspensions of atom variables have to be treated carefully, for instance, $(A\ B) \cdot A = B$, for all instantiations of A and B to atoms, but $(A\ C) \cdot B$ is not necessarily B, since B, C could be mapped to the same atom, resulting in A. We also will use the abbreviations AV for atom-variables and GV for generalization-variables.

▶ **Example 2.3.** Consider the $\mathtt{NL}_A$-expression $s = \lambda(A\ B){\cdot}C.f(C, X)$: it is an abstraction of the suspended atom-variable $(A\ B){\cdot}C$ on the expression $f(C, X)$. Let $\sigma = \{C \mapsto A, X \mapsto B\}$ be an $\mathtt{NL}_A$ substitution. Then, $s\sigma = \lambda(A\ B){\cdot}A.f(A, B) = \lambda B.f(A, B)$. For the ground substitution (i.e., mapping to $\mathtt{NL}_a$), $\rho = \{A \mapsto a_1, B \mapsto a_2, C \mapsto a_3, X \mapsto a_3\}$, we have: $s\rho = \lambda a_3.f(a_3, a_3)$.

Notice that the substitution σ is applied to s in a "capturing" way, as usual in nominal terms and nominal unification.

▶ **Definition 2.4.** $\mathtt{NL}_A$-freshness constraints *are pairs of the form $A\#s$ where A is an atom-variable and s is an $\mathtt{NL}_A$-expression.* $\mathtt{NL}_A$-freshness contexts *$(\nabla, \Delta, \ldots)$ are finite sets (conjunctions) of freshness constraints.*

In the following we permit $\pi{\cdot}A\#s$, but with the convention to replace $\pi{\cdot}A\#s$ immediately by $A\#\pi^{-1}{\cdot}s$, and also to move the permutation π^{-1} inside the expression s. This move is done homomorphically, and stops at suspensions as follows: $\pi{\cdot}(\pi'{\cdot}V) \mapsto (\pi \circ \pi'){\cdot}V$.

▶ **Example 2.5.** It is possible to represent equality and inequality of atom-variables using $\mathtt{NL}_A$-freshness constraints: the constraint $A\#B$ means that instantiations of A, B must be different. The constraint $A\#\lambda B.A$ enforces that the instantiations of A, B must be equal. It is also possible to represent (a restricted form) of propositional formulas over equations on atom-variables, e.g., $(C = A) \vee (C = B)$ can be represented as $C\#\lambda A.\lambda B.C$.

$\mathtt{NL}_A$-freshness constraints and contexts can be further standardized/decomposed in the right hand side until they are of one of the following forms: $A\#\lambda W_1.\cdots.\lambda W_n.W$, or $A\#\lambda W_1.\cdots.\lambda W_n.\pi{\cdot}X$. Notice that $A\#A$ is inconsistent, but more complex constraints cannot be decomposed or evaluated without information about the concrete (i.e., ground) instances of the atom-variables. In the following we sometimes write freshness constraints/-contexts to mean $\mathtt{NL}_A$-freshness constraints/context.

▶ **Definition 2.6.** *A $\mathtt{NL}_A$-freshness constraint $A\#s$ is* valid *for a ground substitution ρ, iff $A\rho\#s\rho$ is valid in $\mathtt{NL}_a$. A freshness context ∇ is* valid *for a ground substitution ρ, iff for every constraint $A\#s \in \nabla$, $(A\#s)\rho$ is valid. A freshness context ∇ is* consistent*, iff there is a ground substitution ρ, such that $\nabla\rho$ is valid.*

▶ **Definition 2.7.** *Let ∇ be an $\mathtt{NL}_A$-freshness context and $A\#s$ be a freshness constraint, and π_1, π_2 be permutations. Then*

- *$\nabla \vDash A\#s$ holds, iff for all ground substitutions ρ and consistent $\nabla\rho$, $\nabla\rho \vDash (A\rho\#s\rho)$ holds.*
- *$\nabla \vDash \pi_1 = \pi_2$ holds, iff for all ground substitutions ρ such that $\nabla\rho$ is consistent, $\nabla\rho \vDash \pi_1\rho = \pi_2\rho$ holds (as functions).*

We will define a more operational approach for deciding implication of $\mathtt{NL}_A$-freshness contexts and constraints, where the basic idea is to make a case analysis of all equal/inequal possibilities of atom-variables. We denote the set of atom-variables occurring in ∇ as $\mathtt{AtVar}(\nabla)$ and write $\mathtt{AtVar}(\nabla, A\#s)$ for $\mathtt{AtVar}(\nabla \cup \{A\#s\})$. Also, $\mathtt{GenVar}(o)$ denotes the set of generalization variables of the object o.

▶ **Definition 2.8.** *Let ∇ be a freshness context, and R be an equivalence relation on $\mathtt{AtVar}(\nabla)$. An* R-realization function *ρ_R is a function $\rho_R : \mathtt{AtVar}(\nabla) \to$ Atoms, mapping every atom-variable A in ∇ to a concrete atom, such that $A_1 \sim_R A_2$ iff $\rho_R(A_1) = \rho_R(A_2)$.*

We assume that ρ_R can be applied to substitutions by homomorphically applying it to the components. In an application $\rho_R(s)$, permutations are applied such that the result is an atom or a suspension $\pi{\cdot}X$ of a (generalization) variable X where π only contains atoms.

▶ **Definition 2.9.** *Let ∇ be a freshness context and $A\#s$ be a freshness constraint. Then,* $\nabla \vdash_{ER} A\#s$ holds *iff for all equivalence relations R on $\mathtt{AtVar}(\nabla, A\#s)$: $\rho_R(\nabla) \vDash \rho_R(A\#s)$.*

▶ **Lemma 2.10.** *Let ∇ be a freshness context, and $A\#s$ be a freshness constraint. Then $\nabla \vDash A\#s$ iff $\nabla \vdash_{ER} A\#s$.*

▶ **Proposition 2.11.** *Let ∇, Δ be freshness contexts. Then the complexity of the problem $\nabla \vDash \Delta$ is in coNP.*

Proof. We analyze the algorithm $\vdash_{ER}$ in Definition 2.9. We have to check for all equivalence-relations of atom-variables whether the relation holds. Since these equivalence-relations can be represented in polynomial size, and the test $\vdash$ is polynomial once the equivalence relation is chosen, we see that the problem is in coNP (see for example [15]). ◀

Alpha-equivalence on $\mathtt{NL}_A$ can be established semantically only: deciding whether all instances of, for example, $\lambda A.A$ and $\lambda B.(B\ C){\cdot}C$ are alpha-equivalent, intuitively takes us (using the usual nominal techniques) to checking whether $A \sim (B\ C){\cdot}C$ and $A\#C$, but the answer relies on all possible instantiations of A, B and C.

▶ **Definition 2.12.** *An $\mathtt{NL}_A$-term-in-context is a pair (∇, s) of an $\mathtt{NL}_A$-freshness context ∇ and an $\mathtt{NL}_A$-expression s. The semantics of (∇, s) is the set of (equivalence classes of) ground instances of s that satisfy ∇, i.e., $[\![(\nabla, s)]\!] := \{[r]_\sim \mid r$ is ground and $\exists\sigma : s\sigma \sim r \wedge \nabla\sigma$ holds$\}$, where $[r]_\sim$ denotes the equivalence class of r modulo $\sim$.*

A term-in-context (∇, t) is more general *than another term-in-context (∇', t'), if $[\![(\nabla', t')]\!] \subseteq [\![(\nabla, t)]\!]$. Two terms-in-context (∇_1, t_1) and (∇_2, t_2) are* equivalent *iff $[\![(\nabla_1, t_1)]\!] = [\![(\nabla_2, t_2)]\!]$. We will also write the equivalence of (∇, t_1) and (∇, t_2) as $\nabla \vDash t_1 = t_2$.*

We recall an example from [4] which are two $\mathtt{NL}_{aX}$ terms-in-context $(\{a\#X\}, f(X))$ and $(\{a\#X\}, f(a))$, where it is shown that $(\{a\#X\}, f(X))$ is not more general than $(\{a\#X\}, f(a))$. This behaviour is improved in our approach: after transferring their example into $\mathtt{NL}_A$, we have that $(\{A\#X\}, f(X))$ is more general than $(\{A\#X\}, f(A))$. In $(\{A\#X\}, f(X))$, the ground instance of A can be chosen arbitrarily, thus $(\{A\#X\}, f(X))$ is equivalent to $(\emptyset, f(X))$.

▶ **Definition 2.13.** *A term-in-context (∇, r) is called a* generalization *of two terms-in-context (∇_1, s) and (∇_2, t), if $[\![(\nabla_1, s)]\!] \subseteq [\![(\nabla, r)]\!]$ and $[\![(\nabla_2, t)]\!] \subseteq [\![(\nabla, r)]\!]$. It is a* least general generalization *(lgg) of (∇_1, s) and (∇_2, t) if it is a smallest one, i.e., for all least general generalizations (∇', r') of (∇_1, s) and (∇_2, t), it holds $[\![(\nabla, r)]\!] \subseteq [\![(\nabla', r')]\!]$. A set G of generalizations is* complete *iff for all generalizations (∇', r') of (∇_1, s) and (∇_2, t), there is*

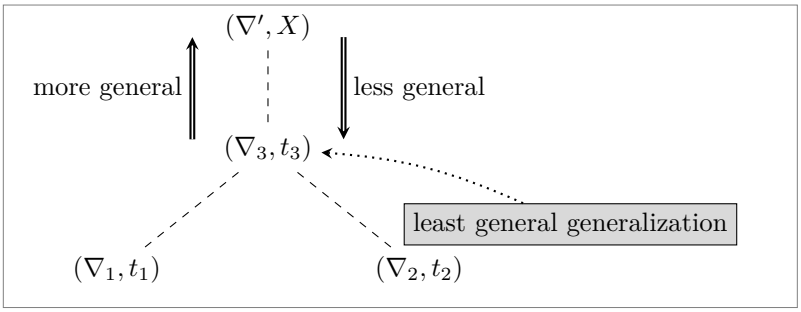
some $g \in G$, such that $[\![g]\!] \subseteq [\![(\nabla', r')]\!]$. The set G is minimal*, if it is non-redundant, i.e. for all different $g_1, g_2 \in G$, $[\![g_1]\!] \not\subseteq [\![g_2]\!]$.*

