Equivariant Unification

Article *in* Journal of Automated Reasoning · January 2005

DOI: 10.1007/s10817-009-9164-3 · Source: DBLP

1 author:

James Cheney
The University of Edinburgh
165 PUBLICATIONS **3,832** CITATIONS

SEE PROFILE

Equivariant Unification

James Cheney

University of Edinburgh

Abstract. Nominal logic is a variant of first-order logic with special facilities for reasoning about names and binding based on the underlying concepts of swapping and freshness. It serves as the basis of logic programming and term rewriting techniques that provide similar advantages to, but remain simpler than, higher-order logic programming or term rewriting systems. Previous work on nominal rewriting and logic programming has relied on nominal unification, that is, unification up to equality in nominal logic. However, because of nominal logic's equivariance property, these applications require a stronger form of unification, which we call *equivariant unification*. Unfortunately, equivariant unification and matching are **NP**-hard decision problems. This paper presents an algorithm for equivariant unification that produces a complete set of finitely many solutions, as well as **NP** decision procedure and a version that enumerates solutions one at a time. In addition, we present a polynomial time algorithm for *swapping-free* equivariant matching, that is, for matching problems in which the swapping operation does not appear.

1 Introduction

Gabbay and Pitts [6] introduced a novel approach to formalizing and reasoning about abstract syntax involving bound names, based on the fundamental ideas of name-swapping and freshness. We call this approach *nominal abstract syntax* (NAS). Initially, this approach was based on FM-set theory, a variant of standard ZF-set theory originally developed to prove the independence of the Axiom of Choice. However, Pitts [7] showed that this radical step can be avoided by incorporating the ideas of nominal abstract syntax into a logic (called *nominal logic*) whose intended semantics is based on FM set theory but rests on standard mathematical foundations.

The key elements of nominal logic are: a collection of infinitely many term symbols $\mathsf{a}, \mathsf{b}, \ldots \in Name$ called *names*; a binary relation $\#$ called *freshness* that can hold between a name and a value; a *swapping function* $(a\ b)\cdot t$ that exchanges the values of names a and b in t; and an *abstraction function* $\langle a\rangle x$ that takes a name and value. Abstractions are considered equal up to α-equivalence; for example $\langle \mathsf{a}\rangle f(\mathsf{a},\mathsf{c}) \approx \langle \mathsf{b}\rangle f(\mathsf{b},\mathsf{c})$.

Nominal logic has been used as a basis for logic programming [1, 3] and term rewriting systems [4]. So far, these techniques have relied upon the (efficiently implementable) *nominal unification* algorithm of Urban, Pitts, and Gabbay [11] as a fundamental tool, just as first-order unification is used in ordinary logic programming and term rewriting. However, as shown by Cheney [3, 2], nominal unification is not the right tool for this job: proof search and term rewriting using nominal unification is incomplete.

First-order unification is complete for first-order resolution and rewriting because ground atomic formulas are logically equivalent if and only if they are equal as terms. But due to nominal logic's *equivariance* property, this is not the case for nominal logic. The equivariance property states that validity is preserved by applying name-swappings uniformly: that is, $p(\bar{t}) \iff p((a\ b)\cdot\bar{t})$. As a result, atomic formulas (such as $p(\mathsf{a})$ and $p(\mathsf{b})$) may be equivalent without being equal nominal terms. Similarly, if a collection of rewriting rules $t \to u$ is used to define a relation $\to$ in nominal logic, then $t \to u$ is equivalent to $(a\ b)\cdot t \to (a\ b)\cdot u$.

Consider the following logic program clauses and rewriting rules:

$$spec(mono(T), [], T). \quad spec(all(\langle \mathsf{a} \rangle T), [\mathsf{a}|Vs], U) :- spec(T, Vs, U).$$
$$subst(var(\mathsf{a}), T, \mathsf{a}) \to T \qquad subst(var(\mathsf{b}), T, \mathsf{a}) \to var(\mathsf{b})$$
$$subst(app(E1, E2), T, \mathsf{a}) \to app(subst(E1, T, \mathsf{a}), subst(E2, T, \mathsf{a}))$$
$$\mathsf{b} \# T \vdash subst(lam(\langle \mathsf{b} \rangle E), T, \mathsf{a}) \to lam(\langle \mathsf{b} \rangle subst(E, T, \mathsf{a}))$$

The $spec$ predicate is taken from an αProlog [1] program that performs ML type inference. It relates a polymorphic type to a list of bound variables and a monomorphic type, and can be used in the forward direction to instantiate the bound variables of a polymorphic type to fresh names, or backwards to quantify the free type variables of an inferred type. The $subst$ rewriting rules perform capture-avoiding substitution on λ-terms encoded using nominal abstract syntax. (Note that nominal rewriting rules can have freshness "guards", e.g. $\mathsf{a} \# X \vdash l \to r$ applies only when $\mathsf{a} \# X$.)

Nominal unification and matching do not (and should not) take equivariance into account. As a result, logic programs or rewriting systems may not work as desired when nominal unification is used for backchaining or nominal matching is used for term rewriting, respectively. The goal $spec(all(\langle \mathsf{a} \rangle mono(tvar(\mathsf{a}))), [\mathsf{b}], U)$ has solution $[U = tvar(\mathsf{b})]$ in nominal logic, but this solution cannot be found using nominal unification. As another example, in nominal logic the first rewriting rule for $subst$ implies that $subst(var(\mathsf{b}), var(\mathsf{a}), \mathsf{b})$ rewrites to $var(\mathsf{a})$, but there is no substitution for T making $subst(var(\mathsf{a}), T, \mathsf{a}) \approx subst(var(\mathsf{b}), var(\mathsf{a}), \mathsf{b})$.

Therefore, it is necessary to unify or match modulo a stronger equational theory that takes equivariance into account. We call these problems *equivariant matching* and *equivariant unification*, respectively. Equivariant unification is of both practical and theoretical interest. On the theoretical side, Cheney [2] showed that equivariant unification is **NP**-hard. On the practical side, there are some interesting programs (such as $spec$) that only appear to be expressible using equivariant unification. In addition, equivariant matching seems desirable in nominal rewriting systems for clarity and simplicity. For example, in the nominal rewriting approach advocated by Fernandez et al. [4], the $subst$ rewrite rules above will not work properly. Instead, the following rewrite system was used for capture-avoiding substitution:

$$subst'(\langle \mathsf{a} \rangle var(\mathsf{a}), T) \to T \qquad \mathsf{a} \# B \vdash subst'(\langle \mathsf{a} \rangle var(B), T) \to var(B)$$
$$subst'(\langle \mathsf{a} \rangle app(E1, E2), T) \to app(subst'(\langle \mathsf{a} \rangle E1, T), subst'(\langle \mathsf{a} \rangle E2, T))$$
$$\mathsf{b} \# T \vdash subst'(\langle \mathsf{a} \rangle lam(\langle \mathsf{b} \rangle E), T) \to lam(\langle \mathsf{b} \rangle subst'(\langle \mathsf{a} \rangle E, T))$$

In this paper we make two significant contributions:

– We present a **NP** algorithm for equivariant unification that produces at most finitely many different solutions. This is the first (terminating) algorithm to be developed for general equivariant unification.[1] Besides taking equivariance into account, our algorithm solves a more general form of nominal unification problems than those considered by [11]. This algorithm can be used to run arbitrary nominal logic programs and rewriting systems and may also be useful in analyzing such systems.
– We present a polynomial-time algorithm for *swapping-free* equivariant matching problems, that is, problems in which the swapping function symbol is not present. This is significant because typical nominal rewriting systems that require equivariance (including $subst$) are swapping-free. This algorithm can be used as the basis of efficient nominal term rewriting for a larger class of programs than considered by Fernandez, Gabbay, and Mackie [4].

