Nominal Anti-Unification

Alexander Baumgartner\(^1\), Temur Kutsia\(^1\), Jordi Levy\(^2\) and Mateu Villaret\(^3\)

\(^1\) Research Institute for Symbolic Computation, Johannes Kepler University Linz, Austria
\(^2\) Artificial Intelligence Research Institute, Spanish Council for Scientific Research (IIIA-CSIC), Barcelona, Spain
\(^3\) Departament d’Informàtica i Matemàtica Aplicada, Universitat de Girona, Spain

1 Introduction

Equation solving between nominal terms has been investigated by several authors, who designed and analyzed algorithms for nominal unification \(^3\)\(^4\)\(^15\)\(^26\), nominal matching \(^5\), equivariant unification \(^6\), permissive nominal unification \(^8\). However, in contrast to unification, its dual problem, anti-unification, has not been studied for nominal terms previously. In \(^18\), it is referred to as “the as-of-yet undiscovered nominal anti-unification”, which “could form a fundamental component of a refactoring tool” for \(\alpha\)Prolog \(^7\) programs.

Software refactoring is one of possible applications of anti-unification. This method, formulated for different theories, has been successfully used in inductive logic programming \(^17\), cognitive modeling \(^24\), analogy making \(^13\), inductive program synthesis \(^12\), proof generalization \(^27\), mathematical reasoning \(^10\)\(^11\), etc. Nominal anti-unification can play a role in extending some of these applications to the nominal setting. For instance, it can be useful to generalize proofs done in nominal logic, or in doing analogical reasoning in mathematics, or in adapting inductive program synthesis methods to \(\alpha\)Prolog programs, etc.

The anti-unification problem for two terms \(t_1\) and \(t_2\) is concerned with finding a generalization term \(t\) such that \(t_1\) and \(t_2\) are substitutive instances of \(t\). The interesting generalizations are the least general ones (lgg). Plotkin \(^20\) and Reynolds \(^22\) initiated research on anti-unification in the 1970s, developing algorithms for first-order terms. Since then, anti-unification has been studied in various theories, including some of those with binding constructs: calculus of constructions \(^19\), \(\mathcal{M}\lambda\) \(^9\), second-order lambda calculus with type variables \(^16\), simply-typed lambda calculus where generalizations are higher-order patterns \(^2\), just to name a few.

In this paper we address the problem of computing lgg’s for nominal terms-in-context, which are pairs of a freshness context and a nominal term. It turned out that without a restriction, there is no lgg for terms-in-context, in general. Therefore we restrict the set of atoms which are permitted in generalizations to be finite. In this case, there exists a single lgg (modulo \(\sim\)) and we design an algorithm to compute it. Computation of nominal lgg’s requires a solution to the equivariance problem which aims at finding a permutation of atoms \(\pi\) for given terms \(t_1\) and \(t_2\) such that \(\pi\) applied to \(t_1\) is \(\alpha\)-equivalent to \(t_2\) (under a given freshness context).

Various anti-unification techniques, such as first-order, higher-order, or equational anti-unification have been used in inductive logic programming, logical and relational learning \(^21\), reasoning by analogy \(^13\), program synthesis \(^23\), program verification \(^16\), etc. Nominal anti-unification can, hopefully, contribute in solving similar problems in nominal setting.

The anti-unification algorithm has been implemented and is available from \url{www.risc.jku.at/projects/stout/software/nau.php}. The implementation of the equivariance algorithm is also accessible separately from \url{www.risc.jku.at/projects/stout/software/nequiv.php}. 
2 Nominal Terms

Nominal terms contain variables and atoms. Variables can be instantiated and atoms can be bound. We have sorts of atoms \( \nu \) and sorts of data \( \delta \) as disjoint sets. Atoms \( (a, b, \ldots) \) have one of the sorts of atoms. Variables \( (X, Y, \ldots) \) have a sort of atom or data. Nominal function symbols \( (f, g, \ldots) \) have an arity of the form \( \tau_1 \times \cdots \times \tau_n \to \delta \), where \( \delta \) is a sort of data and \( \tau_i \) are sorts given by the grammar \( \tau ::= \nu \mid \delta \mid (\nu)\tau \). Abstractions have sorts of the form \( (\nu)\tau \).

