Most General Unifiers in Generalized Nominal Unification

Yunus D. K. Kutz[1] and Manfred Schmidt-Schauß[2*]

[1] Goethe-University Frankfurt am Main, Germany, kutz@ki.informatik.uni-frankfurt.de
[2] Goethe-University Frankfurt am Main, Germany, schauss@ki.informatik.uni-frankfurt.de

Abstract

We consider the generalisation of nominal unification where besides expression variables, also atom variables are considered. It is known that in both cases a most general unifier exists for a solvable input constraint. A most general unifier consists of a substitution and a set of freshness constraints, where the latter is an implicit restriction on the instances of the substitution. In the general case with atom variables, we focus in this paper on an improved computation of a most general unifier, and show that it can be computed in quadratic time. Applications of this algorithm are in reasoning about program transformations in higher order functional languages.

Keywords: nominal unification, most general unifier, program transformations, functional languages

1 Introduction

Our motivation to apply and investigate nominal unification is reasoning for higher-order functional languages and proving properties of program transformations, for example correctness or optimization properties.

A previous investigation [7] described and analyzed the generalized nominal unification problem with atom variables and sketched a method using special rules and a special strategy that leads to a single set of freshness constraints that represents all solutions. This permits to already resolve a part of the problem without performing a search. It is known [7] that solvability of the resulting freshness constraints is NP-complete, hence the final check for solvability would still require a search procedure, where it is sufficient to guess the (dis-)equality of all atom-variable pairs.

In this paper we describe the stream-lined and deterministic algorithm AVMGU for computing a most general unifier that runs in quadratic time, delaying the solvability of the resulting freshness constraints. As additional technique we introduce sharing of permutations in order to avoid a size explosion. This is an improvement over [7], where the size explosion by repeated doubling is defeated by guessing and afterwards exploiting the (dis-)equality of atom variables.

For applications where (dis-)equality of the atoms is a part of the problem, our result appears to exhibit an advantage of our approach compared with equivariant unification [2], and the algorithm in [3].

2 The Languages

We recall usual nominal unification with atoms and its results, introduce the generalized languages with atom-variables, introduce the required notation and define the corresponding nominal unification problems.

*supported by the Deutsche Forschungsgemeinschaft (DFG) under grant SCHM 986/11-1.

Let $\mathcal{F}$ be a set of function symbols $f \in \mathcal{F}$, s.t. each f has a fixed arity $ar(f) \geq 0$. Let $\mathcal{A}t$ be *the set of atoms* ranged over by a, b, c. The ground language NL_a consists of atoms, lambda-expressions which bind atoms, and function symbols. The syntax of nominal terms e is as follows:

$$e \in NL_a \quad ::= \quad a \mid (f\ e_1 \dots e_{ar(f)}) \mid \lambda a.e$$

With $AtVar(e)$ we denote the set of atoms contained in e.

For atoms a, b, a *swapping* $(a\ b)$ represents the function $\{a \mapsto b, b \mapsto a\} \cup \{c \mapsto c \mid c \in \mathcal{A}t, c \notin \{a, b\}\}$. A list of swappings π represents a *permutation*, i.e. a bijective mapping that maps atoms to atoms. We write $\emptyset$ for the empty permutation, $\pi_1 \cdot \pi_2$ for composition of permutations π_1 and π_2, and also write $(a\ b) \cdot \pi$ or $\pi \cdot (a\ b)$ to prepend or append a swapping $(a\ b)$ to a permutation. Additionally, we write $\pi{\cdot}e$ (and $(a\ b){\cdot}e$, resp.) for the application of a permutation (a swapping, resp.) to a term e. Similarly, we apply the permutations and swappings to atoms. The domain of permutations is defined as $dom(\pi) = \{a \in \mathcal{A}t \mid \pi(a) \neq a\}$. We assume that π operates on terms (expressions) like a homomorphism. Compositions $\pi_1 \circ \pi_2$ and inverses π^{-1} can be immediately computed as concatenation and as the reversed list, respectively.