The remainder of this paper is structured as follows. In the next section, we review nominal equational logic. In Section 3, we introduce *permutation graphs*, an important tool for solving basic equivariant unification and matching problems that is used in the rest of the paper. In Section 4, we present the equivariant unification algorithm and sketch proofs of its important properties. Likewise, in Section 5 we present the swapping-free equivariant matching algorithm and prove its properties. Section 6 discusses additional related work and future directions, and Section 7 concludes.

2 Background

We first consider the set $Term$ of ground nominal terms, given by the grammar

$$t ::= \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \mathsf{a} \mid \langle \mathsf{a}\rangle t$$

The first three cases denote units, pairing, and function symbols; we represent constant symbols c as functions applied to unit $f(\langle\rangle)$ and represent n-ary function applications $f(t_1, \ldots, t_n)$ using iterated pairing $f(\langle t_1, \langle t_2, \cdots\rangle\rangle)$. Names a, a' are drawn from a countably infinite set $Name$, and abstractions $\langle \mathsf{a}\rangle t$ represent terms with bound names.

Let $Perm$ be the set of (finite) permutations of names. We write $\pi{\cdot}t$ for the *action* of $\pi \in Perm$ on t, or the result of applying π to rename the names of t. The permutation action function and equality $\approx$: $Term \times Term$ and freshness $\#$: $Name \times Term$ relations are defined in Figure 1. Equality is syntactic equality except for abstractions, which are considered equal modulo renaming of the bound names to a fresh name. The freshness theory spells out when a name is fresh for (not free in) a term. In particular, $\mathsf{a} \# \langle \mathsf{a}\rangle t$ holds unconditionally, while $\mathsf{a} \# \langle \mathsf{b}\rangle t$ holds for $\mathsf{a} \neq \mathsf{b}$ if $\mathsf{a} \# t$.

Our definitions freshness and equality are superficially different from those used by Urban et al., but they are equivalent for ground terms. Urban et al. unified nominal terms modulo an equational theory axiomatizing equality and freshness judgments $\nabla \vdash A$ in the presence of some assumptions ∇ of the form $\mathsf{a} \# X$, for names a and variables

[1] Cheney [2] only established that the equivariant matching and unification problems **for terms involving only names, variables, and swappings** are **NP**-complete, but did not present algorithms or upper bounds for problems involving general nominal terms.

$$\pi{\cdot}\mathsf{a} = \pi(\mathsf{a}) \quad \pi{\cdot}\langle\rangle = \langle\rangle \quad \pi{\cdot}\langle t, u\rangle = \langle \pi{\cdot}t, \pi{\cdot}u\rangle \quad \pi{\cdot}f(t) = f(\pi{\cdot}t) \quad \pi{\cdot}\langle \mathsf{b}\rangle t = \langle \pi{\cdot}\mathsf{b}\rangle \pi{\cdot}t$$

$$\frac{(\mathsf{a} \neq \mathsf{b})}{\mathsf{a} \# \mathsf{b}} \quad \frac{}{\mathsf{a} \# \langle\rangle} \quad \frac{\mathsf{a} \# t}{\mathsf{a} \# f(t)} \quad \frac{\mathsf{a} \# t \quad \mathsf{a} \# u}{\mathsf{a} \# \langle t, u\rangle} \quad \frac{}{\mathsf{a} \# \langle \mathsf{a}\rangle t} \quad \frac{(\mathsf{a} \neq \mathsf{b}) \quad \mathsf{a} \# t}{\mathsf{a} \# \langle \mathsf{b}\rangle t} \quad \frac{}{\mathsf{a} \approx \mathsf{a}} \quad \frac{}{\langle\rangle \approx \langle\rangle}$$

$$\frac{t_1 \approx u_1 \quad t_2 \approx u_2}{\langle t_1, t_2\rangle \approx \langle u_1, u_2\rangle} \quad \frac{t \approx u}{f(t) \approx f(u)} \quad \frac{t \approx u}{\langle \mathsf{a}\rangle t \approx \langle \mathsf{a}\rangle u} \quad \frac{(\mathsf{a} \neq \mathsf{b}) \quad \mathsf{a} \# u \quad t \approx (\mathsf{a}\ \mathsf{b}){\cdot}u}{\langle \mathsf{a}\rangle t \approx \langle \mathsf{b}\rangle u}$$

Fig. 1. Swapping, equality, and freshness for ground terms

X. We instead axiomatize equality for ground terms only. Note that both freshness and equality are *equivariant*, that is, $t \approx u \supset \pi{\cdot}t \approx \pi{\cdot}u$ and $a \# t \supset \pi{\cdot}a \# \pi{\cdot}t$ for any a, t, u, π.

We now generalize to non-ground nominal terms so that *name-variables* $A, B, \ldots \in NVar$ and *term-variables* $X, Y, \ldots \in Var$ are permitted. In addition, we add explicit syntax for *permutation terms* Π applied to nominal terms, including swappings, composition, inversion, and permutation variables $Q, R, \ldots \in PVar$. Consider terms of the form:

$$\begin{array}{ll} v, w ::= \mathsf{a} \mid A & \Pi, \Pi' ::= Q \mid \mathsf{id} \mid (a\ b) \mid \Pi \circ \Pi' \mid \Pi^{-1} \\ a, b ::= \Pi{\cdot}v & t, u ::= \Pi{\cdot}X \mid a \mid \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a\rangle t \end{array}$$

We write $FN(t)$, $FV(t)$, $FNV(t)$ and $FPV(t)$ for the sets of names, term variables, name variables, and permutation variables of t. This grammar forbids permutation terms except immediately around names or variables. We define $\Pi{\cdot}t$ for arbitrary terms t as follows:

$$\Pi{\cdot}\langle\rangle = \langle\rangle \quad \Pi{\cdot}\langle t, u\rangle = \langle \Pi{\cdot}t, \Pi{\cdot}u\rangle \quad \Pi{\cdot}f(t) = f(\Pi{\cdot}t)$$
$$\Pi{\cdot}\langle a\rangle t = \langle \Pi{\cdot}a\rangle \Pi{\cdot}t \quad \Pi{\cdot}(\Pi'{\cdot}t) = (\Pi \circ \Pi'){\cdot}t$$

Urban et al. considered a more restrictive language of nominal terms in which permutation variables were not present, and required a and b to be ground names in terms of the forms $a \# t$, $(a\ b){\cdot}t$, and $\langle a\rangle t$. These restrictions were crucial for obtaining an efficient, deterministic unification algorithm. To avoid confusion, we refer to such terms as *grounded terms*, and to Urban et al.'s algorithm as *grounded nominal unification*. There are several important differences between our nominal terms and grounded terms. For our nominal terms, permutations applied to names cannot always be simplified: for example, $Q{\cdot}\mathsf{a}$ cannot be simplified without knowing something about Q. Another difference is that name variables are permitted in any place where a name would be permitted. Nominal unification is **NP**-complete for arbitrary terms [3, Ch. 7], but tractable for grounded terms [11]. General equivariant unification and matching are **NP**-complete even for grounded terms (see [2] and Section 4). However, equivariant matching is tractable for grounded, swapping-free terms (see Section 5).