A swapping \( (a \ b) \) is a pair of atoms of the same sort. A permutation is a sequence of swappings. We use \( \pi, \rho \) to denote permutations. Nominal terms \( (t, s, r) \) are given by the grammar:

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

The effect of swapping, and permutation application are defined in the standard way. The inverse of a permutation \( \pi = (a_1 b_1) \ldots (a_n b_n) \) is the permutation \( (a_n b_n) \ldots (a_1 b_1) \), denoted by \( \pi^{-1} \). We use \( \text{Id} \) for the empty permutation and write \( X \) as the shortcut of \( \text{Id} \cdot X \).

For a set \( A \), we denote by \( |A| \) its cardinality. The set of atoms of a term \( t \) or a permutation \( \pi \) is the set of all atoms which appear in it and is denoted by \( \text{Atoms}(t) \), \( \text{Atoms}(\pi) \) respectively. \( ||t||_{\text{abs}} \) stand for the number of abstraction occurrences in \( t \).

Suspensions are uses of variables with a permutation of atoms waiting to be applied once a variable is instantiated. Occurrences of an atom \( a \) are said to be bound if they are in the scope of an abstraction of \( a \), otherwise are said to be free. We denote by \( \text{FA}(t) \) the set of all atoms which occur freely in \( t \): \( \text{FA}(f(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} \text{FA}(t_i) \), \( \text{FA}(a) = \{a\} \), \( \text{FA}(a.t) = \text{FA}(t) \setminus \{a\} \), and \( \text{FA}(\pi \cdot X) = \text{Atoms}(\pi) \). \( \text{FA}^*(t) \) is the set of all atoms which occur freely in \( t \) ignoring suspensions: \( \text{FA}^*(f(t_1, \ldots, t_n)), \text{FA}^*(a), \text{FA}^*(a.t) \) are defined like above but \( \text{FA}^*(\pi \cdot X) = \emptyset \).

Substitutions, denoted by \( \sigma \), are defined in the standard way, and their application allows atom capture, for instance, \( a.X \{$X \mapsto a$} = a.a \). The identity substitution is denoted by \( \varepsilon \).

A freshness constraint is a pair of the form \( a \# X \) stating that the instantiation of \( X \) cannot contain free occurrences of \( a \). A freshness context is a finite set of freshness constraints. We will use \( \nabla \) and \( \Gamma \) for freshness contexts. \( \text{Atoms}(\nabla) \) denotes the set of atoms of \( \nabla \).

We say that a substitution \( \sigma \) respects \( \nabla \), if for all \( X, \text{FA}^*(X\sigma) \cap \{a \mid a \# X \in \nabla\} = \emptyset \).

The predicate \( \approx \) stands for \( \sigma \)-equivalence and was defined in [25,26] by the following theory:

\[
\begin{align*}
\nabla \vdash a & \equiv a & \nabla \vdash a.t \equiv a.t & \quad a \neq a' \quad \nabla \vdash t \equiv (a.a') \cdot t' & \quad \nabla \vdash a \# t' \\
& \quad a \# X \in \nabla \quad \text{for all } a \text{ such that } \pi \cdot a \neq \pi' \cdot a & \\
\n\nabla \vdash \pi \cdot X & \equiv \pi' \cdot X & \\
\end{align*}
\]

where the freshness predicate \( \# \) is defined by

\[
\begin{align*}
\nabla \vdash a \neq a' & \quad (\pi^{-1} \cdot a \# X) \in \nabla & \quad \nabla \vdash a.t_1 & \cdots & \quad \nabla \vdash a \# t_n \\
\n\nabla \vdash a \# t_1 & & \cdots & \quad \nabla \vdash a \# f(t_1, \ldots, t_n) & \\
\n\nabla \vdash a \# a.\pi & \quad \nabla \vdash a \# a.t & \\
\end{align*}
\]

