The Complexity of Equivariant Unification

James Cheney

Cornell University
(jcheney@cs.cornell.edu)

Abstract. Nominal logic is a first-order theory of names and binding based on a primitive operation of *swapping* rather than substitution. Urban, Pitts, and Gabbay have developed a *nominal unification* algorithm that unifies terms up to nominal equality. However, because of nominal logic's *equivariance principle*, atomic formulas can be provably equivalent without being provably equal as terms, so resolution using nominal unification is sound but incomplete. For complete resolution, a more general form of unification called *equivariant unification*, or "unification up to a permutation" is required. Similarly, for rewrite rules expressed in nominal logic, a more general form of matching called *equivariant matching* is necessary.
In this paper, we study the complexity of the decision problem for equivariant unification and matching. We show that these problems are **NP**-complete in general. However, when one of the terms is essentially first-order, equivariant and nominal unification coincide. This shows that equivariant unification can be performed efficiently in many interesting common cases: for example, any purely first-order logic program or rewrite system can be run efficiently on nominal terms.

1 Introduction

Nominal logic [13] is a first-order theory of names and binding formalizing the novel Gabbay-Pitts approach to abstract syntax with binding inspired by Fraenkel-Mostowski permutation models of set theory [6]. In nominal logic, names are modeled as *atoms* a, b drawn from a countable set $\mathbb{A}$. Atoms can be tested for equality ('$a = b$') or freshness relative to other terms ('$a \# t$'), bound in abstractions ('$\langle a \rangle t$'), and used in swaps acting on terms ('$(a\ b) \cdot t$'). Nominal logic can serve as a foundation for specifying and reasoning about logics and programming languages encoded using nominal terms and relations; we call this approach to representing such languages *nominal abstract syntax*.

The state of the art of reasoning about languages with binding is *higher-order abstract syntax* [12] (HOAS), in which object-language variables and binders are encoded as meta-variables and λ-abstraction in a higher-order metalanguage. For example, in HOAS, an object-term $\lambda X.F\ X$ would be translated to a meta-language expression $lam(\lambda X.app\ F\ X)$, where $app : exp \to exp \to exp$ and $lam : (exp \to exp) \to exp$ are constants. In contrast, in nominal abstract syntax, variables and binders are translated to atoms $a \in \mathbb{A}$ and atom-abstractions $\langle a \rangle t \in \langle \mathbb{A} \rangle T$; abstractions are considered equal up to α-equivalence. For example,

an object-term $\lambda X.F\ X$ is translated to $lam(\langle x \rangle app(var(f), var(x)))$, where $x, f : \mathbb{A}$, $var : \mathbb{A} \to exp$, $lam : \langle \mathbb{A} \rangle exp \to exp$, and app is as before.

Nominal logic is of interest because it may be much easier to reason about languages with binding using its first-order techniques than using higher-order techniques. For example, unification up to equality in nominal logic is efficiently decidable and unique most general unifiers (MGUs) exist [15], whereas unification up to equality in higher-order logic is undecidable and MGUs may not exist. However, higher-order unification is practically useful despite these theoretical drawbacks: Huet's semi-unification algorithm [9] performs well in practice, and higher-order unification is decidable in linear time and has unique MGUs for the broad special case of *higher-order patterns* [11]. A more serious problem is that reasoning by induction about languages with constructors like $lam : (\underline{exp} \to exp) \to exp$ is difficult because of the (underlined) negative occurrence of exp (see for example Hofmann [8] for a category-theoretic analysis of this problem). In contrast, there is no such negative occurrence in the nominal abstract syntax encoding $lam : \langle \mathbb{A} \rangle exp \to exp$, and induction principles can be derived directly from nominal language specifications (see [6, 13]).

In this paper we consider a significant technical problem with automating reasoning in nominal logic. The *resolution principle* [14] is an important tool in automated deduction and logic programming. It states that given $A \vee P$ and $\neg B \vee Q$, where A, B are atomic and $A \Leftrightarrow B$, we can conclude $P \vee Q$. In first- or higher-order logic, atomic formulas are equivalent precisely when they are equal as first- or higher-order terms, respectively; moreover, we can decide whether and how two atomic formulas A, B can be instantiated to be logically equivalent simply by unifying them. Thus, resolution in these logics reduces to unification.