We refer to atomic formulas $t \approx u$, $a \# u$ as *constraints* C, conjunctions and $\exists$-quantifications of constraints as *problems* S, and disjunctions of problems as *extended problems* $\mathcal{M} = \left\{ \begin{smallmatrix} S_1; \\ \cdots \\ S_n \end{smallmatrix} \right\}$. An arbitrary problem involving terms that may have permutation variables is called an *equivariant unification* problem. A problem involving no

$$\theta(\langle\rangle) = \langle\rangle \quad \theta(\langle t, u\rangle) = \langle\theta(t), \theta(u)\rangle \quad \theta(f(t)) = f(\theta(t))$$
$$\theta(\Pi{\cdot}v) = \theta(\Pi){\cdot}\theta(v) \quad \theta(\mathsf{a}) = \mathsf{a} \quad \theta(\langle a\rangle t) = \langle\theta(a)\rangle\theta(t)$$
$$\theta(\mathsf{id}) = \mathsf{id} \quad \theta(\Pi \circ \Pi') = \theta(\Pi) \circ \theta(\Pi') \quad \theta(\Pi^{-1}) = \theta(\Pi)^{-1} \quad \theta((a\ b)) = (\theta(a)\ \theta(b))$$

Fig. 2. Valuations

permutation variables is called a *nominal unification* problem. A problem in which all equations involving permutation variables are of the form $Q{\cdot}t \approx u$, where u is ground, is called an *equivariant matching* problem. Problems are *grounded*, *name–name*, or *swapping-free* if all terms are grounded, if only names, swappings, and name variables are present, or if the swapping operation is not present, respectively.

A *valuation* is a function θ mapping term variables to ground terms, name variables to ground names, and permutation variables to ground permutations. Valuations are extended to terms as shown in Figure 2. We say that $\theta \vDash t \approx u$ if $\theta(t) \approx \theta(u)$; similarly, $\theta \vDash a \# t$ if $\theta(a) \# \theta(t)$. If S is a set of constraints, then we write $\theta \vDash S$ if $\theta \vDash A$ for each $A \in S$, and $\theta \vDash \exists X.S$ if $\theta[X := t] \vDash S$ for some t.

We write $Solv(S)$ for $\{\theta \mid \theta \vDash S\}$ and $Solv(\mathcal{M})$ for $\bigcup_{S\in\mathcal{M}} Solv(S)$. A problem S is a *pre-solution* to $\mathcal{M}$ if $Solv(S) \subseteq Solv(\mathcal{M})$, and a *solution* if in addition $Solv(S) \neq \varnothing$. A solution S to $\mathcal{M}$ is *more general* than another solution T if $Solv(T) \subseteq Solv(S)$, and *most general* if no strictly more general solution exists. A set $\mathcal{M}'$ of (pre-)solutions to $\mathcal{M}$ is a *complete* for $\mathcal{M}$ if $Solv(\mathcal{M}) = Solv(\mathcal{M}')$ and *minimal* if each $S \in \mathcal{M}'$ is a most general solution.

Example 1. A complete minimal set of solutions to the problem $(A\ B){\cdot}C \approx C$ is $\{\{C \approx A, A \approx B\}, \{A \# C, C \# B\}\}$. A complete minimal solution set to the problem $Q{\cdot}\mathsf{a} \# \langle\mathsf{b}\rangle C$ is $\{\{Q{\cdot}\mathsf{a} \approx \mathsf{b}\}, \{Q{\cdot}\mathsf{a} \# \mathsf{b}, Q{\cdot}\mathsf{a} \# C\}\}$. The equivariant matching problem $Q{\cdot}(A, (\mathsf{a}\ \mathsf{b}){\cdot}A, B, (\mathsf{a}\ \mathsf{b}){\cdot}B) \approx (\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d})$ has no solutions. The problem $f(\langle\mathsf{a}\rangle A, \mathsf{b}) \approx f(\langle\mathsf{c}\rangle\mathsf{d}, \mathsf{d})$ has a unique most general solution $A \approx \mathsf{b}$.

3 Permutation Graphs

In this section we consider an important data structure for representing information about permutations, names, and freshness, called a *permutation graph* (or p-graph).

Definition 1. *A p-graph $G = (N, V, PV, E_{\approx}, E_{\#}, E_Q, \ldots)$ is a structure such that $N \subseteq Name$, $V \subseteq Var$ and $PV \subseteq PVar$ are finite, $E_{\approx}$ and $E_{\#}$ are undirected graphs on $W = N \cup V$, and E_Q is a directed graph on W for each $Q \in PV$.*

Note that the vertices v of a p-graph may be either names $a, b, \ldots$ or name variables $A, B, \ldots$. There are three kinds of edges: undirected equality edges (written using a double line $v == w$), undirected freshness edges (written as a broken line $v \dashv\!\vdash w$), and directed permutation edges (written $v \xrightarrow{Q} w$). We consider the edge $v == w$ equivalent to the formula $v \approx w$, $v \dashv\!\vdash w$ equivalent to $v \# w$, and $v \xrightarrow{Q} w$

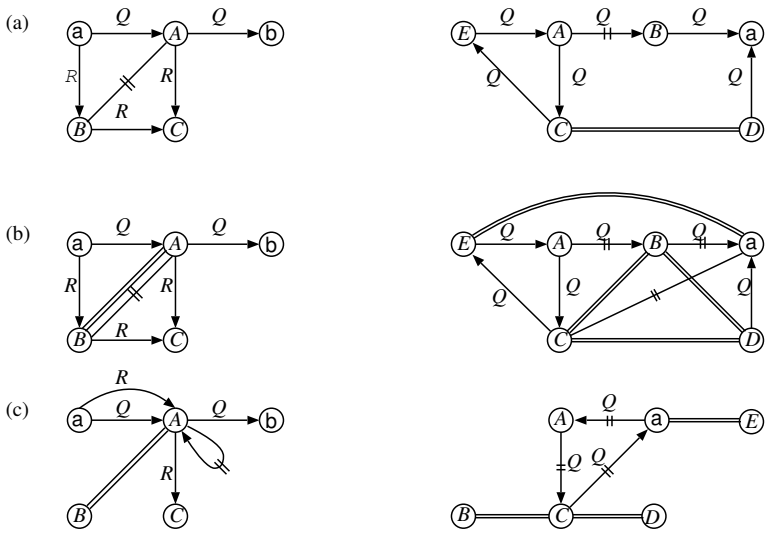

Fig. 3. (a) Example p-graphs. (b) Simplified versions. (c) Solved forms

equivalent to $Q{\cdot}v \approx w$. We write S_G for the problem corresponding to the edges of G and G_S for the p-graph corresponding the problem S.

For example, two small p-graphs are shown in Figure 3(a). Freshness and permutation edges are sometimes superimposed in these diagrams. These graphs correspond to the problems

$$\{Q{\cdot}\mathsf{a} \approx A, Q{\cdot}A \approx \mathsf{b}, R{\cdot}\mathsf{a} \approx B, R{\cdot}B \approx C, R{\cdot}A \approx C, A \# B\}$$
$$\{Q{\cdot}E \approx A, Q{\cdot}A \approx B, A \# B, Q{\cdot}B \approx \mathsf{a}, Q{\cdot}A \approx C, Q{\cdot}C \approx E, C \approx D, Q{\cdot}D \approx \mathsf{a}\}$$

Testing the satisfiability of such problems is not straightforward, because there are hidden consequences. For example, the first set of constraints implies $R{\cdot}A \approx R{\cdot}B$, so $A \approx B$ since R is invertible. Similarly, in the second problem, since $A \# B$, we know $Q{\cdot}A \# Q{\cdot}B$, so $C \approx Q{\cdot}B \# Q{\cdot}A \approx a$. As a result of such observations, additional edges can be added to the graph to obtain a "simpler" graph with fewer hidden consequences. Our example graphs can be simplified in this way as shown in Figure 3(b).

In addition, when there is a variable equality edge involving a variable, such as $A \approx v \in G$, the graph can be simplified by collapsing A and v. This process is exactly analogous to substituting for A in the corresponding problem S_G. The results of collapsing the simplified example graphs are shown in Figure 3(c). The resulting graphs are fully simplified, and testing satisfiability is trivial because there are no remaining hidden consequences. The first graph is clearly unsatisfiable since there is a freshness edge corresponding to a formula $A \# A$. On the other hand, the second graph is satisfiable because there are no such edges. One satisfying valuation is $Q = (\mathsf{a}\ \mathsf{c})(\mathsf{a}\ \mathsf{b}), E = \mathsf{a}, A = \mathsf{b}, B = D = C = \mathsf{c}$.