Given a freshness context \( \nabla \) and a substitution \( \sigma \), we define \( \nabla \sigma \) as the minimal (with respect to \( \subseteq \)) freshness context such that for all \( a \# X \in \nabla \) holds \( \nabla \sigma \vdash a \# X.\sigma \). It can easily be derived from the definition of the freshness predicate, if it exists. Otherwise it is undefined.

**Theorem 1.** \( \sigma \) respects \( \nabla \) iff \( \nabla \sigma \) is defined.

A term-in-context, denoted by \( p \), is a pair \( (\nabla, t) \) of a freshness context and a term. \( (\nabla, t) \) is more general than a term-in-context \( (\Gamma, s) \), written \( (\nabla, t) \preceq (\Gamma, s) \), if there is a substitution \( \sigma \), which respects \( \nabla \), such that \( \nabla \sigma \subseteq \Gamma \) and \( \Gamma \vdash t \sigma \approx s \). We write \( \nabla \vdash t \preceq s \) if there exists a substitution \( \sigma \) such that \( \nabla \vdash t \sigma \approx s \). We also write \( \nabla \vdash t \equiv s \) if \( \nabla \vdash t \preceq s \) and \( \nabla \vdash s \preceq t \).
Example 1. We give some examples to demonstrate the relations we have just defined:

- \(\{a\#X\}, f(a)\) \(\simeq\) \(\emptyset, f(a)\). We can use \(X \mapsto b\) as substitution applied to the first pair.
- \(\emptyset, f(X)\) \(\preceq\) \(\{a\#Y\}, f(Y)\) with \(\sigma = \{X \mapsto Y\}\), but not \(\{a\#Y\}, f(Y)\) \(\preceq\) \(\emptyset, f(X)\).
- \(\{a\#X\}, f(X)\) \(\not\preceq\) \(\{a\#X\}, f(a)\). Notice that \(\sigma = \{X \mapsto a\}\) does not respect \(\{a\#X\}\).
- \(\{b\#X\}, (a\cdot b)\cdot X\) \(\preceq\) \(\{c\#X\}, (a\cdot c)\cdot X\) with the substitution \(\sigma = \{X \mapsto (a\cdot b)(a\cdot c)\cdot X\}\).

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

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

Example 2. Let \(p_1 = \langle \emptyset, a_1 \rangle\) and \(p_2 = \langle \emptyset, a_2 \rangle\) be two terms-in-context. Then in any complete set of generalizations of \(p_1\) and \(p_2\) there is an infinite chain \(\emptyset, X\prec\{a_3\#X\}, X\prec\{a_3\#X, a_4\#X\}, X\prec\cdots\), where \(a_1, a_2, a_3, \ldots\) is the set of all atoms of the language. Hence, \(p_1\) and \(p_2\) do not have a minimal complete set of generalizations.

Theorem 2. The problem of anti-unification for terms-in-context is of nullary type.

However, if we restrict the set of atoms which can be used in the generalizations to be finite, then the anti-unification problem becomes unitary.

We say that a term \(t\) (resp., a freshness context \(\nabla\)) is based on a set of atoms \(A\) iff \(\text{Atoms}(t) \subseteq A\) (resp., \(\text{Atoms}(\nabla) \subseteq A\)). A term-in-context \(\langle \nabla, t \rangle\) is based on \(A\) if both \(t\) and \(\nabla\) are based on it. A permutation is \(A\)-based if it contains only atoms from \(A\). An \(A\)-based lgg of \(A\)-based terms-in-context \(p_1\) and \(p_2\) is an \(A\)-based term-in-context \(p\) which is a generalization of \(p_1\) and \(p_2\) and there is no \(A\)-based generalization \(p'\) of \(p_1\) and \(p_2\) which satisfies \(p \prec p'\).