We call the generalization problem (of a language) unitary *if there always exists a single lgg. We call it* finitary *or* infinitary *resp., if there always exists a minimal (finite, or unrestricted, resp.) complete set of generalizations for any input problem. If there are input problems such that a minimal complete set does not exist, the problem is called* nullary*. The latter case means that for the particular input problem, all complete sets are redundant.*

▶ **Example 2.14.** We reconsider the example of [4] of an infinite chain of generalizations, and show that the corresponding example for atom-variables becomes finite due to our semantics. Consider the sequence of terms-in-context $(\{A\#X\}, f(X,A))$; $(\{A\#X, B_1\#X, B_1\#A\}, f(X,A))$; $(\{A\#X, B_1\#X, B_1\#A,\ B_2\#X, B_2\#A,\ B_2\#B_1\},\ f(X,A))$ etc. In contrast to the framework in [4], the translated chain is not a counterexample to our claim, since the semantics remains constant within the chain: We will argue that $[\![\{A\#X\}, f(X,A)]\!]$ coincides with the set $[\![\{A\#X, B_1\#X, B_1\#A\}, f(X,A)]\!]$. Obviously $[\![\{A\#X, B_1\#X, B_1\#A\}, f(X,A)]\!] \subseteq [\![\{A\#X\}, f(X,A)]\!]$. Hence it is sufficient to show the converse inclusion. Let ρ be such that $A\rho\#X\rho$. We simply have to show that the constraint $\{A\#X, B_1\#X, B_1\#A\}$ can be satisfied by defining ρ appropriately on B_1. Since there are infinitely many atoms, there is always an atom not in $X\rho$, and also different from $A\rho$, say b. We define $B_1\rho := b$, and then the constraint is satisfied, and $f(X,A)\rho$ is an element of $[\![\{A\#X, B_1\#X, B_1\#A\}, f(X,A)]\!]$. Thus, the semantics of both terms-in-contexts is the same. The method can be used for all other terms-in-context of the translated chain and show that the chain is in fact constant, since there are infinitely many atoms available.

2.2 Nominal Anti-Unification Problem with Atom-Variables

The goal is to find a least general generalization of sets of $\mathtt{NL}_A$ terms-in-context.
Problem: Given two $\mathtt{NL}_A$ terms-in-context (∇_1, t_1) and (∇_2, t_2).
Find: A $\mathtt{NL}_A$-term-in-context (∇_3, t_3) such that $[\![(\nabla_1, t_1)]\!] \subseteq [\![(\nabla_3, t_3)]\!]$ and $[\![(\nabla_2, t_2)]\!] \subseteq [\![(\nabla_3, t_3)]\!]$, and $[\![(\nabla_3, t_3)]\!]$ is as small as possible, i.e., (∇_3, t_3) is a lgg of (∇_1, t_1) and (∇_2, t_2).



As example, let $(\{A_1\#A_2\}, f(A_1, \lambda A_2.A_2))$ and $(\emptyset, f(c, \lambda A_3.A_3))$ be $\mathtt{NL}_A$ terms-in-context, where c is a unary symbol in the signature. Their least generalization is $(\emptyset, f(X, \lambda A_2.A_2))$, since the constraint does not restrict the instances. Another example for terms-in-context is $(\emptyset, f(A))$ and $(\emptyset, g(B))$. It is easy to check that $(\emptyset, X)$ is a generalization.

3 The Algorithm AtomAntiUnif

We first define the (non-deterministic) nominal generalization algorithm AtomAntiUnif that computes a generalization of the input terms-in-context. It relies on the subalgorithm Eqvm for equivariant matching (with atom-variables) that computes a permutation. We

prove that the algorithm is sound and complete, computes a generalization, and which can be performed in exponential time. Later in Section 4, we will define a method to compute a least general generalization from the computed generalization by adding a freshness context.

3.1 The Nominal Generalization Algorithm

The *state* of the algorithm AtomAntiUnif is a tuple $(\Gamma, M, \nabla, \theta)$ where

- Γ is a set of generalization triples of the form $X : s \triangleq t$, where X is a fresh (generalization-) variable, and s, t are $\mathtt{NL}_A$-expressions.
- M is a set of solved generalization triples.
- ∇ is a set of freshness constraints.
- θ is a substitution represented as a list of bindings; the empty list is denoted as $[]$.

The rules of the AtomAntiUnif, given in Figure 1, operate on such states. Given two $\mathtt{NL}_A$ expressions s and t, and a freshness context ∇ (possibly empty), to compute generalizations for (∇, s) and (∇, t), we start with $(\{X : s \triangleq t\}, \emptyset, \nabla, [])$, the *initial state* (sometimes we abbreviate it to $(\nabla, \{X : s \triangleq t\})$), where X is a fresh generalization variable, and we apply the rules as long as possible, until no more rule applications are possible, where no alternative rule applications have to be explored. The *final state* will be reached, which has the form $(\emptyset, M, \Delta, \theta)$. We will denote the computation from the initial state to the final state as $(\Gamma, \emptyset, \nabla, []) \Longrightarrow^* (\emptyset, M, \Delta, \theta)$. When convenient we will denote by $(\Gamma, M, \nabla, \theta) \Longrightarrow_{(\mathtt{R})} (\Gamma', M', \nabla', \theta')$, the one-step computation using a rule (R) from Figure 1. The output is a term-in-context obtained from the generated substitution θ and the final freshness constraint Δ, i.e. the output is $(\Delta, X \circ \theta)$, also called the *result computed* by the AtomAntiUnif algorithm.

We now describe the rules in Figure 1:

- The decomposition rule (Dec) is standard.
- Rule (Abs) is applicable for the generalization (on variable X) of two generalized abstractions: apply a swapping with a fresh atom-variable B. Its freshness is guaranteed by adding freshness constraints to ∇', which guarantee that the renaming by the permutation keeps α-equivalence, without losing solutions. The substitution is extended with a mapping $\{X \mapsto \lambda B.Y\}$, where Y is a fresh generalization variable.
- Rules (SusA) and (SusYY) are applicable for the generalization (on variable X) of suspensions of atom-variables or generalization variables, respectively. In both cases, the semantics of ∇ will identify the two suspensions, and the substitution will be extended with a mapping from X to one of the variables in the suspension. Note that only these two rules introduce the variables from the initial s, t into the solution substitution θ.
- Similarly to [4], the merging rule (Mer) relies on a generalization (to $\mathtt{NL}_A$) of the equivariance algorithm Eqvm, for computing a generalized permutation π, such that $\nabla \vDash \pi \cdot (s_1, t_1) = (s_2, t_2)$, if it exists.
- Rules (Solve), (SolveYY) and (SolveAB) can be applied when a condition is satisfied. The first rule applies when s and t do not have the same head, and are not suspensions of atom variables. Rule (SolveYY) treats the case of two suspensions on the same generalization variable Y, say $\pi_1 \cdot Y$ and $\pi_2 \cdot Y$, but the semantics of ∇ permits that both suspensions may be mapped to different atoms. Rule (SolveAB) treats the case of suspensions of atom-variables, in the case where the semantics of ∇ implies that both suspensions will be instantiated to two different terms. The algorithm then ignores the constraints acting on the variables in the equations that are moved to M.

(`Dec`): Decomposition

$$\frac{\{X{:}f(s_1,\ldots,s_n) \triangleq f(t_1,\ldots,t_n)\} \uplus \Gamma, M, \nabla, \theta}{\Gamma \uplus \{X_1{:}s_1 \triangleq t_1, \ldots, X_n{:}s_n \triangleq t_n\}, M, \nabla, \theta \cup \{X \mapsto f(X_1,\ldots,X_n)\}}$$

where X_i are fresh variables

(`Abs`): Abstraction

$$\frac{\{X{:}\lambda W_1.s \triangleq \lambda W_2.t\} \uplus \Gamma, M, \nabla, \theta}{\Gamma \uplus \{Y{:}(W_1\ B)\cdot s \triangleq (W_2\ B)\cdot t\}, M, \nabla \cup \{B\#\lambda W_1.s, B\#\lambda W_2.t\}, \theta \cup \{X \mapsto \lambda B.Y\}}$$

where Y is a fresh variable, and B is a fresh atom-variable

(`SusA`): SuspensionA

$$\frac{\{X{:}W_1 \triangleq W_2\} \uplus \Gamma, M, \nabla, \theta \quad \nabla \vDash W_1 = W_2}{\Gamma, M, \nabla, \theta \cup \{X \mapsto W_1\}}$$

(`SusYY`): SuspensionYY

$$\frac{\{X{:}\pi_1\cdot Y \triangleq \pi_2\cdot Y\} \uplus \Gamma, M, \nabla, \theta \quad \nabla \vDash \pi_1 = \pi_2}{\Gamma, M, \nabla, \theta \cup \{X \mapsto \pi_1\cdot Y\}}$$

(`Mer`): Merging

$$\frac{\Gamma, \{Z_1{:}s_1 \triangleq t_1, Z_2{:}s_2 \triangleq t_2\} \uplus M, \nabla, \theta \quad \textsc{Eqvm}(\{(s_1,t_1) \preceq (s_2,t_2)\}, \nabla) = \pi}{\Gamma, M \cup \{Z_1{:}s_1 \triangleq t_1\}, \nabla, \theta \cup \{Z_2 \mapsto \pi\cdot Z_1\}}$$

where (Z_1, Z_2) is (X, Y) or (A, B)

(`Solve`)

$$\frac{\{X{:}s \triangleq t\} \uplus \Gamma, M, \nabla, \theta}{\Gamma, M \cup \{X{:}s \triangleq t\}, \nabla, \theta}$$

If $\mathit{Head}(s) \neq \mathit{Head}(t)$ and if s and t are not both suspensions of atom-variables.

(`SolveYY`)

$$\frac{\{X{:}\pi_1\cdot Y \triangleq \pi_2\cdot Y\} \uplus \Gamma, M, \nabla, \theta \quad \nabla \nvDash \pi_1 = \pi_2}{\Gamma, M \cup \{X{:}\pi_1\cdot Y \triangleq \pi_2\cdot Y\}, \nabla, \theta}$$

(`SolveAB`)

$$\frac{\{X{:}W_1 \triangleq W_2\} \uplus \Gamma, M, \nabla, \theta \quad \nabla \nvDash W_1 = W_2}{\Gamma, M \cup \{A{:}W_1 \triangleq W_2\}, \nabla, \theta \cup \{X \mapsto A\}}$$

A is a fresh atom-variable.