For an atom a and a term e the construct $a\#e$ is called a *freshness constraint*, which *holds* iff atom a does not occur free in e. Ground freshness constraints can be immediately computed.

Definition 2.1. *Syntactic α-equivalence $\sim$ in NL_a is inductively defined:*

$$\frac{}{a \sim a} \qquad \frac{\forall i : e_i \sim e'_i}{(f\ e_1 \dots e_{ar(f)}) \sim (f\ e'_1 \dots e'_{ar(f)})} \qquad \frac{e \sim e'}{\lambda a.e \sim \lambda a.e'} \qquad \frac{a\#e' \wedge e \sim (a\ b){\cdot}e'}{\lambda a.e \sim \lambda b.e'}$$

It is known that this exactly defines α-equivalence, and hence the relation $\sim$ is a congruence.

2.1 Nominal Unification

We recall nominal unification and results. We extend the syntax of expression by expression-variables $\mathcal{S}$ where elements of $\mathcal{S}$ are ranged over by S, T. The language NL_{aS} is defined by the following grammar:

$$\begin{array}{lll} e \in NL_{aS} & ::= & a \mid S \mid \pi{\cdot}S \mid (f\ e_1 \dots e_{ar(f)}) \mid \lambda a.e \\ \pi & ::= & \emptyset \mid (a\ a') \cdot \pi \end{array}$$

The inclusion of permutations into the syntax is necessary, since terms like $\pi{\cdot}S$ (suspension) cannot be further evaluated.

A *nominal unification problem* is a pair (Γ, ∇) where Γ is a finite set of equations $e \doteq e'$ with $e, e' \in NL_{aS}$ and ∇ is a finite set of freshness constraints $a\#e$ where a is an atom and $e \in NL_{aS}$. A substitution σ that is ground for (Γ, ∇) is a *solution* of a nominal unification problem (Γ, ∇) iff $e\sigma \sim e'\sigma$ for all $(e \doteq e') \in \Gamma$ and $a\#e\sigma$ is valid for all $(a\#e) \in \nabla$.
A nominal unification problem (Γ, ∇) is *solvable* if, and only if there is a solution for (Γ, ∇). A set of freshness constraints ∇ is *solvable* if, and only if $(\emptyset, \nabla)$ is solvable.

For a nominal unification problem (Γ, ∇), a *unifier* is a pair (σ, ∇') where σ is a substitution and ∇' is a solvable set of freshness constraints, s.t. for all substitutions γ for which $\sigma\gamma$ is ground for Γ, ∇, ∇': $(\forall a\#e \in \nabla' : a\#e\sigma\gamma \text{ is valid }) \implies (\sigma \circ \gamma)$ is a solution for (Γ, ∇).

A unifier (σ, ∇') is a *single most general unifier* of (Γ, ∇), iff for every solution ρ there is a substitution γ, such that $\nabla'\sigma\gamma$ is ground and valid and $S\sigma\gamma \sim \rho(S)$ for all variables S occurring in (Γ, ∇).

Theorem 2.2 ([8, 1, 4, 5])**.** *The nominal unification problem in NL_{aS} is solvable in quadratic time. Moreover, for a solvable nominal unification problem (Γ, ∇), there exists a (single) most general unifier, which can be computed in polynomial time.*

2.2 Introducing Atom-Variables

Let $\mathcal{A}$ be the *set of atom-variables* ranged over by A, B, and let atom-variable swappings be denoted as $(A\ B)$, and permutations be represented as lists of swappings (see also Remark 2.5), which stand for concrete permutations as instances. For simplicity, we omit atoms in the language. The grammar of the nominal language NL_{AS} with atom-variables is

$$\begin{array}{lcl} e & ::= & A \mid S \mid \pi{\cdot}A \mid \pi{\cdot}S \mid (f\ e_1 \dots e_{ar(f)}) \mid \lambda\pi{\cdot}A.e \\ \pi & ::= & \emptyset \mid (A\ A') \cdot \pi \end{array}$$

$AtVar(e)$ are the atom-variables contained in e, $ExVar(e)$ the expression-variables contained in e, and $tops(e)$ are the top-symbol of expression e.