3 Nominal Anti-Unification Algorithm

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

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

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

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

The anti-unification algorithm is formulated in a rule-based way and depends on two global parameters, a finite set of atoms \(A\) and a freshness context \(\nabla\). It works on tuples of the form \(P; S; \Gamma; \sigma\), where \(P\) and \(S\) are sets of AUEs, \(\Gamma\) is a freshness context and \(\sigma\) is a substitution. \(P\), \(S\), \(\nabla\), and \(\Gamma\) are \(A\)-based and \(\nabla\) does not constrain generalization variables. Furthermore if X : t \(\triangleq\) s \(\in\) \(P \cup S\), then this is the sole occurrence of X in \(P \cup S\). The rules are the following:

\(^1\)The definition of minimal complete sets of generalizations is standard. For a precise definition, see, e.g. [14].
Nominal Anti-Unification

Baumgartner, Kutsia, Levy, and Villaret

Dec: Decomposition

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

where \(h\) is a function symbol or an atom, \(Y_1, \ldots, Y_m\) are fresh variables of appropriate sorts.

Abs: Abstraction

\{X : a.t \triangleq b.s\} \cup P; S; \Gamma; \sigma \implies \{Y : (c\ a).t \triangleq (c\ b).s\} \cup P; S; \Gamma; \sigma\{X \mapsto c\cdot Y\},

where \(Y\) is fresh, \(c \in A\) such that \(\nabla \vdash c\#a\cdot t\) and \(\nabla \vdash c\#b\cdot s\).

Sol: Solving

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

if neither Dec nor Abs is applicable, where \(\Gamma' = \{a\#X\mid a \in A \land \nabla \vdash a\#t \land \nabla \vdash a\#s\}\).

Mer: Merging

\[\begin{align*}
P; & \{X : t_1 \triangleq s_1, Y : t_2 \triangleq s_2\} \cup S; \Gamma; \sigma \implies \cr & P; \{X : t_1 \triangleq s_1\} \cup S; \Gamma\{Y \mapsto \pi \cdot X\}; \sigma\{Y \mapsto \pi \cdot X\},
\end{align*}\]

where \(\pi\) is an \(A\)-based permutation such that \(\nabla \vdash \pi \cdot t_1 \approx t_2\), and \(\nabla \vdash \pi \cdot s_1 \approx s_2\).

Given a finite set of atoms \(A\), an \(A\)-based freshness context \(\nabla\), and two nominal \(A\)-based terms \(t\) and \(s\), to compute an \(A\)-based generalization of \(\langle \nabla, t \rangle\) and \(\langle \nabla, s \rangle\), we start with \(\{X : t \triangleq s\}; \emptyset; \emptyset; \varepsilon\), where \(X\) is a fresh variable, and apply the rules (don’t care) nondeterministically as long as possible. We denote this procedure by \(\mathcal{N}\), and say that the final state is reached when no more rule is applicable. The final state is of the form \(\emptyset; S; \Gamma; \sigma\), where Mer does not apply to \(S\) and we say that the result computed by \(\mathcal{N}\) is \((\Gamma, X\sigma)\).

Note that the Dec rule works also for the AUEs of the form \(X : a \triangleq a\). In the Abs rule, it is important to have the corresponding \(c\) in \(A\). If not then the Sol rule takes over.