Figure 1 Rules of the algorithm AtomAntiUnif.

- The generalization variables are always chosen fresh, thus these do not occur in freshness contexts, hence application of θ to ∇ is not necessary. Note that the effect of the rules on ∇ is only the addition of constraints by (`Abs`) (without a semantical change).
- By construction, in a computation $(\{X : s \triangleq t\}, \emptyset, \nabla, []) \Longrightarrow^* (\Gamma, M, \nabla, \theta)$ from the initial state to an intermediate state, the generalization variables in the range of the computed substitution θ satisfy $\texttt{GenVar}(\theta) \subseteq \texttt{GenVar}(\Gamma \cup M)$.

▶ Remark 3.1 (About completeness). Several rules do not inherit the optimal set of freshness constraints. It is unclear how and presumably complex to compute these. We leave this computation to the extra algorithm AtomAntiUnifLgg that computes a lgg from the result of AtomAntiUnif by adding and checking generalized freshness constraints (see Section 4)

▶ Remark 3.2. An algorithm for $\nabla \vDash \ldots$ is to check all equivalence classes of atom-variables, where equivalence of A, B semantically means equal images under a ground instantiation (see also Definition 2.8, Lemma 2.10, and Section 4). This can be performed in exponential time since it is sufficient to check all possibilities of equality and disequality of atom-variables.

The next example shows the use of the semantics of ∇ and the treatment of bindings.

▶ **Example 3.3.** Let the $\texttt{NL}_A$-expressions to be generalized be $\lambda A_1.A_2$ and $\lambda A_2.A_1$ under ∇. The lgg is $(\emptyset, \lambda A.A)$ or $(\emptyset, \lambda A.B)$ for $A \neq B$, depending on whether $\nabla \vDash A = B$ or $\nabla \nvDash A = B$. In the first case (`Abs`) is applied and in the latter case also (`SusA`).

The algorithm AtomAntiUnif is naive: it finds a generalization for two $\texttt{NL}_A$-terms-in-context (∇, s) and (∇, t) but not necessarily the least general one:

$$\frac{\Psi \dot\cup \{e \preceq e\}, \Pi, \nabla}{\Psi, \Pi, \nabla} \qquad \frac{\Psi \dot\cup \{W_1 \preceq W_2\}, \Pi, \nabla}{\Psi, \{W_1 \mapsto W_2\} \cup \Pi, \nabla} \qquad \frac{\Psi \dot\cup \{(f\ s_1 \dots s_n) \preceq (f\ s'_1 \dots s'_n)\}, \Pi, \nabla}{\Psi \cup \{s_1 \preceq s'_1, \dots, s_n \preceq s'_n\}, \Pi, \nabla}$$

$$\frac{\Psi \dot\cup \{\pi_1 \cdot X \preceq \pi_2 \cdot X\}, \Pi, \nabla \quad \nabla \vDash \pi_1 \cdot X = \pi_2 \cdot X}{\Psi, \Pi, \nabla} \qquad \frac{\Psi \dot\cup \{\lambda W_1.s \preceq \lambda W_2.t\}, \Pi, \nabla \quad \nabla \vDash W_2 \# \lambda W_1.s}{\Psi \cup \{(W_1\ W_2) \cdot s \preceq t\}, \Pi, \nabla}$$

$$\frac{\Psi \dot\cup \{\lambda W_1.s \preceq \lambda W_2.t\}, \Pi, \nabla \quad \nabla \vDash W_1 \# \lambda W_2.t}{\Psi \cup \{s \preceq (W_1\ W_2) \cdot t\}, \Pi, \nabla} \qquad \frac{\emptyset, \Pi, \nabla \qquad \textsc{EqvBiEx}(\Pi, \nabla) = \pi}{\text{Return } \pi}$$

Figure 2 Rules of the permutation matching algorithm EQVM.

$$\frac{\nabla \vDash W_1 = W'_1 \wedge W_2 = W'_2}{(\Pi \dot\cup \{W_1 \mapsto W_2, W'_1 \mapsto W'_2\}, \nabla)} \qquad \frac{\nabla \vDash W_1 = W'_1, \quad \nabla \nvDash W_2 = W'_2}{(\Pi \dot\cup \{W_1 \mapsto W_2, W'_1 \mapsto W'_2\}, \nabla)}$$
$$(\Pi \dot\cup \{W_1 \mapsto W_2\}, \nabla) \qquad \bot$$

$$\frac{\nabla \vDash W_2 = W'_2, \nabla \nvDash W_1 = W'_1}{(\Pi \dot\cup \{W_1 \mapsto W_2, W'_1 \mapsto W'_2\}, \nabla)} \qquad \frac{\Pi, \nabla \qquad \text{no other rule is applicable}}{\text{Return a permutation computed from } \Pi}$$
$$\bot$$

Figure 3 Rules of the bijection extraction algorithm EQVBIEX.

▶ **Example 3.4.** Consider the $\mathtt{NL}_A$-terms-in-context $(\emptyset, f(c_1, A))$ and $(\emptyset, f(c_2, A))$ that have to be generalized, where c_1 and c_2 are different constant symbols. The generalization computed by the ATOMANTIUNIF algorithm is $(\emptyset, f(X', A))$, obtained via the derivation: $(\{X : f(c_1, A) \triangleq f(c_2, A)\}, \emptyset, \emptyset, []) \Longrightarrow^* (\emptyset, \{X' : c_1 \triangleq c_2\}, \emptyset, \{X \mapsto f(X', A)\})$ However, this is not the smallest generalization, since $(\{A \# X'\}, f(X', A))$ has a smaller set of instances, and is less general. The reason that this construction works is that A is a part of the solution, but it does not occur in the solved equation associated to X'.

Notice, however, that $f(A, A)$ and $f(c, A)$ have a generalization $(\emptyset, f(X', A))$, via the derivation: $(\{X : f(A, A) \triangleq f(c_2, A)\}, \emptyset, \emptyset, []) \Longrightarrow^* (\emptyset, \{X' : A \triangleq c_2\}, \emptyset, \{X \mapsto f(X', A)\})$. But $\{A \# X'\}$ cannot be added as freshness context: although A is part of the solution, A occurs in the solved equation associated to X'. In fact, $(\{A \# X'\}, f(X', A))$ is not a generalization of $f(A, A)$ and $f(c, A)$, and $(\emptyset, f(X', A))$ is the lgg.

3.2 Equivariance Algorithm

The merge-rule as in [4] relies on solving an equivariance problem [9]. It will be treated similarly here, however, generalized to atom-variables and nested permutations. We will use a matching-like rule-based algorithm that finally is able to produce a permutation for the merge rule, if there is one at all, and otherwise fails. Instead of fixing a derivation algorithm $\nabla \vdash \dots$, we will use the semantic variant $\nabla \vDash \dots$ and mean it to be as general as possible, which will leave some open space for optimizations.

▶ **Algorithm 3.5.** *The rules of the non-deterministic algorithm* EQVM *are in Figures 2 and 3. They operate on triples of the form* (Ψ, Π, Δ) *where*

- Ψ *is a set of matching problems of the form* $e \preceq e'$*;*
- Π *are the potential (mapping) components of the permutation computed so far;*
- Δ *is set of freshness constraints.*

The initial triple is $(\Psi, \emptyset, \nabla)$*, where* Ψ *and* ∇ *are delivered in the call to this algorithm. The rules are to be applied as long as possible. If a state of the form* $(\emptyset, \Pi, \Delta)$ *is reached, and* Δ *implies that the mappings in* Π *do not collide and can be completed into a permutation*

(a bijection), then the algorithm is successful. The permutation will be computed using the algorithm EqvBiEx *in Figure 3. In the success case a permutation π from Π is returned after an exhaustive run without a fail, where for the permutation perhaps some mappings have to be added. For example, the result $\{A \mapsto C, B \mapsto D, C \mapsto E\}$ is made a bijection by adding $\{D \mapsto A, E \mapsto B\}$, which can then be represented as a permutation.*

▶ **Proposition 3.6.** *The algorithm* Eqvm *is sound, correct and terminates in a linear number of steps with a computed permutation if there is any.*

Proof. Every step is easily justified and makes the set Ψ strictly smaller. The major parts of the complexity are the calls to $\nabla \vDash$. For the complexity of $\nabla \vDash \ldots$ see Proposition 2.11. ◀

3.3 Properties of AtomAntiUnif

From an AtomAntiUnif derivation $(\{X : s \triangleq t\}, \emptyset, \nabla, []) \Longrightarrow^* (\emptyset, M, \Delta, \theta)$ we can obtain substitutions ρ_1 and ρ_2 mapping generalization variables $\texttt{GenVar}(\theta)$ to $\texttt{NL}_A$-expressions, making a connection between $X \circ \theta$, s and t (similar as in [4]). From the set of solved equations $M = \{X_1 : s_1 \triangleq t_1, \ldots, X_n : s_n \triangleq t_n\}$, we define $\rho_1 = \{X_1 \mapsto s_1, \ldots, X_n \mapsto s_n\}$ and $\rho_2 = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$, such that $[\![(\nabla, s)]\!] \subseteq [\![(\Delta, (X \circ \theta)\rho_1)]\!]$ and $[\![(\nabla, t)]\!] \subseteq [\![(\Delta, (X \circ \theta)\rho_2)]\!]$. In general, $[\![(\nabla, s)]\!] \cup [\![(\nabla, t)]\!] \subseteq [\![(\Delta, (X \circ \theta)\rho_1)]\!] \cup [\![(\Delta, (X \circ \theta)\rho_2)]\!]$ where this is also specialized to the intermediate states of AtomAntiUnif.

This connection motivates an extension of the semantics for $\texttt{NL}_A$-terms-in-context (∇, s) to AtomAntiUnif states, which will allow us to show that the semantics increase in each step of rule application.