A freshness constraint for the language NL_{AS} is of the form $A\#e$ where e is an NL_{AS}-expression and A is an atom-variable.

A *ground substitution* ρ is a mapping from A-variables to atoms, and S-variables to ground expressions (in NL_a). Let Γ be a set of equations of the form $e_1 \doteq e_2$ with NL_{AS}-expressions e_1, e_2, and let ∇ be a set of freshness constraints over NL_{AS}. Then (Γ, ∇) is a *variable-atom nominal unification problem*.

A substitution ρ that is ground for (Γ, ∇) is a *solution* of (Γ, ∇), iff for all equations $e_1 \doteq e_2$ in Γ: $e_1\rho \sim e_2\rho$, and for all freshness constraints $A\#e \in \nabla$ the freshness constraint $A\rho\#e\rho$ is valid. In this case we say that (Γ, ∇) is *solvable*.

A pair (σ, ∇') of a substitution σ and a solvable set ∇' of freshness constraints is a *unifier* of (Γ, ∇) iff for all substitutions γ such that $\sigma\gamma$ is ground for Γ, ∇, ∇': $(\forall A\#e \in \nabla',\ A\sigma\gamma\#e\sigma\gamma$ is valid $) \implies \sigma \circ \gamma$ is a solution for (Γ, ∇).
For a variable-atom nominal unification problem (Γ, ∇), a unifier (σ, ∇') is a *(single) most general unifier*, iff for every solution ρ of (Γ, ∇), there is a substitution γ, such that $\nabla'\sigma\gamma$ is ground and valid and $A\sigma\gamma = \rho(A)$, $S\sigma\gamma \sim \rho(S)$ for all atom-variables A and expression-variables S occurring in (Γ, ∇).

Example 2.3. *Consider the example $(A\ B){\cdot}C \doteq C$ (see Remark 3.10 in [8]). The set $\{(Id, \{A\#C, B\#C\}), (\{A \mapsto C, B \mapsto C\}, \emptyset)\}$ is complete as a set of unifiers and contains two substitutions as unifiers, which are incomparable. In the literature, this example is used as justifying the conjecture that sometimes a set of at least two unifiers is necessary for completeness instead of a (single) most general unifier. However, in fact there is a single most general unifier using freshness constraints: it is the pair $(Id, \{C\#\lambda(A\ B){\cdot}C.C\})$.*

This example grants an insight into the power of freshness constraints. For atom expressions $\pi_i{\cdot}A_i$, $i = 1, 2$ the constraint $A_1\#\lambda{\pi_1}^{-1}\pi_2 \cdot A_2.A_1$ is equivalent to $\pi_1 \cdot A_1 \doteq \pi_2 \cdot A_2$, i.e. any solution of one is also a solution of the other. This technique of translating equations into equivalent constraints yields a most general unifier if the unification problem is solvable.

From now on we syntactically permit $\pi_1 \cdot A_2 \doteq_{\#} \pi_2 \cdot A_2$ in the freshness-constraints as an internal representation of an encoding as a constraint. We also allow $CompFixFC(\pi, S)$ in the constraints, which expands to the set $\{A\#\lambda\pi{\cdot}A.S \mid A \in AtVar(\pi)\}$ (see also Remark 2.5).