Example 3. We illustrate \(\mathcal{N}\) on a couple of examples:

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

\[
\{X : f(a, b) \triangleq f(b, c)\}; \emptyset; \emptyset; \varepsilon \implies_{\text{Dec}}
\{Y : a \triangleq b, Z : b \triangleq c\}; \emptyset; \emptyset; \{X \mapsto f(Y, Z)\} \implies_{\text{Sol}}^2
\emptyset; \{Y : a \triangleq b, Z : b \triangleq c\}; \{c\#Y, d\#Y, a\#Z, d\#Z\}; \{X \mapsto f(Y, Z)\} \implies_{\text{Mer}}
\emptyset; \{Y : a \triangleq b\}; \{c\#Y, d\#Y\}; \{X \mapsto f(Y, (a\cdot b\cdot c)\cdot Y)\}
\]

Hence, \(p = \{c\#Y, d\#Y\}, f(Y, (a\cdot b\cdot c)\cdot Y)\) is the computed result. It generalizes the input pairs: \(p\{Y \mapsto a\} \preceq (\nabla, t)\) and \(p\{Y \mapsto b\} \preceq (\nabla, s)\). The substitutions \(\{Y \mapsto a\}\) and \(\{Y \mapsto b\}\) can be read from the final store. Note that \(\{c\#Y\}, f(Y, (a\cdot b\cdot c)\cdot Y)\) would be also an \(A\)-based generalization of \(\langle \nabla, t \rangle\) and \(\langle \nabla, s \rangle\), but it is strictly more general than \(p\).

- Let \(t = f(b, a)\), \(s = f(Y, (a\cdot b)\cdot Y)\), \(\nabla = \{b\#Y\}\), and \(A = \{a, b\}\). Then \(\mathcal{N}\) computes the term-in-context \(\emptyset; f(Z_1, (a\cdot b)\cdot Z_1)\) which generalizes the input pairs.

- Let \(t = f(a\cdot b, X)\), \(s = f(b\cdot a, Y)\), \(\nabla = \{c\#X\}\), and \(A = \{a, b, c, d\}\). Then \(\mathcal{N}\) computes the term-in-context \(p = \{c\#Z_1, d\#Z_1, f(c\cdot Z_1, Z_2)\}\) which generalizes the input: \(p\{Z_1 \mapsto b, Z_2 \mapsto X\} = \emptyset; f(c\cdot a, Y)\) \preceq (\nabla, t)\) and \(p\{Z_1 \mapsto a, Z_2 \mapsto Y\} = \emptyset; f(c\cdot a, Y)\) \preceq (\nabla, s)\).

Theorem 3. Let \(t, s\) be terms and \(\nabla, \Gamma\) be freshness contexts, all based on a finite atoms set \(A\).

- Termination: The procedure \(\mathcal{N}\) terminates on any input.
Nominal Anti-Unification

Baumgartner, Kutsia, Levy, and Villaret

• Soundness: If \( \{X : t \triangleq s\} ; \emptyset ; \emptyset ; \varepsilon \Rightarrow^+ \emptyset ; S ; \Gamma ; \sigma \) is a derivation obtained by an execution of \( \mathcal{N} \), then \( (\Gamma, X\sigma) \) is an A-based generalization of \( (\nabla, t) \) and \( (\nabla, s) \).

• Completeness: If \( (\Gamma, r) \) is an A-based generalization of \( (\nabla, t) \) and \( (\nabla, s) \), then there exists a derivation \( \{X : t \triangleq s\} ; \emptyset ; \emptyset ; \varepsilon \Rightarrow^+ \emptyset ; S; \Gamma' ; \sigma \) obtained by an execution of \( \mathcal{N} \), such that \( (\Gamma, r) \preceq (\Gamma', X\sigma) \).

• Uniqueness: Let \( \{X : t \triangleq s\} ; \emptyset ; \emptyset ; \varepsilon \Rightarrow^+ \emptyset ; S_1; \Gamma_1 ; \sigma_1 \) and \( \{X : t \triangleq s\} ; \emptyset ; \emptyset ; \varepsilon \Rightarrow^+ \emptyset ; S_2; \Gamma_2 ; \sigma_2 \) be two maximal derivations in \( \mathcal{N} \). Then \( (\Gamma_1, X\sigma_1) \simeq (\Gamma_2, X\sigma_2) \).

Now we study how lgg’s of terms-in-context depend on the set of atoms they are based on.