This is not the case in nominal logic because atomic formulas may be logically equivalent but not equal as nominal terms. This is because of nomial logic's *equivariance principle*, whic states that the validity of an atomic formula is preserved by applying swaps uniformly to its arguments:

$$R(\bar{X}) \Rightarrow R((a\ b) \cdot \bar{X})\ .$$

Since usually $R((a\ b) \cdot \bar{X}) \neq R(\bar{X})$, atomic formulas may differ as nominal terms but still be logically equivalent. For example, if R' is a binary relation, then $R'(a, b) \Leftrightarrow R'(c, a)$ for distinct atoms a, b, and c, because we can convert from one to the other using the swaps $(b\ a)(a\ c)$; but $R'(a, b)$ and $R'(c, a)$ are not equal or even unifiable. Indeed, the following theorems of nominal logic:

$$\forall a, b : \mathbb{A}.R'(a, a) \Rightarrow R'(b, b)$$
$$\forall a, b, c, d : \mathbb{A}.a \neq b \wedge c \neq d \Rightarrow R'(a, b) \Rightarrow R'(c, d)$$

imply that there are essentially only four binary relations on atoms in nominal logic, determined by their behavior on and off the diagonal of $\mathbb{A} \times \mathbb{A}$ (namely, the total and empty relations, $=$, and $\#$).

Rewriting rules defined in nominal logic are also subject to equivariance. For example, nominal rewrite rules such as

$$sub(var(a), a, T) \to T \quad sub(var(b), a, T) \to var(b)$$

define some cases for a substitution function. Here, T is a variable whereas $a, b \in \mathbb{A}$ are distinct *atom constants*, so the two rules do not overlap (as they would if a, b were variables that could be unified). To rewrite an expression like $sub(var(c), c, var(a))$ to $var(a)$, we must match

$$sub(var(a), a, T) \lesssim^? sub(var(c), c, var(a))$$

These terms do not nominally match because the atoms c and a clash. However, by equivariance the first rule is still true if we apply the permutation $(a\ c)$ to it, yielding $sub(var(c), c, (a\ c) \cdot T) \to (a\ c) \cdot T$. This rule's left-hand side does match $sub(var(c), c, var(a))$ via substitution $[T := v(c)]$, so we can rewrite the term to $T = (a\ c) \cdot v(c) = v(a)$, as desired.

In order to obtain a complete resolution procedure, a new form of *equivariant unification* that unifies "up to a permutation" is required. Similarly, for nominal term rewriting rules involving atoms, an *equivariant matching* algorithm that matches a term to a ground term "up to a permutation" is needed. The aim of this paper is to study the complexity of the underlying decision problems of determining whether an equivariant unification or matching problem has a solution. In order to simplify matters, we consider only a special case, that for equivariance over sequences of terms of sort $\mathbb{A}$. Despite its apparent simplicity, all the computational complexity resides in this case (though the details of the reduction are beyond the scope of this paper).

Our main results are that equivariant matching and satisfaction are both **NP**-complete. Thus, the situation is not as good as for first-order, nominal or higher-order pattern unification, and in particular, equivariant unification cannot be reduced to nominal or higher-order pattern unification unless $\mathbf{P} = \mathbf{NP}$. However, in practice the situation may not be so bad. We identify an important special case which is in **P**: If two terms have no variables in common and one does not mention atoms or swaps, then equivariant unification reduces to nominal unification. This result can be generalized to show that ordinary first-order logic programs or rewrite rules can be applied to nominal terms efficiently using nominal unification.

2 Fundamentals

We use the notation $\boldsymbol{x}$ for an *n-tuple* (or *sequence*, when n is not important) $(x_1, \ldots, x_n) \in X^n$ of elements of a set X.

Fix a countable set $\mathbb{A} = \{a_1, a_2, \ldots, \}$ of *atoms*. Recall that a (finitary) permutation of $\mathbb{A}$ is a bijection $\pi : \mathbb{A} \to \mathbb{A}$ that moves at most finitely many elements of $\mathbb{A}$. The *support* of a permutation is the set of atoms it moves: $supp(\pi) = \{a \in \mathbb{A} \mid \pi(a) \neq a\}$; a permutation is finitary if and only if it has finite support. The *finitary permutation group over* $\mathbb{A}$, written $FSym(\mathbb{A})$, is the permutation group consisting of all finitary permutations of $\mathbb{A}$. Henceforth in this paper, all permutations are taken to be elements of $FSym(\mathbb{A})$, and we omit the adjective 'finitary'.