Definition 2.4. *The rules of our unification algorithm* AVMGU *are in Fig. 1 and the failure rules are in Fig. 2. The algorithm* AVMGU *is applied on a triple (Γ, ∇, θ) of symmetric unification equations Γ, freshness constraints ∇, and a substitution θ, where θ initially is empty. In addition we assume that all equations are flattened. This means that abstractions and function applications in Γ are only of the form $\lambda\pi{\cdot}A.\pi'{\cdot}S$ and $(f\ \pi_1{\cdot}S_1\ \dots\ \pi_n{\cdot}S_n)$ where equations may be added if necessary. The calls $CompFixFC(\pi, S)$ are not executed but delayed.*

(U0) $\dfrac{(\Gamma \cup \{e \doteq e\}, \nabla, \theta)}{(\Gamma, \nabla, \theta)}$ (U1) $\dfrac{(\Gamma \cup \{\pi \cdot A \doteq \pi' \cdot A'\}, \nabla, \theta)}{(\Gamma, \nabla \cup \{\pi \cdot A \doteq_{\#} \pi' \cdot A'\}, \theta)}$

(U2) $\dfrac{(\Gamma \cup \{(f\ e_1 \dots e_{ar(f)}) \doteq (f\ e'_1 \dots e'_{ar(f)})\}, \nabla, \theta)}{(\Gamma \cup \{e_1 \doteq e'_1, \dots, e_{ar(f)} \doteq e'_{ar(f)}\}, \nabla, \theta)}$ (U3) $\dfrac{(\Gamma \cup \{\lambda A.e_1 \doteq \lambda A.e_2\}, \nabla, \theta)}{(\Gamma \cup \{e_1 \doteq e_2\}, \nabla)}$

(U4) $\dfrac{(\Gamma \cup \{\lambda A_1.e_1 \doteq \lambda A_2.e_2\}, \nabla, \theta)}{(\Gamma \cup \{e_1 \doteq (A_1\ A_2) \cdot e_2\}, \nabla \cup \{A_1 \# \lambda A_2.e_2\}, \theta)}$

(U5) $\dfrac{(\Gamma \cup \{\lambda \pi_1 \cdot A_1.e_1 \doteq \lambda \pi_2 \cdot A_2.e_2\}, \nabla, \theta)}{(\Gamma \cup \{e_1 \doteq (A'_1\ A'_2) \cdot e_2\}, \nabla \cup \{A'_1 \doteq_{\#} \pi_1 \cdot A_1, A'_2 \doteq_{\#} \pi_2 \cdot A_2, A'_1 \# \lambda A'_2.e_2\}, \theta)}$ where A'_1, A'_2 are new variables.

(U6) $\dfrac{(\Gamma \cup \{\pi \cdot S \doteq \pi' \cdot S\}, \nabla, \theta)}{(\Gamma, \nabla \cup \mathit{CompFixFC}(\pi^{-1}\pi', S), \theta)}$

(U7) $\dfrac{(\Gamma \cup \{\pi \cdot S \doteq \pi' \cdot X\}, \nabla, \theta)}{(\Gamma[(\pi^{-1}\pi' \cdot X)/S], \nabla[(\pi^{-1}\pi' \cdot X)/S], \theta \cup \{S \mapsto \pi^{-1}\pi' \cdot X\})}$ if $X \in \mathcal{S} \cup \mathcal{A}t$ and $S \neq X$

(MMS) $\dfrac{(\Gamma \cup \{\pi_1 \cdot S \doteq e_1, \dots, \pi_n \cdot S \doteq e_n\}, \nabla, \theta)}{(\Gamma \cup \{\pi_1^{-1} \cdot e_1 \doteq \pi_2^{-1} \cdot e_2, \dots, \pi_1^{-1} \cdot e_1 \doteq \pi_n^{-1} \cdot e_n\}, \nabla, \theta \cup \{S \mapsto \pi_1^{-1} \cdot e_1\})}$ if the rules (Ui) for $i \in \{0, \dots, 7\}$ are not applicable and S does not occur in Γ, nor in e_j for all $j \in \{0, \dots, n\}$.