▶ **Definition 3.7.** *Let $(\{X : s \triangleq t\}, \emptyset, \nabla, [])$ be an initial state of* AtomAntiUnif*. The semantics of terms-in-context w.r.t. states $S := (\Gamma, M, \nabla, \theta)$ of* AtomAntiUnif *as follows:*

- $[\![(\{X : s \triangleq t\}, \emptyset, \nabla, [])]\!] := [\![(\nabla, s)]\!] \cup [\![(\nabla, t)]\!]$
- $[\![(\emptyset, \emptyset, \nabla, \theta)]\!] := [\![(\nabla, X \circ \theta)]\!]$*, where X is the input generalization variable.*
- *Otherwise, for $\Gamma \cup M = \{Y_i : s_i \triangleq t_i \mid i = 1, \ldots, n\} \neq \emptyset$:*

$$[\![(\Gamma, M, \nabla, \theta)]\!] := \{[(X \circ \theta)\rho_s]_\sim \mid \rho_s \text{ is ground, } \nabla\rho_s \text{ holds, and } \forall i.[Y_i\rho_s]_\sim \in [\![(\nabla, s_i)]\!]\} \cup \{[(X \circ \theta)\rho_t]_\sim \mid \rho_t \text{ is ground, } \nabla\rho_t \text{ holds, and } \forall i.[Y_i\rho_t]_\sim \in [\![(\nabla, t_i)]\!]\}$$

where X is the input generalization variable.

▶ **Proposition 3.8.** *The algorithm* AtomAntiUnif *never gets stuck and will yield a generalization. The number of rule applications is linear.*

As illustrated in previous examples, our algorithm outputs a generalization but not always the least general one (cf. Example 3.4). Therefore, we can establish the following weaker completeness theorem that establishes a characterization of lggs up to a freshness context.

▶ **Theorem 3.9** (Completeness up to Freshness Contexts)**.** *Given $\texttt{NL}_A$ expressions s and t, and a freshness context ∇. If (∇', r) is a generalization of (∇, s) and (∇, t), then there exists a ∇'' and a derivation $(\{X : s \triangleq t\}, \emptyset, \nabla, []) \Longrightarrow^* (\emptyset, M, \Delta, \theta)$ such that $[\![(\Delta \cup \nabla'', X \circ \theta)]\!]$ is a generalization of (∇, s) and (∇, t), and $[\![(\emptyset, M, \Delta \cup \nabla'', \theta)]\!] \subseteq [\![(\nabla', r)]\!]$.*

Proof. The proof follows by induction on the number of rule applications from Figure 1. The complete proof can be found in the appendix. ◀

Soundness of AtomAntiUnif is obtained by an easy inspection of the rules in Figure 1 and the proof is omitted.

▶ **Theorem 3.10** (Complexity of the Algorithm). *The algorithm* AtomAntiUnif *requires simple exponential time to compute a solution. The number of rule applications is polynomial, and the solution requires polynomial space.*

Proof. The number of rule applications of the main algorithm is polynomial, since the size of $\Gamma \cup M$ is strictly reduced in each step, where the *size* is meant as follows: the term size ignoring the permutations, and a generalization variable has size 2, whereas an atom-variable has size 1. The total size, including permutations, remains polynomial, since there is only a constant-size increase per rule application in Γ, M, ∇. The size of θ also remains polynomial, since solution components are not applied. The call to subalgorithms may be (simply) exponential depending on the size of the problem. Since there is only a polynomial number of such calls, and the size remains polynomial, the algorithm requires simple exponential time in the worst case. ◀

The following shows that the $\mathtt{NL}_A$-freshness contexts have a finiteness property, where an upper bound is given. We consider two freshness constraints ∇_1, ∇_2 as equivalent w.r.t. s iff $[\![(\nabla_1, s)]\!] = [\![(\nabla_2, s)]\!]$.

▶ **Theorem 3.11.** *Let s be a generalization term. Then the number of equivalence classes of $\mathtt{NL}_A$-freshness constraints ∇ w.r.t. s is finite. Their number is in $O(n^n) = O(e^{n*(log(n)+1)})$.*

Proof. Let $M = \mathtt{AtVar}(s)$, and let ∇_1 and ∇_2 be freshness contexts, where we assume that freshness constraints are of the two forms $A\#\lambda W_1 \ldots \lambda W_n.W$ and $A\#\lambda W_1 \ldots \lambda W_n.\pi{\cdot}X$. **(Equivalence test)** The test for equivalence of two freshness contexts ∇_1 and ∇_2 is as follows: For all equivalence relations R on M and for all freshness contexts $\Delta_0 \subseteq \{A\#X \mid A \in \mathtt{AtVar}(s), X \in \mathtt{GenVar}(s)\}$:

- Check the equivalence of ∇_1 and ∇_2 (under R and Δ_0) of the following two tests:
  - **(1)** There is an equivalence relation R' on $\mathtt{AtVar}(\nabla_1, s)$ that soundly extends R, i.e. $A \sim_R A'$ iff $A \sim_{R'} A'$ for all atom-variables A, A' in M. Such that:
    - **(i)** All AV-constraints $A\#\lambda W_1 \ldots \lambda W_n.W$ in ∇_1 evaluate to true under R'.
    - **(ii)** The set of GV-constraints ∇_G in ∇_1 is equivalent to Δ_0 using R' for simplification.
  - **(2)** There is an equivalence relation R'' on $\mathtt{AtVar}(\nabla_2, s)$ that soundly extends R, i.e. $A \sim_R A'$ iff $A \sim_{R''} A'$ for all atom-variables A, A' in $\mathtt{AtVar}(\nabla_2, s)$. Such that:
    - **(i)** All AV-constraints $A\#\lambda W_1 \ldots \lambda W_n.W$ in ∇_2 evaluate to true under R''.
    - **(ii)** The set of GV-constraints ∇_G in ∇_2 is equivalent to Δ_0 using R' for simplification.
- If for all R, and all selected freshness contexts Δ_0, the combined tests leads to true, then the freshness contexts ∇_1 and ∇_2 are equivalent w.r.t. s.

(**Finiteness**) of the set of equivalence classes of freshness constraints follows, since the set M depends only on s, and hence there is a finite set of equivalence relations over M. Also, there is only a finite set of possibilities for Δ_0. An upper bound for the number of equivalence classes R is $O(n^n) = O(e^{(n*log(n)}$, where n is the number of atom-variables in s. (The number of equivalence relations for n atom-variables is also called the Bell-number[2] of n.)

The number of different sets of freshness constraints is at most exponential, i.e. $O(2^n)$ where n is the number of atom-variables, since these are all subsets of a set of atom-variables. Since both are combined, the number of possibilities is their product, and thus the growth is in $O(e^{(n*(log(n)+1))}$. ◀

[2] see `https://mathworld.wolfram.com/BellNumber.html`.

▶ **Corollary 3.12.** *Let (∇, Γ) be an input for the* AtomAntiUnif *algorithm and (∇', s) be the computed result. Then, there are no infinitely properly descending chains (∇', s), (∇'_1, s), $(\nabla'_2, s), \ldots$ of generalizations. A consequence is that the solution type of generalization problems with atom-variables is finitary or even unitary.*

4 Computing Least General Generalizations

In this section we describe the final step of computing lggs, which builds upon the result of the algorithm AtomAntiUnif. First, we define a more general form of freshness constraints. Then, we provide a specialized algorithm, called AtomAntiUnifLgg, that computes an lgg by strengthening the (general) freshness constraints.

Notice that the addition of constraints of the form $A\#X$ is not sufficient to reach an lgg:

▶ **Example 4.1.** Let the input be $(\emptyset, \{X : (f(A), A, B) \triangleq (c, A, B)\})$, where c, f are function constants. AtomAntiUnif computes the generalization $(\emptyset, (Y, A, B))$, however, it is not an lgg. Adding $A\#Y$ or $B\#Y$ violates the generalization property. However, $B\#\lambda A.Y$ can be added as constraint, and then $(\{B\#\lambda A.Y\}, (Y, A, B))$ allows the instance $(f(a), a, a)$, since $a\#\lambda a.f(a)$ holds. We see that $(\{B\#\lambda A.Y\}, (Y, A, B))$ is the lgg.

4.1 A Generalized Representation for lgg

In order to represent and compute lggs, we switch to a more general representation of freshness constraints and contexts, called Eqr-*freshness constraints*, that are obtained by analyzing the possible Equivalence Relations over a set of atom-variables $\mathcal{A}$. We will show that a least general generalization using Eqr-*freshness contexts* can be obtained and how to compute it. The extended form allows for a finite generation of extra freshness constraints and also a computable test of whether the extended solution still is a generalization. Note that this is the same as making a complete case-distinction over all possible equality/disequalities of ground instantiations of atom-variables.

▶ **Definition 4.2.** *Let $\mathcal{A} = \{A_1, \ldots A_n\}$ be a set of atom-variables and $\{Y_1, \ldots, Y_m\}$ be a set of (generalization) variables.*

- *An* Eqr-freshness constraint *is of the form $(Q \implies Q')$ where Q is a conjunction of constraints of the forms $A \neq A'$ or $A = A'$. The part Q' is* `True`*, or* `False`*, or a (positive) propositional formula formed using $\wedge$,$\vee$, and freshness constraints of the form $A_i\#Y_j$.*
- *An* Eqr-(freshness) context *is a conjunction of* Eqr*-constraints, i.e., it has the form $(Q_1 \implies Q'_1) \wedge \ldots \wedge (Q_m \implies Q'_m)$ where the Q_i are exactly all equivalence relations within $\mathcal{A}$. Notice that if $A = B$ holds in Q_i, then the representation is not unique and could be extended or restricted, e.g., by replacing $A\#X$ by $B\#X$ or by similar operations.*

▶ **Example 4.3.** Let $\mathcal{A} = \{A, B\}$, and X be a variable. An example for an Eqr- freshness context is: $(\{A = B\} \implies (A\#X \wedge B\#X)) \wedge (\{A \neq B\} \implies (B\#X))$.
A second example is the constraint $A\#\lambda B.A$ that has an equivalent Eqr-freshness context: $(\{A = B\} \implies \texttt{True}) \wedge (\{A \neq B\} \implies \texttt{False})$.
A third example is the freshness constraint $A\#\lambda B.(B\ A)\cdot Y$ with $\mathcal{A} = \{A, B\}$. The equivalent Eqr-freshness context is $(A = B \implies \texttt{True}) \wedge (A \neq B \implies B\#Y)$.

▶ **Definition 4.4.** *For an equivalence relation R on $\mathcal{A}$, we define a related evaluation of freshness constraints, denoted as $\gamma(R, A\#s)$, which exploits the equalities and inequalities of a single R to maximally simplify the constraint(s), and thereby eliminating all permutations by perhaps applying the inverse if necessary, such that the result is only* `True`,`False`*, or a freshness constraint of the form $A\#B$, or $A\#Y$.*

The next result establishes an action of $\gamma(R)$ over freshness constraints, connecting them to EQR-freshness constraints.