We write id for the identity permutation and write other permutations in transposition notation $(b_1\ c_1)\cdots(b_n\ c_n)$, where each $b_i, c_i \in \mathbb{A}$. In this notation, functional composition $\pi \circ \pi'$ is just the concatenation of π, π' as transposition lists. Permutations are equal when they denote the same function; equivalently, $\pi = \pi'$ when $supp(\pi \circ \pi'^{-1}) = \emptyset$. For example, $(a\ b) = (c\ d)(b\ a)(d\ c)$. We write $\pi \cdot_{\mathbb{A}} a$ for the result of applying π to a. For example $(a\ b) \cdot_{\mathbb{A}} a = b$ and $(a\ b) \cdot_{\mathbb{A}} c = c$ if $c \notin \{a, b\}$. Permutations act componentwise on sequences and sets of atoms: $\pi \cdot_{\mathbb{A}^n} (b_1, \ldots, b_n) = (\pi \cdot_{\mathbb{A}} b_1, \ldots, \pi \cdot_{\mathbb{A}} b_n)$, $\pi \cdot_{\mathcal{P}(\mathbb{A})} B = \{\pi \cdot_{\mathbb{A}} b \mid b \in B\}$. We omit the subscript on '$\cdot$' when there is no ambiguity.

One convenient property of $FSym(\mathbb{A})$ is that given a finite subset of $\mathbb{A}$, we can always find a disjoint finite subset of $\mathbb{A}$ (or a finite family of pairwise disjoint subsets) of the same size, together with permutations translating between them.

Proposition 1. *Suppose $B \subset \mathbb{A}$ is finite. Then there exists a permutation $\pi \in FSym(\mathbb{A})$ such that $\pi \cdot B$ and B are disjoint. More generally, if I is a finite index set, then there exists a family $(\tau_i \in FSym(\mathbb{A}) \mid i \in I)$ such that every pair of sets in $\{B\} \cup \{\tau_i \cdot B \mid i \in I\}$ is disjoint.*

Proof. The first part follows from Neumann's Lemma for the group $FSym(\mathbb{A})$ (see for example [2], section 6.2 for a proof). The second part follows from the first by (a slightly stronger) induction on the size of I. QED.

Definition 1. *Two sequences $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{A}^n$ are* equivariant *(written $\boldsymbol{a} \sim \boldsymbol{b}$) if there is a permutation $\pi \in FSym(\mathbb{A})$ such that $\pi \cdot \boldsymbol{a} = \boldsymbol{b}$.*

Example 1. For example, $(a, b) \sim (b, c)$ as witnessed by $(a\ b)(b\ c)$, $(a, a) \not\sim (b, d)$, and

$$(a, b, a, c, a, d) \sim (c, d, c, b, c, a)$$

as witnessed by $(a\ c)(c\ b)(b\ d)$.

Note that equivariance is obviously an equivalence relation; its equivalence classes are *orbits* of $FSym(\mathbb{A})$ acting on $\mathbb{A}^n$, in group-theoretic terms. It is important to note that equivariance is not a congruence with respect to pairing (or in general, composition of sequences): for example, $a \sim b$ and $a \sim c$ but $(a, a) \not\sim (b, c)$.

Let $\mathbb{V} = \{X, Y, \ldots\}$ be a countable set of variables. Terms s, t are either atoms $a \in \mathbb{A}$, or *suspensions* $\pi \cdot X$, where $\pi \in FSym(\mathbb{A})$, and $X \in \mathbb{V}$. The set of all terms is $\mathbb{T}$. The functions $V : \mathbb{T} \to \mathcal{P}(\mathbb{V})$ and $A : \mathbb{T} \to \mathcal{P}(\mathbb{A})$ calculate the sets of variables and atoms appearing in a term respectively. When the suspended permutation is id, the suspension $\mathsf{id} \cdot X$ is abbreviated to X. A term or sequence of terms is *ground* if no variables occur in it.

A *valuation* is a function $\theta : \mathbb{V} \to \mathbb{A}$. Valuations are extended to terms $\theta_{\mathbb{T}} : \mathbb{T} \to \mathbb{A}$ as follows:

$$\theta_{\mathbb{T}}(a) = a \qquad \theta_{\mathbb{T}}(\pi \cdot X) = \pi \cdot_{\mathbb{A}} \theta(X)$$

Suspended permutations come into effect after a valuation has been applied. Valuations operate componentwise on n-tuples: $\theta_{\mathbb{T}^n}(s_1, \ldots, s_n) = (\theta_{\mathbb{T}}(s_1), \ldots, \theta_{\mathbb{T}}(s_n))$.