**Lemma 1.** Let \( A_1 \) and \( A_2 \) be two finite sets of atoms with \( A_1 \subseteq A_2 \) such that the \( A_1 \)-based terms-in-context \( (\nabla, t) \) and \( (\nabla, s) \) have an \( A_1 \)-based lgg \( (\Gamma_1, r_1) \) and an \( A_2 \)-based lgg \( (\Gamma_2, r_2) \). Then \( \Gamma_2 \vdash r_1 \preceq r_2 \).

In general, we can not replace \( \Gamma_2 \vdash r_1 \preceq r_2 \) with \( \Gamma_2 \vdash r_1 \simeq r_2 \) in Lemma 1. Consider for instance the example \( t = ab, s = ba, \nabla = \emptyset, A_1 = \{a, b\}, \) and \( A_2 = \{a, b, c\} \). Then for \( (\nabla, t) \) and \( (\nabla, s) \), \( (a, X) \) is an \( A_1 \)-based lgg and \( (\{c#X\}, c.X) \) is an \( A_2 \)-based lgg. Obviously, \( \{c#X\} \vdash X \preceq c.X \) but not \( \{c#X\} \vdash c.X \preceq X \).

We say that a set of atoms \( A \) is saturated for \( A \)-based \( t, s, \nabla \), if \( |A \setminus (\text{Atoms}(t) \cup \text{Atoms}(s) \cup \text{Atoms}(\nabla))| \geq \min\{|\|t\|_{\text{Abs}}, |\|s\|_{\text{Abs}}\|\} \).

**Lemma 2.** Under the conditions of Lemma 1 if \( A_1 \) is saturated for \( t, s, \nabla \), then \( \Gamma_2 \vdash r_1 \simeq r_2 \).

### 4 Deciding Equivalence

Computation of \( \pi \) in the condition of the rule Mer above requires an algorithm that solves the following problem: Given a finite set of atoms \( A \), terms \( t \) and \( s \), and a freshness context \( \nabla \), all based on \( A \), find an \( A \)-based permutation \( \pi \) such that \( \nabla \vdash \pi\cdot t \approx s \). This is the problem of deciding whether \( t \) and \( s \) are equivariant with respect to \( \nabla \) and \( A \).

We describe a rule-base algorithm, which we call \( \mathcal{E} \), that solves this problem by effectively computing the corresponding permutation. It works on tuples of the form \( E; \nabla; A; \pi \) (called systems). \( E \) is a set of equivariance equations of the form \( t \approx s \) where \( t, s \) are nominal terms. \( \nabla \) is a freshness context, and \( A \) is a finite set of atoms which are available for computing \( \pi \). The latter holds the permutation to be returned in case of success.

The algorithm is split into two phases. In phase 1, function applications, abstractions, and suspensions are decomposed as long as possible. Phase 2 is the permutation computation.

**Phase 1 – Dec-E: Decomposition**
\[
\{f(t_1, \ldots, t_m) \approx f(s_1, \ldots, s_m)\} \cup E; \nabla; A; Id \Rightarrow \{t_1 \approx s_1, \ldots, t_m \approx s_m\} \cup E; \nabla; A; Id.
\]

**Phase 1 – Alp-E: Alpha Equivalence**
\[
\{a.t \approx b.s\} \cup E; \nabla; A; Id \Rightarrow \{(a \cdot c).t \approx (b \cdot c).s\} \cup E; \nabla; A; Id,
\]
where \( c \) is a fresh atom of the same sort as \( a \) and \( b \).