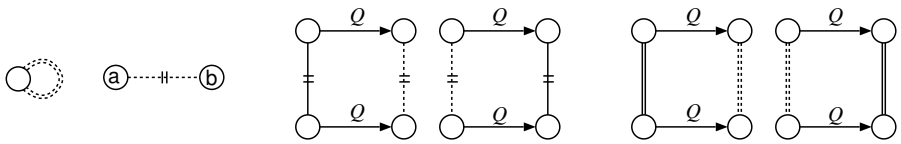

Fig. 4. Simplification rule diagrams for $(\approx^p_r), (\#^p_r), (\#^p_{\rightarrow}), (\#^p_{\leftarrow}), (\approx^p_{\rightarrow}), (\approx^p_{\leftarrow})$ (respectively)

In the rest of this section, we present and prove correct an algorithm for testing satisfiability for p-graphs based on this intuitive approach. We consider several rules for simplifying graphs, shown in Figure 4. Each diagram consists of a solid part and an edge formed with dotted lines. Such a diagram indicates that if G has a subgraph of the form described by the solid part, then the dotted edge should be added. These rules correspond to the following transformations on sets of formulas:

$$(\approx^p_{ref})\ S \rightarrow_p S, v \approx v \quad (E^p_{\rightarrow})\ \ S[v\ E\ v', Q{\cdot}v \approx w, Q{\cdot}v' \approx w'] \rightarrow_p S, w\ E\ w'$$
$$(\#^p_{irr})\ S \rightarrow_p S, \mathsf{a} \# \mathsf{b} \quad (E^p_{\leftarrow})\ S[w\ E\ w', Q{\cdot}v \approx w, Q{\cdot}v' \approx w'] \rightarrow_p S, v\ E\ v'$$

where in the $(\approx^p_{ref})$ and $(\#^p_{irr})$ rules, v or $\mathsf{a} \neq \mathsf{b}$ must be in S, respectively; in the $(E^p_{\rightarrow})$ and $(E^p_{\leftarrow})$ rules, $E \in \{\approx, \#\}$; and $S[S_1] \rightarrow_p S, S_2$ means "If S contains the formulas S_1, then add S_2 to S," .

Once the simplification rules above have been applied we can "collapse" equality edges involving variables, as outlined in the informal example. We say that a variable in G is solved if it appears in just one equality edge in G, otherwise it is unsolved; if $A \approx v \in G$ and A is unsolved then we can solve A in G by replacing A with v in all the other edges of G. This transformation on the graph corresponds to a variable elimination step on its corresponding constraint set:

$$(\approx^p_{var}) \quad S, A \approx v \rightarrow_p S[A := v], A \approx v \quad (\text{if } A \in FV(S),\ A \neq v)$$

We define $G[A := v]$ as the result of removing A from G and replacing A with v in all edges of G; a collapsing step on $A \approx v$ transforms G to $G[A := v], A \approx v$. We write $G \rightarrow_p G'$ if G can be transformed to G' via a simplification or collapsing step.

When considering satisfiability, solved variables can be ignored. The *collapsing* $c(G)$ of a graph G is the graph formed by eliminating all solved vertices. If $c(G) = G$, we say that G is fully collapsed.

Lemma 1. *If $A \approx v \in G$ then $Solv(G[A := v], A \approx v) = Solv(G)$. Moreover, $c(G)$ is satisfiable if and only if G is.*

Proof. If $\theta \vDash G$, then $\theta \vDash A \approx v$, hence $\theta \vDash A[A := v]$ for each $A \in G$ besides $A \approx v$. Conversely, if $\theta \vDash G[A := v], A \approx v$, then $\theta \vDash A \approx v$ and so $\theta \vDash A$ for each $A \in G$. The second part follows by induction on the number of solved variables in G.

We say that a p-graph is in *normal form* if none of the simplification or collapsing rules apply. A normalized graph is *solved* if in addition, $E_{\approx}$ and $E_{\#}$ are disjoint. We consider the possible forms of fully collapsed solved forms.

Proposition 1. *In a fully collapsed solved form, $E_{\approx} = Id_W$, and each E_Q is a partial injective function on W.*

Proof. Clearly $Id_W \subseteq E_{\approx}$ since otherwise $(\approx^p_{ref})$ would apply. Suppose $v \approx w \in G$. If v is a variable A, then w must also be A because otherwise v and w could be collapsed. The case in which w is a variable is symmetric. If v and w are names, then since G is normalized we must have $\mathsf{a} \# \mathsf{b} \in G$ and $\mathsf{a} \approx \mathsf{b} \notin G$ for any distinct names $\mathsf{a} \neq \mathsf{b}$, so it must be the case that $v = \mathsf{a} = w$.

For the second part, let $Q \in PV$ be given and consider $(v, w), (v, w') \in E_Q$. Since G is normalized, we must have $v \approx v \in G$ and $w \approx w' \in G$. By the first part, $w = w'$. Hence E_Q is a function. Moreover, by a similar argument if $(v, w), (v', w) \in E_Q$ then $v = v'$, so E_Q is injective.

Proposition 2. *A normalized p-graph is satisfiable if and only if it is solved.*

Proof. For the forward direction, we prove the contrapositive. If G is normalized but not solved, then there exists $(v, w) \in E_{\#} \cap E_{\approx}$. No valuation satisfies both $v \approx w$ and $v \# w$, so G is unsatisfiable.

For the reverse direction, it suffices to consider only normalized, fully collapsed graphs. By Proposition 1, G must satisfy $E_{\approx} = Id_W$, and E_Q must be an injective function on W for each Q.

Recall that $W = V \cup N$, where N is the set of names and $V = \{A_1, \ldots, A_k\}$ the set of variables of G. Let $\mathsf{b}_1, \ldots, \mathsf{b}_k$ be k names fresh for each other and not appearing in N. Define $\theta(A_i) = \mathsf{b}_i$ for $A_i \in V$. Note that θ is a bijection between V and $B = \{b_1, \ldots, b_k\}$. It extends to a bijection $\theta : W \to B \cup N$. If $v \approx w \in G$ then $v = w$ so clearly $\theta(v) \approx \theta(w)$. On the other hand, if $v \# w \in G$, we must have $v \neq w$, so $\theta(v) \# \theta(w)$ since θ is bijective. This shows that any valuation based on θ satisfies the edges $E_{\approx}$ and $E_{\#}$ of G.

Since each E_Q is injective it can be completed to a bijection $\pi_Q : W \to W$. Define $\theta(Q) = \theta \circ \pi_Q \circ \theta^{-1}$ for each Q. Suppose $(v, w) \in E_Q$. By construction, $\pi_Q(v) = w$, so

$$\theta(Q{\cdot}v) = \theta(Q)(\theta(v)) = \theta(\pi_Q(\theta^{-1}(\theta(v)))) = \theta(\pi_Q(v)) = \theta(w)$$

as desired. This completes the proof that the valuation θ satisfies G.

Theorem 1 (Soundness). *If $G \to_p G'$ then $Solv(G') = Solv(G)$.*

Proof. Suppose $G \to_p G'$. For a simplification step, $G' = G \cup A$ where A is either $v \approx w$, $v \# w$, or $v \xrightarrow{Q} w$. Trivially, $Solv(G') \subseteq Solv(G)$ since $G \subseteq G'$. To show that $Solv(G) \subseteq Solv(G')$, we need only verify that $G \vDash A$ in each case.