We omit the subscript on θ when there is no possibility of confusion. We write valuations using shorthand such as $\theta = [X := a, Y := b]$. For example

$$\theta(X, (a\ b) \cdot X, Y, (a\ c) \cdot Y) = (a, (a\ b) \cdot_{\mathbb{A}} a, b, (a\ c) \cdot_{\mathbb{A}} b) = (a, b, b, b)\ .$$

Definition 2. *An* equivariant satisfiability problem *is a pair* $(\boldsymbol{s}, \boldsymbol{t}) \in \mathbb{T}^n$ *(written* $\boldsymbol{s} \sim? \boldsymbol{t}$*) for which it is desired to find a valuation* θ *such that* $\theta(\boldsymbol{s}) \sim \theta(\boldsymbol{t})$*. An* equivariant matching problem *is an equivariant satisfiability problem with* $\boldsymbol{t}$ *ground; then we write* $\boldsymbol{s} \lesssim? \boldsymbol{t}$*.*

Example 2. The equivariance problem

$$(X, (a\ b) \cdot X, (a\ c) \cdot X) \sim? (b', a', c')$$

has solution $[X := a]$. In fact, this is the only solution. On the other hand,

$$(a, X) \sim? (c, d)$$

has infinitely many solutions $[X := b]$ for $b \neq a$, and neither of the following have any solutions:

$$(a, b) \sim? (X, X) \qquad (X, (a\ b) \cdot X, Y, (a\ c) \cdot Y) \sim? (a, b, c, d)$$

Definition 3. *A sequence* $\boldsymbol{a} \in \mathbb{A}^n$ *is* distinct *if no atom is repeated in it. For each* n*, fix a* distinct sequence of length n *(denoted* $\mathbb{A}_n$*),* $\mathbb{A}_n = (a_1, \ldots, a_n)$ *for distinct atoms* $a_i \in \mathbb{A}$*. A* distinct matching problem *is an equivariant matching problem for which* $\boldsymbol{t} = \mathbb{A}_n$*.*

Note that $\boldsymbol{a}$ is distinct if and only if $i \mapsto a_i$ is injective. The fixed distinct sequences $\mathbb{A}_n$ are concrete representatives of the equivariance classes of distinct n-tuples, so $\boldsymbol{a} \sim \mathbb{A}_n$ precisely when $\boldsymbol{a}$ is distinct. Though distinct matching is a very restricted case of equivariant matching, it is still **NP**-complete, as we shall show in the next section.

Remark 1 (Atoms as Constants vs. Variables). Note that, following Urban, Pitts, and Gabbay, our term language treats *atoms as constants* (AAC): we use concrete atom symbols as individual terms a and in permutations π. They are not subject to replacement by valuation, only to swapping. In contrast, Pitts' nominal logic treats *atoms as variables* (AAV): in fact, theories with atom constants are inconsistent because any atom is fresh for any constant, but no atom is fresh for itself. Atom constants simplify many matters; for example, nominal unification is much easier when only atom constants can appear in swaps or abstractions. Formalizing equivariant unification is much more complex in an AAV setting as well: for example, valuations cannot be defined as ground substitutions since there are no ground atom constants in AAV. The AAC approach can be simulated within AAV, so nominal and equivariant unification for AAV must be at least as hard as for AAC. We leave the study of these problems in AAV for future work.

3 Complexity

We define the following decision problems:

$$\begin{aligned}
\text{EV} &= \{\boldsymbol{a} \sim \boldsymbol{b} \mid \boldsymbol{a}, \boldsymbol{b} \text{ ground}\} \\
\text{DMAT} &= \{\boldsymbol{s} \lesssim? \ \mathbb{A}_n \mid \exists\theta.\theta(\boldsymbol{s}) \sim \mathbb{A}_n\} \\
\text{EVMAT} &= \{\boldsymbol{s} \lesssim? \ \boldsymbol{b} \mid \boldsymbol{b} \text{ ground}, \exists\theta.\theta(\boldsymbol{s}) \sim \boldsymbol{t}\} \\
\text{EVSAT} &= \{\boldsymbol{s} \sim? \ \boldsymbol{t} \mid \exists\theta.\theta(\boldsymbol{s}) \sim \theta(\boldsymbol{t})\} \\
\text{EVSAT}^* &= \{S \mid \exists\theta.\forall(\boldsymbol{s} \sim? \ \boldsymbol{t}) \in S.\theta(\boldsymbol{s}) \sim \theta(\boldsymbol{t})\}
\end{aligned}$$

Note that DMAT $\leq$ EVMAT $\leq$ EVSAT $\leq$ EVSAT* by inclusion reductions. We now establish that EV is in **P**, and the rest of the problems are in **NP**.