**Phase 1 – Sus-E: Suspension**
\[
\{\pi_1.X \approx \pi_2.X\} \cup E; \nabla; A; Id \Rightarrow \{\pi_1.a \approx \pi_2.a \mid a \in A \land a#X \notin \nabla\} \cup E; \nabla; A; Id.
\]

**Phase 2 – Rem-E: Remove**
\[
\{a \approx b\} \cup E; \nabla; A; \pi \Rightarrow E; \nabla; A \setminus \{b\}; \pi, \quad \text{if } \pi\cdot a = b.
\]

**Phase 2 – Sol-E: Solve**
\[
\{a \approx b\} \cup E; \nabla; A; \pi \Rightarrow E; \nabla; A \setminus \{b\}; (\pi\cdot a b)\pi, \quad \text{if } \pi\cdot a, b \in A \land \pi\cdot a \neq b.
\]
The input for $E$ is initialized in the $\text{Mer}$ rule, which needs to compute an $A$-based permutation $\pi$ for $A$-based context $\nabla$ and two AUEs $X : t_1 \triangleq s_1$ and $Y : t_2 \triangleq s_2$. The system is initialized by $\{t_1 \approx t_2, s_1 \approx s_2\}; \nabla; A; \pi$. First we apply the rules of phase 1 exhaustively and afterwards $\text{Rem-E}$ and $\text{Sol-E}$ are applied as long as possible. If the final system is the success state $\emptyset; \nabla; A; \pi$, then we say that $E$ computes the permutation $\pi$. Otherwise it has the form $E; \nabla; A; \pi$ with $E \neq \emptyset$ to which no rule applies. It is transformed into $\bot$, called the failure state.

**Example 4.** We illustrate the algorithm $E$ on examples and consider the equivariance problems:

- For $E = \{a \approx a, a.(a.b)(c.d) \cdot X \approx b.X\}$, $A = \{a, b, c, d\}$, and $\nabla = \{a#X\}$, we derive
  \[
  \begin{align*}
  &\{a \approx a, a.(a.b)(c.d) \cdot X \approx b.X\}; \{a#X\}; \{a, b, c, d\}; \text{Id} \Rightarrow \text{AlpE} \\
  &\{a \approx a, (a.(a.b)(c.d) \cdot X \approx b.(c.d) \cdot X\}; \{a#X\}; \{a, b, c, d\}; \text{Id} \Rightarrow \text{BusE} \\
  &\{a \approx a, \epsilon \approx \epsilon, c \approx d, d \approx c\}; \{a#X\}; \{a, b, c, d\}; \text{Id} \Rightarrow 2 \Rightarrow \text{RemE} \\
  &\{c \approx d, d \approx c\}; \{a#X\}; \{b, c, d\}; \text{Id} \Rightarrow \text{RemE} \quad \emptyset; \{a#X\}; \{b\}; \{c, d\}.
  \end{align*}
  \]

- For $E = \{a.b(a.b)(a.c) \cdot X = b.a.(a.c) \cdot X\}$, $A = \{a, b\}$, and $\nabla = \emptyset$, $E$ returns $\text{Id}$.

**Theorem 4.** Let $t, s$ be terms and $\nabla$ be a freshness context, all based on a finite set of atoms $A$.

- Termination: The procedure $E$ terminates on any input.
- Soundness: Let $\{t \approx s\}; \nabla; A; \text{Id} \Rightarrow^* \emptyset; \nabla; B; \pi$ be a derivation in $E$, then $\pi$ is an $A$-based permutation such that $\nabla \vdash \pi \cdot t \approx s$.
- Completeness: If $\nabla \vdash \rho \cdot t \approx s$ for some $A$-based permutation $\rho$, then there is a derivation $\{t \approx s\}; \nabla; A; \text{Id} \Rightarrow^* \emptyset; \Gamma; B; \pi$, obtained by $E$, such that $\pi \cdot a = \rho \cdot a$ for all $a \in \text{FA}(t)$.

**Theorem 5.** The equivariance algorithm $E$ has $O(n^2)$ space and time complexity and the anti-unification algorithm $\mathcal{N}$ has $O(n^4)$ time and $O(n^2)$ space complexity, where $n$ is the input size.

**Acknowledgment**

This research has been partially supported by the project HeLo (TIN2012-33042) and by the Austrian Science Fund (FWF) with the project SToUT (P 24087-N18).

**References**