For the $(\approx^p_{ref})$ rule, $A = v \approx v$, and $G \vDash v \approx w$. For the $(\#^p_{irr})$ rule, $A = \mathsf{a} \# \mathsf{b}$ for names $\mathsf{a} \neq \mathsf{b}$, and clearly $G \vDash \mathsf{a} \# \mathsf{b}$. For the $(E^p_{\rightarrow})$ rule, we have $A = x'\ E\ y'$ and $v \xrightarrow{Q} v', w \xrightarrow{Q} w', v\ E\ w \in G$ for $E \in \{\approx, \#\}$. Then $\theta(v)\ E\ \theta(w)$, $\theta(Q)(\theta(w)) = \theta(w')$ and $\theta(Q)(\theta(v)) = \theta(v')$, so

$$\theta(v') = \theta(Q)(\theta(v))\ E\ \theta(Q)\theta(w) \approx \theta(w')$$

The cases for the $(E^p_{\leftarrow})$ rules are symmetric, since permutations are invertible and $\approx$ and $\#$ are equivariant. The case for a $(\approx^p_{var})$ step is shown in Lemma 1.

Theorem 2 (Termination). *There are no infinite sequences of simplification steps. Moreover, p-graph normalization can be performed in polynomial time.*

Proof. Each reduction step either adds an edge or solves a vertex in G, so the maximum number of steps is bounded above by $(2 + |PV|) \cdot |W|^2|V|$. Each reduction step can be identified and performed in polynomial time, and normalized and solved graphs can be recognized in polynomial time.

Corollary 1 (Completeness). *The relation $\rightarrow_p$ reduces any p-graph G to a normal form G' which is solved if and only if G is satisfiable; moreover, $Solv(G) = Solv(G')$.*

4 Equivariant Unification

In the previous section, we considered a very limited case of equivariant unification, namely solving systems of formulas of the form $v \approx w$, $v \# w$, and $Q{\cdot}v \approx w$. We showed that this problem can be solved in polynomial time using permutation graphs. In this section, we give an algorithm for reducing equivariant unification for arbitrary nominal terms to the problem of testing the satisfiability of a finite (but possibly exponential) number of permutation graphs. This algorithm can be easily be modified to obtain a nondeterministic polynomial time procedure for testing the satisfiability of such a problem, or as a procedure for enumerating the solutions one at a time.

We break the process into two phases. In the first phase, we simplify all problems involving subterms of the form $\langle\rangle, \langle t, u\rangle, \langle a\rangle t, f(t)$. After the first phase, the remaining satisfiable subproblems are of the form $a \# b, a \approx b$, where a, b are formed using only names, variables, and permutations. In the second phase, we convert these subproblems into p-graphs by eliminating permutations. Once each p-graph is constructed, we can test its satisfiability as shown in the previous section.

4.1 First phase

The first phase of the algorithm (defined as a relation $\rightarrow_1$) is presented as a collection of multiset rewriting rules in Figure 5. Each rule is of the form $S \rightarrow_1 \mathcal{M}$, and indicates that an extended problem $\mathcal{M}'; S$ should be rewritten to the problem $\mathcal{M}'; \mathcal{M}$.

A problem S is in *solved form* if it consists only of constraints of the form $a \# b$, $a \approx b$, or $X \approx t$ where X does not appear in t or elsewhere in S; an extended problem is solved if all its problems are solved. "Stuck" subproblems S that are unsolved and can take no transition can always be removed from an extended problem.

Example 2. The problem $Q{\cdot}(\langle \mathsf{a}\rangle\langle A, B\rangle) \approx \langle \mathsf{b}\rangle\langle \mathsf{b}, \mathsf{c}\rangle$ reduces to a solved form as follows:

$$\langle Q{\cdot}\mathsf{a}\rangle\langle Q{\cdot}A, Q{\cdot}B\rangle \approx \langle \mathsf{b}\rangle\langle \mathsf{b}, \mathsf{c}\rangle \rightarrow_1 \left\{ \begin{array}{c} Q{\cdot}\mathsf{a} \approx \mathsf{b}, \langle Q{\cdot}A, Q{\cdot}B\rangle \approx \langle \mathsf{b}, \mathsf{c}\rangle; \\ Q{\cdot}\mathsf{a} \# \langle \mathsf{b}, \mathsf{c}\rangle, \langle Q{\cdot}A, Q{\cdot}B\rangle \approx \langle (Q{\cdot}\mathsf{a}\ \mathsf{b}){\cdot}\mathsf{b}, (Q{\cdot}\mathsf{a}\ \mathsf{b}){\cdot}\mathsf{c}\rangle \end{array} \right\}$$
$$\rightarrow_1^* \left\{ \begin{array}{c} Q{\cdot}\mathsf{a} \approx \mathsf{b}, Q{\cdot}A \approx \mathsf{b}, Q{\cdot}B \approx \mathsf{c}; \\ Q{\cdot}\mathsf{a} \# \mathsf{b}, Q{\cdot}\mathsf{a} \# \mathsf{c}, Q{\cdot}A \approx (Q{\cdot}\mathsf{a}\ \mathsf{b}){\cdot}\mathsf{b}, Q{\cdot}B \approx (Q{\cdot}\mathsf{a}\ \mathsf{b}){\cdot}\mathsf{c} \end{array} \right\}$$

$$
\begin{array}{lrl}
(\approx_{\mathbf{1}}) & S, \langle\rangle \approx \langle\rangle & \to_1 S \\
(\approx_{\times}) & S, \langle t_1, t_2\rangle \approx \langle u_1, u_2\rangle & \to_1 S, t_1 \approx u_1, t_2 \approx u_2 \\
(\approx_{f}) & S, f(t) \approx f(u) & \to_1 S, t \approx u \\
(\approx_{abs}) & S, \langle a\rangle t \approx \langle b\rangle u & \to_1 \left\{ \begin{array}{c} S, a \approx b, t \approx u; \\ S, a \# u, t \approx (a\ b)\cdot u \end{array} \right\} \\
(\approx_{var}) & S, \Pi\cdot X \approx t & \to_1 S[X := \Pi^{-1}\cdot t], X \approx \Pi^{-1}\cdot t \\
 & & \text{(where } X \notin FV(t), X \in FV(S)\text{)} \\
(\#_{\mathbf{1}}) & S, a \# \langle\rangle & \to_1 S \\
(\#_{\times}) & S, a \# \langle u_1, u_2\rangle & \to_1 S, a \# u_1, a \# u_2 \\
(\#_{f}) & S, a \# f(u) & \to_1 S, a \# u \\
(\#_{abs}) & S, a \# \langle b\rangle u & \to_1 \left\{ \begin{array}{c} S, a \approx b; \\ S, a \# u \end{array} \right\}
\end{array}
$$

Fig. 5. Equivariant unification: phase one

Some constraints in a solved form may be of the form $\Pi\cdot X \approx \Pi'\cdot X$ where X is not a name-variable so cannot be substituted with names. These constraints are always satisfiable so can be set aside. This leaves *name–name* constraints $a \# b, a \approx b$ involving only permutations, names, and name variables.

Theorem 3 (Soundness). *If $\mathcal{M} \to_1 \mathcal{M}'$ then $Solv(\mathcal{M}) = Solv(\mathcal{M}')$.*

Proof. The cases for $(\approx_{\mathbf{1}})$, $(\approx_{\times})$, $(\approx_{f})$, and $(\approx_{var})$ are straightforward. For $(\approx_{abs})$, it suffices to show that $Solv(\langle a\rangle t \approx \langle b\rangle u) = Solv(a \approx b, t \approx u) \cup Solv(a \# u, t \approx (a\ b)\cdot u)$. Clearly, if $\theta \vDash a \approx b, t \approx u$, or $\theta \vDash a \# u, t \approx (a\ b)\cdot u$, then $\theta \vDash \langle a\rangle t \approx \langle b\rangle u$ using the rules in Figure 1. If $\theta \vDash \langle a\rangle t \approx \langle b\rangle u$, then there are two cases. If $\theta(a) = \theta(b)$, then $\theta \vDash a \approx b, t \approx u$, so $\theta \vDash \mathcal{M} \uplus \{a \approx b, t \approx u\}$. Otherwise, we must have $\theta \vDash a \# u, t \approx (a\ b)\cdot u$, so $\theta \vDash \mathcal{M} \uplus \{a \# u, t \approx (a\ b)\cdot u\}$.