For a ground sequence $\boldsymbol{a}$, let $E_{\boldsymbol{a}} = \{(i,j) \mid a_i = a_j\}$. That is, $E_{\boldsymbol{a}}$ is an equivalence relation whose equivalence classes are the indices of equal elements of a.

Proposition 2. *For ground sequences $\boldsymbol{a}, \boldsymbol{b}$ of equal length, $\boldsymbol{a} \sim \boldsymbol{b}$ if any only if $E_{\boldsymbol{a}} = E_{\boldsymbol{b}}$.*

Proof. If $\boldsymbol{a} \sim \boldsymbol{b}$, assume $\pi \cdot \boldsymbol{a} = \boldsymbol{b}$ and suppose $(i,j) \in E_{\boldsymbol{a}}$. Then $a_i = a_j$. So $b_i = \pi \cdot a_i = \pi \cdot a_j = b_j$. Hence $(i,j) \in E_{\boldsymbol{b}}$ and so $E_{\boldsymbol{a}} \subseteq E_{\boldsymbol{b}}$. A symmetric argument shows $E_{\boldsymbol{b}} \subseteq E_{\boldsymbol{a}}$, so the two sets are equal.

If $E_{\boldsymbol{a}} = E_{\boldsymbol{b}} = E$, note that the functions $f : i \mapsto a_i$ and $g : i \mapsto b_i$ are both constant on equivalence classes of E. Hence, the functions $f_E : [i]_E \mapsto a_i$ and $g_E : [i]_E \mapsto b_i$ are well-defined. Moreover, both are injective, since if $a_i = a_j$ then $[i]_E = [j]_E$ and similarly for $\boldsymbol{b}$; consequently the functions (considered on range $A(\boldsymbol{a})$ and $A(\boldsymbol{b})$ respectively) are invertible. Then the function $g \circ f_E^{-1} : A(\boldsymbol{a}) \to A(\boldsymbol{b})$ is also invertible. Any bijection between finite sets $B, C \subseteq \mathbb{A}$ can be extended to a permutation $\pi : \mathbb{A} \to \mathbb{A}$, so by choosing such an extension we have $\pi \cdot a_i = g_E \circ f_E^{-1}(a_i) = g_E([i]_E) = b_i$ for each i $(1 \leq i \leq n)$, so $\pi \cdot \boldsymbol{a} = \boldsymbol{b}$. QED.

The relations $E_{\boldsymbol{a}}, E_{\boldsymbol{b}}$ can obviously be represented as graphs which can be constructed from $\boldsymbol{a}, \boldsymbol{b}$ and compared in polynomial time.

Corollary 1. EV *is in* **P**.

Furthermore, the remaining four problems obviously have polynomial-time checkable certificates, namely minimal witnessing valuations θ.

Corollary 2. EVSAT*, EVSAT, EVMAT, *and* DMAT *are in* **NP**.

In the rest of this section we prove

Theorem 1. *The problem* DMAT *is* **NP***-complete.*

Proof. Having already shown DMAT $\in$ **NP**, we show **NP**-hardness only. We reduce from the **NP**-complete problem GRAPH 3-COLORABILITY, that is, determining whether a graph's vertices can be colored with one of three colors so that no neighboring vertices are the same color.

Let a (directed) graph $G = (V, E)$ with n vertices and m edges be given. We assume without loss of generality that $V = \{1, \ldots, n\}$ and $E = \{e_1, \ldots, e_m\}$. We write e^s, e^t for the source and target of the edge $e \in E$. Let $C = \{r, g, b\}$ be a three-element subset of $\mathbb{A}$. We define a 3-coloring as an n-tuple $\boldsymbol{c} \in C^n$ such that $c_{e^s} \neq c_{e^t}$ whenever $e \in E$.