Figure 1: Unification rules for computing an mgu

(ClashFailure) $\dfrac{(\Gamma \cup \{e_1 \doteq e_2\}, \nabla, \theta)}{\mathit{Fail}}$ if e_1, e_2 are not variables nor suspensions, and $\mathit{tops}(e_1) \neq \mathit{tops}(e_2)$

(VarFail) $\dfrac{(\Gamma \cup \{\pi \cdot A \doteq e\}, \nabla, \theta)}{\mathit{Fail}}$ if $\mathit{tops}(e)$ is a function symbol f or λ

(CycleDetection) $\dfrac{(\Gamma \cup \{\pi_1 \cdot S_1 \doteq e_1, \dots, \pi_n \cdot S_n \doteq e_n\}, \nabla, \theta)}{\mathit{Fail}}$ if all e_i are neither variables nor suspensions and S_{i+1} occurs in e_i for $i = 1, \dots, n-1$ and S_1 occurs in e_n

Figure 2: Failure rules

These rules permit to resolve all the expression variables S in a don't care fashion, and the result will be a single set of freshness constraints C. Note that the solvability of the freshness constraints in NL_{aS} is NP-hard [7].
We have to take care to control the size of the equations and freshness constraint. This requires as precaution a compact representation of the substitutions for S-variables, as well as a dag-representation for the permutations as lists of swappings. It is not possible to simplify the permutations as in NL_{aS}, since in NL_{AS} they are composed of swappings of atom variables. The construction of the substitution θ uses a representation in triangle-form, which is like a dag-representation.

Remark 2.5. *The dag-representation of permutations can be done using a grammar-like notation (like straight-line programs (SLPs) see [6]). Let the nodes of the directed acyclic graph be $P_1, \dots, P_n$, and let there be leaf nodes where P is labeled with a swapping $(A\ B)$, and inner*

nodes where P labeled with $P_1 \cdot P_2$, where P_1, P_2 are nodes, or with P'^{-1}. Every node represents a list of swappings, where leaf nodes represent a list of one element. Inverses mean that the reversed list is represented. The represented list may be exponentially long in the size of the dag. If all atom variables are instantiated, then the concrete permutation can be computed in time $O(n^2 \log n)$ by computing the permutations bottom-up in the dag using an efficient data structure. The first step in solving the constraints would be to expand $CompFixFC(\pi, S)$ to $\{A\#\lambda\pi\cdot A.S \mid A \in AtVar(\pi)\}$. This computation requires the set $AtVar(\pi)$, which can be computed in $O(n \cdot \log n)$ time in the size of the dag-representation using a dynamic programming approach. □

The representation of permutations is left implicit in our algorithm. We also do not use the computations with permutations in the unification algorithm AVMGU. These do not contribute to the complexity, since AVMGU does not operate on the permutations.

The condition for applying (MMS) ensures that exponentially long execution sequences are avoided, since (MMS) is applied only if S is "on the top", and after (MMS)-application there will be a series of decompositions that make the system of equations smaller.

The rules (U4), (U5) differ from the formulation of the rules in [8]. This is due to goal of computing a single mgu in the presence of atom variables. Consider the input $\{\lambda\pi_1\cdot A_1.e_1 \doteq \lambda\pi_2\cdot A_2.e_2\}$. If we follow [8], then the approach would be to distinguish the two cases $(\pi_1\cdot A_1)\rho = (\pi_2\cdot A_2)\rho$ with the resulting equation $e_1 \doteq e_2$; and the case $(\pi_1\cdot A_1)\rho \neq (\pi_2\cdot A_2)\rho$ with the resulting equation $e_1 \doteq ((\pi_1\cdot A_1)\ (\pi_2\cdot A_2))\cdot e_2$ and the constraint $\pi_1\cdot A_1\#e_2$. However, these are alternatives and thus the unification has to perform a search. Furthermore, nested permutations, e.g. $(A\ ((C\ D)\ B))$, would need to be allowed without the introduction of fresh atom variables.