The cases involving freshness are straightforward, with the reasoning for $(\#_{abs})$ similar to that for $(\approx_{abs})$.

Theorem 4 (Termination). *The relation $\to_1$ terminates.*

Proof. We define a measure on terms as follows: $\mu(\langle\rangle) \approx 1$, $\mu(\langle t, u\rangle) = \mu(t)+\mu(u)+1$, $\mu(f(t)) = \mu(f) + 1, \mu(\langle a\rangle t) = \mu(t) + 1$, $\mu(\Pi\cdot X) = \mu(\Pi\cdot \mathsf{a}) = 0$. Let $\mu(t\ E\ u) = \mu(t)+\mu(u)$ and $\mu(S) = \sum_{A\in S} \mu(A)$. Let $\mu'(S)$ be the number of unsolved variables in S. Define $\nu(S) = (\mu'(S), \mu(S))$ and $\nu(\mathcal{M}) = \{\nu(S) \mid S \in \mathcal{M}\}$. It is straightforward to verify that if $P \to_1 P'$ then $\nu(\mathcal{M}) > \nu(\mathcal{M}')$ in the multiset order generated by the lexicographic order on $\mathbb{N} \times \mathbb{N}$.

Lemma 2. *If $\mathcal{M}$ is satisfiable and $\to_1$-normalized, then $\mathcal{M}$ is in solved form.*

Proof. We prove that if $\mathcal{M}$ is unsolved and satisfiable, it is not normalized. Suppose $\mathcal{M}$ is satisfiable but not solved. Then there must be some constraint in $\mathcal{M}$ which is not of the form $a \# b, a \approx b$, or $X \approx t$ where X is solved in $\mathcal{M}$. If the constraint is of the form $\Pi\cdot X \approx t$ where X is a term variable and t starts with a term symbol, then we must have $\theta(\Pi\cdot X) \approx \theta(t)$, which can only be the case if X does not appear in t, so $(\approx_{var})$ applies. Otherwise, the constraint must be of the form $t \approx u$ or $a \# u$, where

$$
\begin{array}{lrl}
(id) & S[\mathsf{id}\cdot v] & \to_2 S[v] \\
(inv) & S[\Pi^{-1}\cdot v] & \to_2 \exists X.S[X], \Pi\cdot X \approx v \\
(comp) & S[\Pi \circ \Pi'\cdot v] & \to_2 \exists X.S[\Pi\cdot X], \Pi'\cdot v \approx X) \\
(swap) & S[(a\ a')\cdot v] & \to_2 \left\{ \begin{array}{c} S[a], a' \approx v; \\ S[a'], a \approx v; \\ \exists X.S[X], v \approx X, a \# X, a' \# X \end{array} \right\} \\
(\#_Q) & S, Q\cdot v \# w & \to_2 \exists X.S, Q\cdot v \approx X, X \# w
\end{array}
$$

Fig. 6. Equivariant unification: phase two.

t, u start with term symbols. For the case of $a \# u$, a step can be taken no matter which term symbol is at the head of u. For $t \approx u$, since $\theta \vDash t \approx u$, the head symbols of t and u must match, so that we can take a step. In any case, $\mathcal{M} \to \mathcal{M}'$ for some $\mathcal{M}'$.

Corollary 2 (Completeness). *The relation $\to_1$ reduces any finite equivariant unification problem to a finite complete set of pre-solutions.*

4.2 Second phase

In the second phase, we reduce name–name constraints to p-graphs whose satisfiability can be checked easily. As a preprocessing step, we assume that all constraints of the form $\Pi_1\cdot v \approx \Pi_2\cdot w$ or $\Pi_1\cdot v \# \Pi_2\cdot w$ are normalized to $(\Pi_2^{-1} \circ \Pi_1)\cdot v \approx w$ or $(\Pi_2^{-1} \circ \Pi_1)\cdot v \# w$ respectively. This is without loss of generality because $\#$ and $\approx$ are preserved by applying permutations to both sides. The rules for the second phase of equivariant unification shown in Figure 6 reduce the results of the first phase to a form suitable for satisfiability checking via p-graphs. In several rules, we introduce fresh existentially-quantified variables; these are required not to already appear in the problem.

Example 3. We continue Example 2. The first subproblem, $Q\cdot\mathsf{a} \approx \mathsf{b}, Q\cdot A \approx \mathsf{b}, Q\cdot B \approx \mathsf{c}$, is already in solved form (and is satisfiable provided $A \approx \mathsf{a}$). The second problem reduces as follows:

$$
\begin{array}{ll}
 & Q\cdot A \approx (Q\cdot\mathsf{a}\ \mathsf{b})\cdot\mathsf{b}, Q\cdot B \approx (Q\cdot\mathsf{a}\ \mathsf{b})\cdot\mathsf{c}, Q\cdot\mathsf{a} \# \mathsf{b}, Q\cdot\mathsf{a} \# \mathsf{c} \\
\to_2 & (Q\cdot\mathsf{a}\ \mathsf{b}) \circ Q\cdot A \approx \mathsf{b}, (Q\cdot\mathsf{a}\ \mathsf{b}) \circ Q\cdot B \approx \mathsf{c}, S \\
\to_2 & (Q\cdot\mathsf{a}\ \mathsf{b})\cdot C_1 \approx \mathsf{b}, Q\cdot A \approx C_1, (Q\cdot\mathsf{a}\ \mathsf{b})\cdot C_2 \approx \mathsf{c}, Q\cdot B \approx C_2, S \\
\to_2 & \left\{ \begin{array}{ll} & Q\cdot\mathsf{a} \approx C_1, \mathsf{b} \approx \mathsf{b}; \\ (*) & Q\cdot\mathsf{a} \approx \mathsf{b}, \mathsf{a} \approx C_1; \\ (*) & Q\cdot\mathsf{a} \# C_1, \mathsf{b} \# C_1, C_1 \approx \mathsf{b} \end{array} \right\} \otimes \left\{ \begin{array}{ll} (*) & Q\cdot\mathsf{a} \approx C_2, \mathsf{b} \approx \mathsf{c}; \\ (*) & \mathsf{a} \approx C_2, Q\cdot\mathsf{a} \approx \mathsf{c}; \\ & Q\cdot\mathsf{a} \# C_2, \mathsf{b} \# C_2, C_2 \approx \mathsf{c} \end{array} \right\} \otimes \{S'\}
\end{array}
$$

where $S = Q\cdot\mathsf{a} \# \mathsf{b}, Q\cdot\mathsf{a} \# \mathsf{c}$ and $S' = Q\cdot A \approx C_1, Q\cdot B \approx C_2, S$, and $\mathcal{M} \otimes \mathcal{M}'$ denotes $\{T \wedge T' \mid T \in \mathcal{M}, T' \in \mathcal{M}'\}$. There are a total of nine cases; however, the starred subproblems are unsatisfiable, so there is only one solution.

Theorem 5 (Soundness). *If $\mathcal{M} \to_2 \mathcal{M}'$ then $Solv(\mathcal{M}) = Solv(\mathcal{M}')$.*

Proof. There are several cases, one for each rule replacing $\mathcal{M}, S$ with $\mathcal{M}, \mathcal{M}'$ for $S \rightarrow_2 \mathcal{M}'$. The cases for (id), (inv), $(comp)$, and $(\#_Q)$ are straightforward. For the $(swap)$ rule, it suffices to show that $Solv(S[(a\ b)\cdot v]) = T = Solv(S[b], a \approx v) \cup Solv(S[a], b \approx v) \cup Solv(\exists X.S[X], v \approx X, a \# X, b \# X)$. If $\theta \in Solv(S[(a\ b)\cdot v])$, then there are three cases. If $\theta \vDash a \approx v$, then $\theta \vDash (a\ b)\cdot v \approx b$ so $\theta \in Solv(S[b], a \approx v)$. The case for $\theta \vDash b \approx v$ is symmetric. If $\theta \vDash a \# v, b \# v$, then $\theta \vDash (a\ b)\cdot v \approx v$ so $\theta[X := \theta(v)] \vDash X \approx v, a \# X, b \# X, S[X]$, and $\theta \in Solv(\exists X.S[X], v \approx X, a \# X, b \# X)$. So in any case $\theta \in T$. The reverse direction, $T \subseteq Solv(S[(a\ b)\cdot v])$, is straightforward.