▶ **Proposition 4.5.** *Every freshness context can also be expressed as an* EQR*-freshness context. The size of the generated output may be exponential.*

There exist EQR-freshness contexts that cannot be encoded as $\mathtt{NL}_A$-freshness context:

▶ **Lemma 4.6.** *Let $\mathcal{A} = \{A, B\}$ be a set of atom-variables, X and Y be generalization variables. The* EQR*- freshness context $((A \neq B) \implies A\#X) \wedge ((A = B) \implies A\#Y \wedge B\#Y)$ cannot be encoded as an $\mathtt{NL}_A$-freshness context, i.e, a conjunction of $\mathtt{NL}_A$-freshness constraints.*

▶ **Corollary 4.7.** EQR*-contexts are strictly more expressive than $\mathtt{NL}_A$-freshness contexts.*

The next example illustrates (i) the evaluation of a freshness constraint under an equivalence relation R in a term-in-context (ii) the omission of a redundant atom-variable, and is an example where a single least general generalization does not exist.

▶ **Example 4.8.** Let $\mathcal{A} = \{A, B, C\}$ and $\nabla = \{C\#\lambda A.\lambda B.C, C\#X\}$ be a freshness context. A first generalization of $(\nabla, X' : f(A, B, C) \triangleq f(A, B, X))$ is the term-in-context $(\nabla, f(A, B, X))$. An EQR-freshness context ∇_0 representing this (obtained using Definition 4.2) is as follows:

1)		$(\{A = B, B = C, A = C\}$	$\implies$	$(C\#X))\}$	(or $A\#X$)
2)	$\wedge$	$(\{A = B, B \neq C, A \neq C\}$	$\implies$	`False`)	
3)	$\wedge$	$(\{A \neq B, B \neq C, A \neq C\}$	$\implies$	`False`)	
4)	$\wedge$	$(\{A \neq B, B = C, A \neq C\}$	$\implies$	$(C\#X))$	(or $B\#X$)
5)	$\wedge$	$(\{A \neq B, A = C, B \neq C\}$	$\implies$	$(C\#X))$	(or $A\#X$)

Some computations (see also Proposition 4.9 for the general case) show that $(\{A = B\} \implies A\#X) \wedge (\{A \neq B\} \implies (B\#X \vee A\#X))$ is an equivalent EQR-freshness context for the set $\{A, B\}$, which cannot be represented as an $\mathtt{NL}_A$-freshness context using only $\{A, B\}$ due to the disjunction. This is the (unique) lgg w.r.t. EQR-freshness contexts.

Notice that a complete set consists of two lggs w.r.t. $\mathtt{NL}_A$-freshness contexts: $(\{A\#X\}, f(A, B, X))$ and $(\{B\#X\}, f(A, B, X))$, if only atom-variables from $f(A, B, X)$ are permitted. However, using additional atom-variables (here C), we obtain a unique lgg $(\nabla, f(A, B, X))$. This example provides an argument for permitting a larger set of atom-variables as the ones contained in the term of the term-in-context for obtaining lggs w.r.t. $\mathtt{NL}_A$-freshness contexts.

▶ **Proposition 4.9** (Restricting)**.** *Let $\mathcal{A}' \subset \mathcal{A}$ be a set of atoms, $X_1, \ldots, X_n$ be variables, t be an expression with $\mathtt{AtVar}(t) \subseteq \mathcal{A}'$, and ∇ be a freshness context over $\mathcal{A}$. Then there is an equivalent* EQR*-freshness context ∇' over $\mathcal{A}'$, such that $[\![(\nabla, t)]\!] = [\![(\nabla', t)]\!]$.*

4.2 The AtomAntiUnifLgg algorithm

This algorithm strengthens the freshness constraints of the computed generalization given as output by the AtomAntiUnif algorithm. It takes as input $(\nabla_{in}, Y{:}r_1 \triangleq r_2)$ and $(\nabla_{\mathrm{AAU}}, t)$, where $(\nabla_{in}, Y{:}r_1 \triangleq r_2)$ is the input of AtomAntiUnif with result $(\nabla_{\mathrm{AAU}}, t)$, and $\nabla_{\min}$ is the result of restricting ∇_{AAU} to the atom-variables in t (cf. Proposition 4.9). Then all atom variables occurring in $\nabla_{\min}$ also occur in $(\nabla_{in}, Y : r_1 \triangleq r_2)$. The output will be the freshness context $\nabla_{\min}$ joined with a generalized freshness context $\mathcal{Q}$.

▶ **Definition 4.10.** *The algorithm* AtomAntiUnifLgg *for computing a least general generalization is defined in Figure 4.*

1: **Input:** $(Y : r_1 \triangleq r_2, \nabla_{in}), (\nabla_{\mathrm{AAU}}, t)$
2: Let $\mathcal{A} = \mathtt{AtVar}(t)$.
3: Let $\nabla_{\min}$ be the result of restricting ∇_{AAU} to $\mathcal{A}$ as in (the algorithm from) Proposition 4.9.
4: $\mathcal{X} = \mathtt{GenVar}(t)$; $\quad \mathcal{Q} := \mathtt{True}$
5: **for** every equivalence relation R on $\mathcal{A}$ **do** ▷ Computation of $\mathcal{Q}$
6: let $Q_1 = \gamma(R)$
7: let Q_2 be the maximal possible positive formula of GV-constraints $A\#X$ with $A \in \mathcal{A}$
8: and $X \in \mathcal{X}$ such that the following holds:

a) $[\![(\nabla_{in}, r_1)]\!] \subseteq [\![((Q_1 \implies Q_2) \cup \nabla_{\min}), t)]\!]$.
b) $[\![(\nabla_{in}, r_2)]\!] \subseteq [\![((Q_1 \implies Q_2) \cup \nabla_{\min}), t)]\!]$.

9: Let Q_2' and $\mathcal{Q}'$ be such that $\mathcal{Q} = (\mathcal{Q}' \wedge (Q_1 \implies Q_2'))$
10: $\mathcal{Q} := (\mathcal{Q}' \wedge (Q_1 \implies (Q_2' \wedge Q_2)))$
11: **end for**
12: **Output:** $(\nabla_{\min} \cup \mathcal{Q}, t)$

Figure 4 The AtomAntiUnifLgg Algorithm.

▶ **Proposition 4.11** (Unique Maximal Constraint). *Let (∇, t) be a term-in-context with* Eqr*-freshness context ∇, where $\mathtt{AtVar}(\nabla) \subseteq \mathtt{AtVar}(t)$ and $\mathtt{GenVar}(\nabla) \subseteq \mathtt{GenVar}(t)$, and let (∇_1, r_1), (∇_1, r_2) be terms-in-context with $\mathtt{AtVar}(r_1, r_2) \subseteq \mathtt{AtVar}(t)$, $\mathtt{GenVar}(r_1, r_2) \subseteq \mathtt{GenVar}(t)$, and $[\![(\nabla_1, r_1)]\!] \subseteq [\![(\nabla, t)]\!]$, $[\![(\nabla_1, r_2)]\!] \subseteq [\![(\nabla, t)]\!]$. Then there is an* Eqr*-freshness context ∇', such that $[\![(\nabla', t)]\!] \subseteq [\![(\nabla, t)]\!]$, and $[\![(\nabla_1, r_1)]\!] \subseteq [\![(\nabla', t)]\!]$, $[\![(\nabla_1, r_2)]\!] \subseteq [\![(\nabla', t)]\!]$, and $[\![(\nabla', t)]\!]$ is the minimum that is unique w.r.t. the properties above.*

Proof. Two Eqr-freshness constraints over the same set $\mathcal{A}$ of atom-variables can be joined as follows: If $(Q_1 \implies P_{1,i}) \wedge \ldots \wedge (Q_n \implies P_{n,i})$ for $i = 1, 2$ are two Eqr-freshness constraints over $\mathcal{A}$. Then the join can be represented as the Eqr-freshness constraint $(Q_1 \implies (P_{1,1} \wedge P_{1,2})) \wedge \ldots \wedge (Q_n \implies (P_{n,1} \wedge P_{n,2}))$, and the result follows. ◀

▶ **Theorem 4.12.** *Let $(\nabla_{\mathrm{AAU}}, r)$ be a generalization output by the nominal* AtomAntiUnif *algorithm when given $(\nabla, X : r_1 \triangleq r_2)$ as input. Then* AtomAntiUnifLgg *applied to $(\nabla_{\mathrm{AAU}}, r)$ outputs a least general generalization for $(\Delta, X : r_1 \triangleq r_2)$. In addition, this strengthening has a maximal result and it is unique up to equivalences.*

Since the algorithm AtomAntiUnif and the other algorithms also work in the same way if AtomAntiUnif is started with more general terms-in-context where the constraint is an Eqr-constraint, we obtain that the nominal anti-unification problem with atom-variables and Eqr-freshness constraints is decidable and a minimal complete set of lggs exists and has exactly one generalization.

▶ **Theorem 4.13.** *The nominal anti-unification problem of $\mathtt{NL}_A$-expressions with atom-variables and* Eqr*-freshness contexts is decidable and unitary.*

Application of Theorem 3.11 implies that the anti-unification problem for $\mathtt{NL}_A$-expressions and contexts is at most finitary.

▶ **Theorem 4.14.** *Let A_0 be a finite set of atom-variables. Then the nominal anti-unification problem of $\mathtt{NL}_A$-expressions and $\mathtt{NL}_A$-freshness contexts and where only atom-variables from A_0 may appear in the constraint part of the computed generalizations, is decidable and unitary or finitary.*

▶ **Theorem 4.15.** *If the atom-variables in the constraint parts of the terms-in-context are not restricted, then the nominal anti-unification problem of* $\mathtt{NL}_A$ *-expressions and* $\mathtt{NL}_A$*-freshness contexts is unitary or finitary.*

Investigating decidability of computing a complete set of lggs in Theorem 4.15 with $\mathtt{NL}_A$-freshness contexts and also determining the exact solution type is future research. Applying Theorem 3.11 does not solve the computation problem in the general case of $\mathtt{NL}_A$-freshness contexts, since we do not know an upper bound on the number of atom-variables that may occur in the constraint part of the lggs.