Our rules avoid nesting of permutations and make progress without searching: (U5) abbreviates the suspensions and (U4) joins the two alternatives: $\lambda A_1.e_1 \doteq \lambda A_2.e_2$ is transformed into $e_1 \doteq (A_1\ A_2)\cdot e_2$ and the additional constraint $A_1\#\lambda A_2.e_2$, which for $A_1\rho = A_2\rho$, is exactly the result in this case after a simplification, and for $A_1\rho \neq A_2\rho$ is exactly the result in this case after simplifying $A_1\#\lambda A_2.e_2$ to $A_1\#e_2$.

Example 2.6. *The equation $\lambda A.A \doteq \lambda B.S$ has a solution $\{A \mapsto a, B \mapsto b, S \mapsto b\}$. It has a (most general) unifier $\{S \mapsto B\}$ with empty set ∇: The algorithm* AVMGU *computes theta $= \{\S \mapsto B\}$ and $\nabla = \{A\#\lambda B.S\}$, where ∇ can be simplified to the empty set.*

Example 2.7. *The equation $\lambda A.(f\ A\ S) \doteq \lambda B.(A\ B)\cdot(f\ S\ B)$ for a binary symbol f has a solution $\{A \mapsto a, B \mapsto a, S \mapsto a\}$. The algorithm computes the most general unifier: $(\{S \mapsto (A\ B)A\}, \{A\#\lambda B.(A\ B)\cdot(f\ S\ B)\})$. After deducing from the constraint $A \doteq_{\#} B$ and applying some simplification rules, one obtains: $(\{S \mapsto B, A \mapsto B\}, \emptyset)$ as an alternative mgu.*

In the following we assume that the size of the signature, in particular the arity of function symbols is a constant.

Theorem 2.8. *The algorithm* AVMGU *computes for solvable input (Γ, ∇) a most general unifier in time $O(n^2)$. Using dag-compression for permutations, the output size is at most $O(n^2)$. The size of the input is n, where names count as 1.*

Proof. (Sketch) Translation into flattened form can be done in linear time. Let $size_{+p}$ or $size_{-p}$ be the size of Γ with (or without, respectively) counting permutations. Every rule application, after a constant number of further rule applications, strictly decreases $size_{-p}$ of Γ. Hence there is at most a linear number of applications of rules, and every application requires at most a linear number of substitution actions. Hence the time $O(n^2)$ is sufficient.
For estimating the size, we assume that permutations are represented in a global dag. Note that (MMS) may increase $size_{+p}$, but at most by $O(n)$. The same holds for (U7). Thus the global

size of the dag is $O(n^2)$. The other output components are increased at most a constant amount in every rule application, and may refer to the global permutation-dag, hence the overall size of the output is $O(n^2)$. □

Note that solving the resulting freshness constraints first has to expand the *CompFixFC*-calls, and to make analyses using the permutation dag, and thus the complexity is presumably worse than non-deterministic quadratic time.

3 Conclusion

A single most general unifier consisting of a substitution and freshness constraint for generalized nominal unifications problems with atom variables can be computed in quadratic time, provided the freshness constraints are known as solvable.
Future research is to implement a practically useful variant of our algorithm and to construct a good solution algorithm for the NP-complete problem of solvability of freshness constraints.

References

[1] C. Calvès and M. Fernández. A polynomial nominal unification algorithm. *Theor. Comput. Sci.*, 403(2-3):285–306, 2008.

[2] J. Cheney. Equivariant unification. *JAR*, 45(3):267–300, 2010.

[3] M. R. Lakin. Constraint solving in non-permutative nominal abstract syntax. *Logical Methods in Computer Science*, 7(3), 2011.

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

[5] J. Levy and M. Villaret. An efficient nominal unification algorithm. In C. Lynch, editor, *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 209–226. Schloss Dagstuhl, 2010.

[6] M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

[7] M. Schmidt-Schauß, D. Sabel, and Y. Kutz. Nominal unification with atom-variables. *JSC, special issue*, 2017. to be published.

[8] C. Urban, A. M. Pitts, and M. Gabbay. Nominal unification. In *17th CSL, 12th EACSL, and 8th KGC*, volume 2803 of *LNCS*, pages 513–527. Springer, 2003.