Theorem 6 (Termination). *The relation $\rightarrow_2$ terminates.*

Proof. We employ a measure μ that measures the complexity of the permutation terms remaining in $\mathcal{M}$. We define $\mu(v) \approx 0$, $\mu(\Pi\cdot v) = \mu(\Pi)$, $\mu((a\ b)) = 1 + \mu(a) + \mu(b)$, $\mu(\Pi \circ \Pi') = \mu(\Pi) + \mu(\Pi')$, $\mu(\Pi^{-1}) = \mu(\Pi) + 1$, and $\mu(\mathsf{id}) = 1$. In addition, $\mu(a \# v) = \mu(a \approx v) = \mu(a)$, $\mu(S) = \sum_{A \in S} \mu(A)$, and $\mu(\mathcal{M}) = \{\mu(S) \mid S \in \mathcal{M}\}$. If $\mathcal{M} \rightarrow_2 \mathcal{M}'$, then $\mu(\mathcal{M})$ is decreasing in the multiset ordering generated by $>_{\mathbb{N}}$.

Lemma 3. *If $\mathcal{M}$ is $\rightarrow_2$-normalized problem, then it is in solved form.*

Proof. Since $\mathcal{M}$ is normalized, it cannot contain any constraints of the form $\Pi\cdot v\ E\ w$ where $E \in \{\#, \approx\}$ and Π is not a variable, since otherwise one of the rules (id), $(comp)$, (inv), $(swap)$ can be applied. Similarly, $\mathcal{M}$ cannot contain a constraint of the form $Q\cdot v \# w$, since otherwise $(\#_Q)$ applies. Because only constraints of the form $v \approx w$, $v \# w$, and $Q\cdot v \approx w$ remain, $\mathcal{M}$ is in solved form.

Corollary 3 (Completeness). *The relation $\rightarrow_2$ reduces any finite name–name problem to a finite complete set of pre-solutions.*

Example 4. Consider the query $\vdash spec(all(\langle \mathsf{a} \rangle tvar(\mathsf{a})), [\mathsf{b}], U)$. Equivariant unification against a suitably renamed/permuted head clause $P\cdot spec(all(\langle \mathsf{a}' \rangle T'), \mathsf{a}' :: L', U')$ yields a single unifier $P\cdot \mathsf{a}' \approx \mathsf{b}, T' := tvar(P^{-1} \circ (\mathsf{a}\ P\cdot \mathsf{a}')\cdot \mathsf{a}), L' := [], U := P\cdot U'$. The resulting subgoal $spec(tvar(P^{-1} \circ (\mathsf{a}\ P\cdot \mathsf{a}')\cdot \mathsf{a}), [], U')$ produces the unique solution $U' := tvar(P^{-1} \circ (\mathsf{a}\ P\cdot \mathsf{a}')\cdot \mathsf{a})$. This gives the overall solution $U := tvar(P \circ P^{-1} \circ (\mathsf{a}\ P\cdot \mathsf{a}')\cdot \mathsf{a})$, which can be simplified to $U := tvar(\mathsf{b})$ since $P\cdot \mathsf{a}' \approx \mathsf{b}$.

5 Swapping-Free Equivariant Matching

In equivariant unification, only the abstraction and swapping operations cause branching. This implies (perhaps surprisingly) that equivariant unification is tractable for problems involving names, term symbols, and freshness but not abstraction and swapping. If we restrict attention to equivariant matching of grounded terms, however, we can get a stronger result: swapping-free grounded terms can be matched efficiently. We consider grounded problems of the form $t \leq u$, where u is ground; a solution is a ground substitution θ and ground permutation π such that $\theta(t) \approx \pi\cdot u$.

When one side of an equation is ground, the structure of the bound names on that side must be mirrored exactly on the other side. For example, consider the problem

$$
\begin{array}{lr}
(\leq_{\mathbf{1}}) & S, l.t \leq l'.\langle\rangle \rightarrow_m S, t \approx \langle\rangle \\
(\leq_f) & S, l.t \leq l'.f(u) \rightarrow_m \exists X.S, l.X \leq l'.u, t \approx f(X) \\
(\leq_\times) & S, l.t \leq l'.\langle u_1, u_2\rangle \rightarrow_m \exists X_1, X_2.S, l.X_1 \leq l'.u_1, l.X_2 \leq l'.u_2, t \approx \langle X_1, X_2\rangle \\
(\leq_{abs}) & S, l.t \leq l'.\langle \mathsf{a}\rangle u \rightarrow_m \exists X.S, l\mathsf{b}.X \leq l'\mathsf{a}.u, t \approx \langle \mathsf{b}\rangle X \quad (\mathsf{b} \notin FN(S)) \\
(\leq_\approx) & S, l\mathsf{a}.v \leq l'\mathsf{b}.\mathsf{b} \rightarrow_m S, v \approx \mathsf{a} \\
(\leq_\#) & S, l\mathsf{a}.v \leq l'\mathsf{b}.\mathsf{c} \rightarrow_m S, l.v \leq l'.\mathsf{c}, \mathsf{a} \# v \quad (\mathsf{b} \neq \mathsf{c})
\end{array}
$$

Fig. 7. Swapping-free equivariant matching

$\langle \mathsf{a}\rangle\langle \mathsf{b}\rangle X \leq \langle \mathsf{c}\rangle\langle \mathsf{d}\rangle e$, where X is a name-variable and e is a ground name. If $e = \mathsf{d}$, then we must have $X = \mathsf{b}$; if $e = \mathsf{c}$, then we must have $X = \mathsf{a}$; and if e is some name other than c, d then we must have $\mathsf{a}, \mathsf{b} \# X$ and $Q{\cdot}X = e$.

Also, in a problem of the form $\langle \mathsf{a}_1\rangle\cdots\langle \mathsf{a}_n\rangle X \leq \langle \mathsf{b}_1\rangle\cdots\langle \mathsf{b}_n\rangle t$, if t starts with unit, pairing, or a function symbol f, then X must also start with unit, pairing, or f, so we can proceed by *simulating* the head symbol of t by making an appropriate substitution of $X \approx \langle\rangle$, $X \approx \langle X_1, X_2\rangle$, or $X \approx f(X')$, where X_1, X_2, X' are new variables. More generally, if the problem is of the form $\langle \mathsf{a}_1\rangle\cdots\langle \mathsf{a}_n\rangle t \leq \langle \mathsf{b}_1\rangle\cdots\langle \mathsf{b}_n\rangle u$, then we can proceed by unifying t with $\langle\rangle$, $\langle X_1, X_2\rangle$, or $f(X')$, as appropriate.

Based on this intuition, we propose the following algorithm for matching swapping-free grounded terms with ground nominal terms. We write l, l' for lists of names $\mathsf{a}_1 \cdots \mathsf{a}_n$ and consider problems of the form $l.t \leq l'.u$ where u is ground. This problem is equivalent to the problem $\langle a_1\rangle\cdots\langle a_n\rangle t \leq \langle b_1\rangle\cdots\langle b_n\rangle u$.