4.3 The Atoms-Only Case

We reconsider the nominal unification problem treated by Baumgartner et al. [4]. We model this case by adding the global condition that all atom-variables must be instantiated with different atoms and call it the *atoms-only case*. Then simplifications on the expressions of the abstract language are valid, such that only suspensions of generalization variables are needed, and freshness contexts are conjunctions of constraints of the form $A\#X$. Specializing our algorithm is straightforward. The (`Abs`)-rule only needs the condition that a *fresh* atom variable is used for renaming. The four rules (`SusA`), (`SusYY`), (`SolveYY`), (`SolveAB`) now only rely on an easy equality test of permutations. The same for EQVM and EQVBIEX, which now run in polynomial time. If also sharing of the substitution components is obeyed, then ATOMANTIUNIF runs in polynomial time, and since alternatives do not have to be investigated, one run is sufficient. For computing an lgg, from the result, we extend the algorithm by simply checking for all possible constraints of the form $A\#X$, whether these can be added to the solution and keeping the generalization property. This check is polynomial. The result constraint will be the union of all these constraints that pass the test.

▶ **Theorem 4.16.** *For the atoms-only case, the adapted and extended algorithm* ATOMANTIUNIF *runs in polynomial time, and always computes a unique lgg of the input* (∇, s) *and* (∇, r)*. This implies that the solution-type is unitary.*

5 Conclusion and Future Work

A sound and complete rule-based algorithm (ATOMANTIUNIF) for nominal anti-unification with atom-variables is provided. Our algorithm relies on an equivariance algorithm (EQVM) that is generalized to atom-variables and nested permutations. It computes, in exponential time, a *unique* (but not necessarily least) generalization of two terms-in-context. We refine the output generalization with the ATOMANTIUNIFLGG algorithm, which adds EQR-freshness constraints to the solutions, and a unique lgg is obtained. Our approach improves previous results [4], since the nominal anti-unification problem with atom-variables and EQR-freshness constraints is decidable and unitary. The variant with $\mathtt{NL}_A$-freshness constraints is unitary or finitary, but its decidability and the determination of the exact solution type is future work. We also describe an improved algorithm of the problem setting of Baumgartner et al.(the atoms-only-case), which is unitary and still requires polynomial time. Future work is to optimize the algorithms, investigate specializations like a restriction to linear generalizations, and to determine the exact complexity of the problem. Applications of nominal techniques in code duplication and recursion scheme detection are also to be investigated. Also an anti-unification algorithm for a core-language of Haskell [20, 12] that takes recursive lets into account (see [22, 23]) would extend applicability.

References

1 Franz Baader. Unification, weak unification, upper bound, lower bound, and generalization problems. In Ronald V. Book, editor, *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings*, volume 488 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 1991. doi:10.1007/3-540-53904-2_88.

2 Adam D. Barwell, Christopher Brown, and Kevin Hammond. Finding parallel functional pearls: Automatic parallel recursion scheme detection in haskell functions via anti-unification. *Future Gener. Comput. Syst.*, 79:669–686, 2018. doi:10.1016/j.future.2017.07.024.

3 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. doi:10.1007/978-3-319-11558-0_38.

4 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, 2015, Warsaw, Poland*, volume 36 of *LIPIcs*, pages 57–73. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.RTA.2015.57.

5 Peter E. Bulychev, Egor V. Kostylev, and Vladimir A. Zakharov. Anti-unification algorithms and their applications in program analysis. In Amir Pnueli, Irina B. Virbitskaite, and Andrei Voronkov, editors, *Perspectives of Systems Informatics, 7th International Andrei Ershov Memorial Conference, PSI 2009, Novosibirsk, Russia, June 15-19, 2009. Revised Papers*, volume 5947 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2009. doi:10.1007/978-3-642-11486-1_35.

6 Christophe Calvès and Maribel Fernández. A polynomial nominal unification algorithm. *Theor. Comput. Sci.*, 403(2-3):285–306, 2008. doi:10.1016/j.tcs.2008.05.012.

7 David M. Cerna and Temur Kutsia. Higher-order equational pattern anti-unification. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.FSCD.2018.12.

8 David M. Cerna and Temur Kutsia. Higher-order pattern generalization modulo equational theories. *Math. Struct. Comput. Sci.*, 30(6):627–663, 2020. doi:10.1017/S0960129520000110.

9 James Cheney. Toward a general theory of names: Binding and scope. In *MERLIN 2005*, pages 33–40. ACM, 2005.

10 Hubert Comon. Sufficient completness, term rewriting systems and "anti-unification". In Jörg H. Siekmann, editor, *8th International Conference on Automated Deduction, Oxford, England, July 27 - August 1, 1986, Proceedings*, volume 230 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 1986. doi:10.1007/3-540-16780-3_85.

11 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects Comput.*, 13(3-5):341–363, 2002. doi:10.1007/s001650200016.

12 Haskell. Haskell, an advanced, purely functional programming language, 2019. URL: www.haskell.org.

13 Boris Konev and Temur Kutsia. Anti-unification of concepts in description logic EL. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, pages 227–236. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12880.

14 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, *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*, volume 4830 of *Lecture*

Notes in Computer Science, pages 273–282. Springer, 2007. doi:10.1007/978-3-540-76928-6_29.

15 Yunus D. K. Kutz and Manfred Schmidt-Schauß. Rewriting with generalized nominal unification. *Math. Struct. Comput. Sci.*, 30(6):710–735, 2020. doi:10.1017/S0960129520000122.

16 Jean-Louis Lassez and Kim Marriott. Explicit representation of terms defined by counter examples. *J. Autom. Reason.*, 3(3):301–317, 1987. doi:10.1007/BF00243794.

17 Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. In *19th RTA*, volume 5117 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2008.

18 Jordi Levy and Mateu Villaret. An efficient nominal unification algorithm. In Christopher Lynch, editor, *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 209–226. Schloss Dagstuhl, 2010. doi:10.4230/LIPIcs.RTA.2010.209.

19 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. doi:10.1023/A:1018952121991.

20 Simon Marlow, editor. *Haskell 2010 – Language Report*. Haskell.org, 2010.

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

22 Manfred Schmidt-Schauß, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal unification of higher order expressions with recursive let. In Manuel V. Hermenegildo and Pedro López-García, editors, *Logic-Based Program Synthesis and Transformation - 26th International Symposium, LOPSTR 2016, Edinburgh, UK, September 6-8, 2016, Revised Selected Papers*, volume 10184 of *LNCS*, pages 328–344. Springer, 2016. doi:10.1007/978-3-319-63139-4_19.

23 Manfred Schmidt-Schauß, Temur Kutsia, Jordi Levy, Mateu Villaret, and Yunus Kutz. Nominal unification and matching of higher order expressions with recursive let. *Fundamenta Informaticae*, 2022. to be published in volume 185 issue 03.

24 Manfred Schmidt-Schauß, David Sabel, and Yunus D. K. Kutz. Nominal unification with atom-variables. *J. Symb. Comput.*, 90:42–64, 2019. doi:10.1016/j.jsc.2018.04.003.

25 Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. Nominal unification. In *17th CSL, 12th EACSL, and 8th KGC*, volume 2803 of *LNCS*, pages 513–527. Springer, 2003. doi:10.1007/978-3-540-45220-1_41.

A Proofs of Section 3 and Section 4

It is convenient to note that given a state $(\Gamma, M, \nabla, \theta)$ obtained from a starting configuration $(\{X : s \triangleq t\}, \emptyset, \nabla, [])$, the instance of the input variable $(X \circ \theta)$ is a context $C[X_1, \ldots, X_n]$ such that $\texttt{GenVar}(M \cup \Gamma) = \{X_1, \ldots, X_n\}$. The AtomAntiUnif rules do not alter the semantics of states in a derivation.

▶ **Lemma A.1.** *Let* $(\{X : s \triangleq t\}, \emptyset, \nabla_0, [])$ *an input for the* AtomAntiUnif *algorithm and* $(\Gamma, M, \nabla, \theta)$ *a state in the derivation. If* $(\Gamma, M, \nabla, \theta) \Longrightarrow_{(\mathtt{R})} (\Gamma', M', \nabla', \theta')$ *then* $[\![(\Gamma', M', \nabla', \theta')]\!] = [\![(\Gamma, M, \nabla, \theta)]\!]$.

Proof. The proof is by induction on the rule (R) from Figure 1 applied in $(\Gamma, M, \nabla, \theta) \Longrightarrow_{(\mathtt{R})} (\Gamma', M', \nabla', \theta')$. The interesting case is for rule (Abs):

In this case, $\Gamma = \{X_0 : \lambda W_1.s \triangleq \lambda W_2.t\} \cup \Gamma_0$, $\Gamma' = \{X_1 : (W_1\ B) \cdot s \triangleq (W_2\ B) \cdot t\} \cup \Gamma_0$, $M = M'$, $\theta' = \theta \cup \{X_0 \mapsto \lambda B.X_1\}$ and the reduction is: (we assume $\Gamma_0 = M = \emptyset$ w.l.o.g. because they will not be used in this step and to simplify the notation)

$$(\{X_0 : \lambda W_1.s \triangleq \lambda W_2.t\}, \emptyset, \nabla, \theta) \Longrightarrow (\{X_1 : (W_1\ B) \cdot s \triangleq (W_2\ B) \cdot t\}, \emptyset, \nabla \cup \nabla', \theta')$$

where $\nabla' = \{B \# \lambda W_1.s, B \# \lambda W_2.t\}$.
($\supseteq$) $[\![(\{X_1{:}(W_1\ B) \cdot s \triangleq (W_2\ B) \cdot t\}, \emptyset, \nabla \cup \nabla', \theta')]\!] \subseteq ([\![\{X_0{:}\lambda W_1.s \triangleq \lambda W_2.t\}, \emptyset, \nabla, \theta)]\!]$.