The rules in Figure 7 define a relation $\rightarrow_m$ that reduces equivariant matching problems to the form $S_\leq \cup S_{NP}$, where $S_\leq$ is a collection of inequalities of the form $[].v \leq [].\mathsf{a}$, and S_{NP} is a collection of equality and freshness constraints among grounded terms. The satisfiability of S_{NP} can be tested using grounded nominal unification; if successful, this results in a unifier $\langle \nabla, \sigma\rangle$, where ∇ is a set of freshness constraints and σ is a substitution. Now let Q be a permutation variable, let $S_Q = \nabla \cup \{Q{\cdot}\sigma(v) \approx \mathsf{a} \mid [].v \leq [].\mathsf{a} \in S_\leq\}$, and test the satisfiability of the p-graph G_{S_Q}.

We now state the important properties of $\rightarrow_m$. The proofs are complicated without being particularly enlightening so are omitted.

Theorem 7 (Soundness). *If $S \rightarrow_m S'$, then $Solv(S) = Solv(S')$.*

Theorem 8 (Termination). *The relation $\rightarrow_m$ is terminating, and normal forms can be computed in polynomial time.*

Lemma 4. *If S is a normalized swapping-free equivariant matching problem, then S is in solved form.*

Theorem 9 (Completeness). *The relation $\rightarrow_m$ reduces any equivariant matching problem S to a pre-solution S' such that $Solv(S) = Solv(S')$.*

Example 5. Recall the problem $subst(var(\mathsf{a}), T, \mathsf{a}) \leq subst(var(\mathsf{b}), var(\mathsf{a}), \mathsf{b})$ mentioned in the Introduction. The above algorithm reduces to the solved form $\mathsf{a} \leq \mathsf{b}, T := var(B), B \leq \mathsf{a}$. Since $subst(var(\mathsf{a}), T, \mathsf{a})$ rewrites to $T = var(B)$ and $B \leq \mathsf{a}$, we rewrite $subst(var(\mathsf{b}), var(\mathsf{a}), \mathsf{b})$ to $var(\mathsf{a})$.

6 Related and Future Work

Many researchers (see for example [9]) have studied the problem of E-unification, or unification with respect to a general equational theory E. However, nominal logic poses some unique challenges to standard E-unification techniques based on confluent rewrite systems. One reason is that it appears that the equivariance principle $p(\bar{x}) \approx p((a\ b)\cdot\bar{x})$ cannot be directed so as to obtain a confluent rewrite system for nominal terms.

There also may be an interesting connection between equivariant unification and unification modulo equational laws having to do with name-restriction in the π-calculus, such as $\nu a.p \equiv p$ (where $a \notin FN(p)$) and $\nu a.\nu b.p \equiv \nu b.\nu a.p$. More generally, it may be interesting to study E-unification for equational theories specified in nominal logic.

Cheney [2], Fernandez, Gabbay and Mackie [4], and Urban and Cheney [10] have developed increasingly general tests for identifying rewriting systems or logic programs for which nominal unification is adequate. These results demonstrate that nominal unification can often be used instead of equivariant unification to execute programs efficiently. Such special cases should be recognized and exploited whenever possible.

FreshML [8] is a ML-like functional programming language based on nominal abstract syntax. In FreshML, programs can perform pattern matching against terms involving abstraction and name-variables but not constant names, swappings; also, such pattern matching is not modulo equivariance. However, as usual in ML, variables may appear at most once in patterns, so the matching problems involved in FreshML can be solved efficiently and without backtracking: for example, to match u against an abstraction $\langle A\rangle t$, it suffices to generate a fresh name c and match u against $\langle \mathsf{c}\rangle t$. It would be interesting to see whether constant names could be incorporated into FreshML-style functional programming.

In logic programming, nondeterminism is often a bigger performance problem than exponential worst-case complexity, so it would be worthwhile to find ways of avoiding duplicate answers, delaying nondeterministic search, and detecting failure early. One possibility is to look for and factor out symmetries in subproblems as soon as they appear. Another step in this direction is to replace the $(\approx_{abs})$ and $(\#_{abs})$ rules with

$$\begin{array}{ll}(\approx_{abs}) & S, \langle a\rangle t \approx \langle b\rangle u \rightarrow_1 S, И\mathsf{c}.(a\ \mathsf{c})\cdot t \approx (b\ \mathsf{c})\cdot u \\ (\#_{abs}) & S, a \# \langle b\rangle u \rightarrow_1 S, И\mathsf{c}.a \# (b\ \mathsf{c})\cdot u\end{array}$$

where $\mathsf{c} \notin FN(a, b, t, u)$ and И is nominal logic's "new" or "fresh name" quantifier. This is correct because in nominal logic, $\langle a\rangle t \approx \langle b\rangle u \iff И\mathsf{c}.(a\ \mathsf{c})\cdot t \approx (b\ \mathsf{c})\cdot u$ and $a \# \langle b\rangle u \iff И\mathsf{c}.a \# (b\ \mathsf{c})\cdot u$. This approach concentrates the nondeterminism in name–name constraints, which suggest that a practical approach may be to delay attempts to solve such constraints as long as possible.

We expressed equivariant unification in terms of permutation terms and variables. In contrast, in nominal logic, only the swapping operator is present; general permutations are not. It is not clear how solutions involving permutation variables produced by our algorithm relate to nominal logic. Thus, it would be an advantage if permutation variables could be eliminated from the results of logic programming queries. This issue needs to be investigated.

We have developed a prototype implementation of equivariant unification using Constraint Handling Rules [5], which are available in many Prolog implementations.

This helped identify some subtle issues and is a first step towards incorporating nominal abstract syntax into standard logic programming languages.

7 Conclusions

Equivariant unification and matching are computationally hard problems requiring subtle algorithmic techniques. Solutions to these problems are necessary for complete implementations of nominal rewriting and logic programming. This paper makes two contributions building upon an important technical device called *permutation graphs*. We present an equivariant unification algorithm, the first terminating algorithm for this problem. This algorithm can be viewed as a nondeterministic polynomial time algorithm for reducing equivariant unification problems to finite complete sets of solutions. It is evident from the structure of the algorithm that the only sources of nondeterminism in equivariant unification are swappings and abstractions. Based on this observation, we developed an algorithm for efficient matching of swapping-free grounded terms. This algorithm can be used to run interesting nominal rewrite systems that do not work properly using nominal unification alone. However, there are several potential efficiency problems which will need to be addressed for equivariant unification to be practical.

Acknowledgments: This work was supported by EPSRC grant R37476. The author wishes to thank the anonymous reviewers for their comments.

References

1. J. Cheney and C. Urban. Alpha-Prolog: A logic programming language with names, binding and alpha-equivalence. In *Proc. 20th Int. Conf. on Logic Programming (ICLP 2004)*, number 3132 in LNCS, pages 269–283, 2004.
2. James Cheney. The complexity of equivariant unification. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *LNCS*, pages 332–344. Springer-Verlag, 2004.
3. James R. Cheney. *Nominal Logic Programming*. PhD thesis, Cornell University, Ithaca, NY, August 2004.
4. Maribel Fernández, Murdoch Gabbay, and Ian Mackie. Nominal rewriting systems. In *Proceedings of the 6th Conference on Principles and Practice of Declarative Programming (PPDP 2004)*, 2004. To appear.
5. Thom Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1–3):95–138, October 1998.
6. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.
7. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 183:165–193, 2003.
8. M. R. Shinwell, A. M. Pitts, and M. J. Gabbay. FreshML: Programmming with binders made simple. In *Proc. 8th ACM SIGPLAN Int. Conf. on Functional Programming (ICFP 2003)*, pages 263–274, Uppsala, Sweden, 2003. ACM Press.
9. Wayne Snyder. *A Proof Theory for General Unification*, volume 11 of *Progress in Computer Science and Applied Logic*. Birkhäuser, 1991.
10. C. Urban and J. Cheney. Avoiding equivariance in alpha-Prolog. To appear, 2004.
11. C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.