By Definition 3.7 and with X the input variable,

$$\begin{aligned}&[\![(\{X_1{:}(W_1\ B)\cdot s \triangleq (W_2\ B)\cdot t\},\emptyset,\nabla\cup\nabla',\theta')]\!]\\&= \{[(X\circ\theta')\rho_s] \mid \rho_s \text{ is ground}, (\nabla\cup\nabla')\rho_s \text{ holds}, [X_1\rho_s]_\sim \in [\![(\nabla\cup\nabla', (W_1\ B)\cdot s)]\!]\}\\&\cup \{[(X\circ\theta')\rho_t] \mid \rho_t \text{ is ground}, (\nabla\cup\nabla')\rho_t \text{ holds}, [X_1\rho_t]_\sim \in [\![(\nabla\cup\nabla', (W_2\ B)\cdot t)]\!]\}\end{aligned}$$

By Definition 2.12

$$[\![(\nabla\cup\nabla', (W_1\ B)\cdot s)]\!] = \{[((W_1\ B)\cdot s)\sigma]_\sim \mid \exists\sigma \text{ ground} : (\nabla\cup\nabla')\sigma \text{ holds}\}.$$

Let σ be a ground substitution s.t. $(\nabla\cup\nabla')\sigma$ holds, i.e, $\nabla\sigma$ and $\nabla'\sigma = \{B\#\lambda W_1.s, B\#\lambda W_2.t\}\sigma$ hold. Then, $B\sigma\#\lambda W_1\sigma.s\sigma$ and $B\sigma\#\lambda W_2\sigma.t\sigma$. We will analyze some cases:

- $B\sigma = W_1\sigma$.
  Then, $r\sim((W_1\ B)\cdot s)\sigma = s\sigma$. Thus, $[r]_\sim \in [\![(\nabla\cup\nabla', s)]\!]$ and $[X_1\rho_s]_\sim \in [\![(\nabla\cup\nabla', s)]\!]$. Notice that

  $$\begin{aligned}(X\theta')\rho_s &= (X(\theta\cup\{X_0\mapsto\lambda B.X_1\}))\rho_s\\&= C[X_1]\rho_s, \text{ for an } \mathtt{NL}_A\text{-context } C\\&= C'[\lambda B.X_1]\rho_s, \text{ for } C[] = C'[\lambda B.[]]\\&= C'[\lambda B\rho_s.X_1\rho_s] = C'[X_0\rho'_s] = (X\circ\theta)\rho'_s\end{aligned}$$

  where $[X_0\rho'_s]_\sim \in [\![(\nabla\cup\nabla', \lambda W_1.s)]\!]$. Since $\nabla\rho_s$ holds and

  $$\begin{aligned}[\![(\{X_0 : \lambda W_1.s \triangleq \lambda W_2.t\},\emptyset,\nabla,\theta)]\!] = \{[(X\theta)\rho_s] \mid \rho_s \text{ is ground}, \nabla\rho_s \text{ holds}, [X_0\rho_s]_\sim \in [\![(\nabla, \lambda W_1.s)]\!]\}\\\cup \{[(X\theta)\rho_t] \mid \rho_t \text{ is ground}, \nabla\rho_t \text{ holds}, [X_0\rho_t]_\sim \in [\![(\nabla, \lambda W_2.t)]\!]\},\end{aligned}$$

  one has $[(X\theta')\rho_s]_\sim \in [\![(\{X_0 : \lambda W_1.s \triangleq \lambda W_2.t\},\emptyset,\nabla,\theta)]\!]$, and the result follows.

- $B\sigma \neq W_1\sigma$.
  From $B\sigma\#\lambda W_1\sigma.s\sigma$, it follows that $B\sigma\#s\sigma$.

  - If W_1 does not occur in s then $r\sim((W_1\ B)\cdot s)\sigma = s\sigma$ and the case is similar to the previous.
  - Otherwise, W_1 occurs in s and $r\sim((W_1\sigma\ B\sigma)\cdot s\sigma)$, i.e., we replace all the occurrences of $W_1\sigma$ in $s\sigma$ for $B\sigma$.
    Thus, $[X_1\rho_s]_\sim = [r]_\sim \in [\![(\nabla\cup\nabla', (W_1\ B)\cdot s)]\!]$, for some such σ, and $(\lambda B.X_1)\rho_s \sim \lambda B\sigma.(W_1\sigma\ B\sigma)\cdot s\sigma$. Then,

    $$\begin{aligned}(X\circ\theta')\rho_s &= C'[\lambda B.X_1]\rho_s, \text{ for } C[] = C'[\lambda B.[]]\\&= C'[X_0\rho'_s], \text{ where } [X_0\rho'_s]_\sim \in [\![(\nabla\cup\nabla', \lambda W_1.s)]\!]\\&= (X\theta)\rho'_s, \text{ for some ground } \rho'_s.\end{aligned}$$

    Then, $[(X\theta')\rho_s]_\sim \in [\![(\{X_0 : \lambda W_1.s \triangleq \lambda W_2.t\},\emptyset,\nabla,\theta)]\!]$, and the result follows.

  The proof that $[(X\circ\theta')\rho_t]_\sim \in [\![(\{X_0 : \lambda W_1.s \triangleq \lambda W_2.t\},\emptyset,\nabla,\theta)]\!]$ is analogous. ◀

▶ **Theorem A.1** (cf.Theorem 3.9). *Given* $\mathtt{NL}_A$ *expressions* s *and* t*, and a freshness context* ∇*. If* (∇', r) *is a generalization of* (∇, s) *and* (∇, t)*, then there exists a* ∇'' *and a derivation* $(\{X : s \triangleq t\},\emptyset,\nabla,[]) \Longrightarrow^* (\emptyset, M, \Delta, \theta)$ *computed by* AtomAntiUnif*, such that* $[\![(\Delta\cup\nabla'', X\circ\theta)]\!]$ *is a generalization of* (∇, s) *and* (∇, t)*, and* $[\![(\emptyset, M, \Delta\cup\nabla'', \theta)]\!] \subseteq [\![(\nabla', r)]\!]$.

Proof. The proof is by induction on r by investigating the rules from AtomAntiUnif algorithm (Figure 1) used. We show some cases below.

1. $r = \pi \cdot Y'$ (a suspension of a generalization variable) There are three possible cases:

   a. $s = \pi_1 \cdot Y$, $t = \pi_2 \cdot Y$ and the rule (SusYY) was applied.
   Then,
   $$\frac{(\{X : \pi_1 \cdot Y \triangleq \pi_2 \cdot Y\}, \emptyset, \nabla, []) \qquad \nabla \vDash \pi_1 = \pi_2}{(\emptyset, \emptyset, \nabla, \{X \mapsto \pi_1 \cdot Y\})}\ (\texttt{SusYY})$$
   Since, by hypothesis, $(\nabla', \pi \cdot Y)$ is a generalization of $(\nabla, \pi_1 \cdot Y)$ and $(\nabla, \pi_2 \cdot Y)$, we have that $[\![(\nabla, \pi_1 \cdot Y)]\!] = [\![(\nabla, \pi_2 \cdot Y)]\!] \subseteq [\![(\nabla', \pi \cdot Y')]\!]$.
   By Definition 3.7, $[\![(\emptyset, \emptyset, \nabla, \{X \mapsto \pi_1 \cdot Y\})]\!] = [\![(\nabla, \pi_1 \cdot Y)]\!]$ and the result follows trivially for $\Delta = \nabla$ and $\nabla'' = \emptyset$.

   b. $s : \pi_1 \cdot Y$, $t = \pi_2 \cdot Y$ and the rule (SolveYY) was applied.
   Then $\nabla \nvDash \pi_1 = \pi_2$ and this case follows a similar reasoning used for (SolveAB).

   c. s and t are such that the rule (Solve) was applied.
   Then, $(\{X : s \triangleq t\}, \emptyset, \nabla, []) \Rightarrow_{(\texttt{Solve})} (\emptyset, \{X : s \triangleq t\}, \nabla, [])$ where $Head(s) \neq Head(t)$ and s, t are not both atom-variables. By hypothesis, $(\nabla', \pi \cdot Y')$ is a generalization for (∇, s) and (∇, t), i.e., $[\![(\nabla, s)]\!] \subseteq [\![(\nabla', \pi \cdot Y')]\!]$ and $[\![(\nabla, t)]\!] \subseteq [\![(\nabla', \pi \cdot Y')]\!]$. By soundness (∇, X) is a generalization of (∇, s) and (∇, t). In addition, $[\![(\emptyset, \{X : s \triangleq t\}, \nabla, [])]\!] = [\![(\nabla, s)]\!] \cup [\![(\nabla, t)]\!] \subseteq [\![(\nabla', \pi \cdot Y')]\!]$. The result follows trivially for $\Delta = \nabla$ and $\nabla'' = \emptyset$.

2. $r = \lambda W.r'$
   Then, $s = \lambda W_1.s'$ and $t = \lambda W_2.t'$, and the following holds:
   $$[\![(\nabla, \lambda W_1.s')]\!] \subseteq [\![(\nabla', \lambda W.r')]\!] \quad \text{and} \quad [\![(\nabla, \lambda W_2.t')]\!] \subseteq [\![(\nabla', \lambda W.r')]\!].$$
   In addition, (∇', r') is a generalization of (∇, s') and (∇, t'). By induction hypothesis, there exist ∇'' and a derivation $(\{X' : s' \triangleq t'\}, \emptyset, \nabla, []) \Longrightarrow^* (\emptyset, M, \Delta, \theta)$ such that $(\Delta \cup \nabla'', X' \circ \theta)$ is a generalization of (∇, s') and (∇, t'), and $[\![(\emptyset, M, \Delta \cup \nabla'', \theta)]\!] \subseteq [\![(\nabla', r')]\!]$. Notice that with such s and t we can apply the AtomAntiUnif algorithm as follows
   $$(\{X{:}\lambda W_1.s' \triangleq \lambda W_2.t'\}, \emptyset, \nabla, []) \Longrightarrow (\{Y{:}(W_1\ B) \cdot s' \triangleq (W_2\ B) \cdot t'\}, \emptyset, \nabla \cup \nabla_B, \{X \mapsto \lambda B.Y\})$$
   where Y is a fresh variable, B is a fresh atom-variable and $\nabla_B = \{B \# \lambda W_1.s', B \# \lambda W_2.t'\}$. Notice that s' and $(W_1\ B) \cdot s'$ are structurally the same term, with W_1 renamed to a fresh atom-variable B. The same with t' and $(W_2\ B) \cdot t'$. Then,
   $$(\{Y : (W_1\ B) \cdot s' \triangleq (W_2\ B) \cdot t'\}, \emptyset, \nabla \cup \nabla_B, []) \Longrightarrow^* (\emptyset, M', \Delta' \cup \nabla_B, \theta')$$
   where M', θ', Δ' are versions of M, θ and Δ with W_1 and W_2 renamed to B.

   $$\begin{aligned}(\{X{:}\lambda W_1.s' \triangleq \lambda W_2.t'\}, \emptyset, \nabla, []) &\Longrightarrow (\{Y{:}(W_1\ B) \cdot s' \triangleq (W_2\ B) \cdot t'\}, \emptyset, \nabla \cup \nabla_B, \{X \mapsto \lambda B.Y\}) \\ &\Longrightarrow^* (\emptyset, M', \Delta' \cup \nabla_B, \{X \mapsto \lambda B.Y\} \cup \theta')\end{aligned}$$
   Then $(\Delta' \cup \nabla_B, X \circ \theta'')$ is a generalization of $(\nabla, \lambda W_1.s')$ and $(\nabla, \lambda W_2.t')$.

   ▷ Claim. $[\![(\emptyset, M', \Delta' \cup \nabla_B \cup \nabla'', \theta'')]\!] \subseteq [\![(\nabla', \lambda W.r')]\!]$.

In fact, if $[(X\theta'')\rho]_\sim \in [\![(\emptyset, M', \Delta' \cup \nabla_B \cup \nabla'', \theta'')]\!]$ then ρ is ground, $(\Delta' \cup \nabla_B \cup \nabla'')\rho$ holds, and ρ acts on the variables in the range of θ' accordingly to M'. Notice that $(X\theta'') = (\lambda B.Y)\theta' = \lambda(B\theta').Y\theta'$. Then, $[(X\theta'')\rho]_\sim = [\lambda(B\theta')\rho.(Y\theta')\rho]_\sim$, where $[(Y\theta')\rho]_\sim \in [\![(\emptyset, M', \Delta' \cup \nabla'', \theta')]\!] \subseteq [\![(\nabla', r')]\!]$, by the IH. Therefore, $(X \circ \theta'')\rho \sim \lambda b.r^*$ where $r^* \sim r'\sigma$, for some σ ground such that $\nabla'\sigma$ holds, $[(X\theta')\rho]_\sim \in [\![(\nabla', \lambda W.r')]\!]$, and the result follows.

3. $r = f(r_1, \ldots, r_n)$.
Then $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and the following hold: $[\![(\nabla, f(s_1, \ldots, s_n))]\!] \subseteq [\![(\nabla', r)]\!]$ and $[\![(\nabla, f(t_1, \ldots, t_n))]\!] \subseteq [\![(\nabla', r)]\!]$. In addition, (∇', r_i) is a generalization of (∇, s_i) and (∇, t_i) for each $i = 1, \ldots, n$. By the IH, there exist ∇_i'' and derivations $(\{X_i : s_i \triangleq t_i\}, \emptyset, \nabla, []) \Longrightarrow^* (\emptyset, M_i, \Delta_i, \theta_i)$ s.t., for each $i = 1, \ldots, n$, the following hold:

a. $(\Delta_i \cup \nabla_i'', X_i\theta_i)$ is a generalization of (∇, s_i) and (∇, t_i), i.e., $[\![(\nabla, s_i)]\!] \subseteq [\![(\Delta_i \cup \nabla_i'', X_i\theta_i)]\!]$ and $[\![(\nabla, t_i)]\!] \subseteq [\![(\Delta_i \cup \nabla_i'', X_i\theta_i)]\!]$.

b. and $[\![(\emptyset, M_i, \Delta_i \cup \nabla_i'', \theta_i)]\!] \subseteq [\![(\nabla', r_i)]\!]$.

Now we need to combine these semantics to obtain the final result. We want to prove that there exists ∇'' and an AtomAntiUnif computation

$$\begin{aligned}(\{X : f(s_1, \ldots, s_n) \triangleq f(t_1, \ldots, t_n)\}, \emptyset, \nabla, []) &\Longrightarrow_{(\mathsf{Dec})} (\Gamma_1, \emptyset, \nabla, \{X \mapsto f(X_1, \ldots, X_n)\}) \\ &\Longrightarrow^* (\emptyset, M, \Delta, \theta)\end{aligned}$$

where $\Gamma_1 = \{X : f(s_1, \ldots, s_n) \triangleq f(t_1, \ldots, t_n)\}$ and $\theta = \{X \mapsto f(X_1, \ldots, X_n)\} \cup \theta'$ s.t. $(\Delta \cup \nabla'', X \circ \theta)$ is a generalization of (∇, s) and (∇, t), and $[\![(\emptyset, M, \Delta \cup \nabla'', \theta)]\!] \subseteq [\![(\nabla', r)]\!]$.

▷ Claim. $(\nabla \cup \bigcup_i (\Delta_i \cup \nabla_i''), f(X_1\theta_1, \ldots, X_n\theta_n))$ is a generalization of (∇, s) and (∇, t).

There are no critical clash among our rules (in applications in the same constraint). Thus, from state $(\{X_1 : s_1 \triangleq t_1, \ldots, X_n : s_n \triangleq t_n\}, \emptyset, \nabla, \{X \mapsto f(X_1, \ldots, X_n)\})$ we can choose to simplify first X_1, then X_2, then ... until we reach X_n.

$$\begin{gathered}(\{X_1 : s_1 \triangleq t_1, \ldots, X_n : s_n \triangleq t_n\}, \emptyset, \nabla, \{X \mapsto f(X_1, \ldots, X_n)\}) \\ \vdots \\ (\{X_2 : s_2 \triangleq t_2, \ldots, X_n : s_n \triangleq t_n\}, M_1, \nabla \cup \Delta_1, \{X \mapsto f(X_1\theta_1, \ldots, X_n\theta_1)\}) \\ \vdots \\ (\{X_3 : s_3 \triangleq t_3, \ldots, X_n : s_n \triangleq t_n\}, M_1 \cup M_2, \nabla \cup \Delta_1 \cup \Delta_2, \{X \mapsto f(X_1\theta_1, X_2\theta_2 \ldots, X_n\theta_n)\}) \\ \vdots \\ (\emptyset, \bigcup_i M_i, \nabla \cup \bigcup_i \Delta_i, \{X \mapsto f(X_1\theta_1, \ldots, X_n\theta_n)\})\end{gathered}$$

From soundness $(\nabla \cup \bigcup_i \Delta_i, f(X_1\theta_1, \ldots, X_n\theta_n))$ is a generalization of (∇, s) and (∇, t). In addition, $[\![(\nabla \cup \bigcup_i (\Delta_i \cup \nabla_i''), f(X_1\theta_1, \ldots, X_n\theta_n))]\!] \subseteq [\![(\nabla \cup \bigcup_i \Delta_i, f(X_1\theta_1, \ldots, X_n\theta_n))]\!]$. It is straightforward to check that $[\![(\nabla \cup \bigcup_i (\Delta_i \cup \nabla_i''), f(X_1\theta_1, \ldots, X_n\theta_n))]\!]$ is also a generalization of (∇, s) and (∇, t). We take $\theta := \{X \mapsto f(X_1, \ldots, X_n)\} \cup \theta_1 \cup \ldots \cup \theta_n$, $M = \bigcup_i M_i$, $\Delta = \bigcup_i \Delta_i \cup \nabla$ and $\nabla'' = \bigcup_i \nabla_i''$.

▷ Claim. $[\![(\emptyset, M, \Delta \cup \nabla'', \theta)]\!] \subseteq [\![(\nabla', r)]\!]$.

Let $[(X\theta)\rho]_\sim \in [\![(\emptyset, M, \Delta \cup \nabla'', \theta)]\!]$. Then ρ is ground, $(\Delta \cup \nabla'')\rho$ holds, and $[(X\theta)\rho]_\sim = [(f((X_1\theta_1)\rho, \ldots, (X_n\theta_n)\rho)]_\sim$ where, for each i, $[(X_i\theta_i)\rho]_\sim \in [\![(\emptyset, M_i, \Delta_i \cup \nabla_i'', \theta_i)]\!] \subseteq [\![(\nabla', r_i)]\!]$, from item (b) of the IH. Then, $[(f((X_1\theta_1)\rho, \ldots, (X_n\theta_n)\rho)]_\sim \in [\![(\nabla', f(r_1, \ldots, r_n))]\!] = [\![(\nabla', r)]\!]$, and the result follows. ◀

▶ **Proposition A.2** (cf. Proposition 3.8). *The algorithm* AtomAntiUnif *never get stuck and will yield a generalization. The number of rule applications is linear.*

Proof. The claim can be shown by a measure that is the sum of 2∗(size of Γ)+ size of M, where the permutations are ignored in the size. The steps (`Dec`) and (`Abs`) strictly decrease the size, and are always applicable to the generalization triples of type (f, f), and $\lambda, \lambda)$. (`Solve`) also strictly decreases the size. (`SolveAB`) and (`SusA`) are complementary and remove $W_1 \triangleq W_2$ triples. (`SolveYY`) and (`SusYY`) are also complementary and remove $\pi_1 \cdot Y \triangleq \pi_2 \cdot Y$ triples. The other rule is (`Merge`) which removes a triple in M. ◀

▶ **Lemma A.3** (cf. Lemma 4.6). *Let $\mathcal{A} = \{A, B\}$ be a set of atom-variables, X and Y be generalization variables. The* Eqr*- freshness context $((A \neq B) \implies A\#X) \wedge ((A = B) \implies A\#Y \wedge B\#Y)$ cannot be encoded as an $\mathtt{NL}_A$-freshness context, i.e, a conjunction of freshness constraints.*

Proof. Suppose, by contradiction, that $\{A\#s_1, \ldots, A\#s_n, B\#t_1, \ldots, B\#t_m\}$ is the freshness context that is the encoding of the Eqr-freshness context. Then the evaluation mechanism must construct the Eqr-freshness context above (up to some modifications for $A = B$).

- If $A = B$, then for every freshness constraint of the form $A\#\lambda\pi \cdot W.s'$, the result will be $A\#\lambda A. \ldots$, since all binders will become equal to A, hence the constraint evaluates to `True`. We also derive that the terms s_i, t_i in the constraints that evaluate to the desired freshness constraints $A\#X, B\#Y$ do not contain abstractions.
- Let $C\#\pi \cdot Y$ (for $C \in \{A, B\}$) be the constraint that evaluates under $(A = B)$ to $B\#Y$. If $A \neq B$ then evaluating $C\#\pi \cdot Y$ will result in either $A\#Y$ or $B\#Y$, which are both not part of the results for $(A \neq B)$ in the generalized freshness context above.

Hence there is no such encoding. ◀