Define $\pi_C = (r\ g)(g\ b)$, a cyclic permutation on $\mathbb{A}$ with support C. Choose (by Lemma 1) $n + m$ permutations $\tau_1, \ldots, \tau_n, \sigma_1, \ldots, \sigma_m$ so that if $T_i = \tau_i \cdot C$ for each $i \in \{1, \ldots, n\}$, and $S_j = \sigma_j \cdot C$ for each $j \in \{1, \ldots, m\}$, then the sets C, $\{T_i \mid 1 \leq i \leq n\}$, and $\{S_j \mid 1 \leq j \leq m\}$ are mutually disjoint.

Let $X_1, \ldots, X_n \in \mathbb{V}$ be n distinct variables.

Idea of the proof. We will construct an instance of DMAT such that for any solution θ, $\boldsymbol{c} = (\theta(X_1), \ldots, \theta(X_n)$ is a 3-coloring. To do this, we need to force all of the X_i to be elements of C and for each edge e force X_{e^s} and X_{e^t} to be different.

Observe $X \neq \pi_C \cdot X$ if and only if $X \in \mathit{supp}(\pi_C) = C$. So it is easy to encode a single set constraint $X \in C$ as a DMAT problem

$$(X, \pi_C \cdot X) \lesssim^? \mathbb{A}_2\ .$$

However, for two variables this does not quite work:

$$(X_1, \pi_C \cdot X_1, X_2, \pi_C \cdot X_2) \lesssim^? \mathbb{A}_4$$

forces $X_1, X_2 \in C$ but also forces $X_1 \neq X_2$, $\pi_C \cdot X_1 \neq X_2$, etc. This is too strong. To prevent interference between subproblems, we isolate them using the permutations τ_1, τ_2:

$$(\tau_1 \cdot X_1, \tau_1 \circ \pi_C \cdot X_1, \tau_2 \cdot X_2, \tau_2 \circ \pi_C \cdot X_2) \lesssim^? \mathbb{A}_4$$

First note that $\tau_1 \cdot X_1 \neq \tau_1 \circ \pi_C \cdot X_1$ implies $X_1 \neq \pi_C \cdot X_1$ so $X_1 \in C$ and similarly $X_2 \in C$, as before. On the other hand, if X_1, X_2 are in C, then all four components are different, since the first two lie in T_1 and the last two in T_2, and the two sets are disjoint. It is not hard to show by induction that

$$\boldsymbol{s} = (\tau_1 \cdot X_1, \tau_1 \circ \pi_C \cdot X_1, \ldots, \tau_n \cdot X_n, \tau_n \circ \pi_C \cdot X_n) \lesssim^? \mathbb{A}_{2n}$$

is in DMAT if and only if $X_1, \ldots, X_n \in C$.

Now we need to enforce that whenever $e \in E$, we have $X_{e^s} \neq X_{e^t}$. For a single edge, the following problem suffices:

$$(X_{e^s}, X_{e^t}) \lesssim^? \mathbb{A}_2$$

However, as was the case earlier, problems cannot always be combined correctly because they might interfere. For example, for two edges $(1, 2), (1, 3)$, the problem

$$(X_1, X_2, X_1, X_3) \lesssim^? \mathbb{A}_4$$

is unsatisfiable because the value of X_1 is repeated in any valuation, but $[X_1 := r, X_2 := g, X_3 := b]$ is a 3-coloring. To get around this problem, we use the permutations σ_i to isolate the constraints for each edge e_i. For example,

$$(\sigma_1 \cdot X_1, \sigma_1 \cdot X_2, \sigma_2 \cdot X_1, \sigma_2 \cdot X_3) \lesssim^? \mathbb{A}_4$$

ensures $X_1 \neq X_2$ and $X_1 \neq X_3$. Also, if $X_1, X_2, X_3 \in C$ then the first two components are in S_1 and the second two in S_2, and $S_1 \cap S_2 = \emptyset$. So more generally, the problem

$$\boldsymbol{t} = (\sigma_1 \cdot X_{e_1^s}, \sigma_1 \cdot X_{e_1^t}, \ldots, \sigma_m \cdot X_{e_m^s}, \sigma_m \cdot X_{e_m^t}) \lesssim^? \mathbb{A}_{2m}$$

enforces the coloring property for each edge and permits all valid colorings.

Define $\boldsymbol{u}$ to be the $2n+2m$-tuple resulting from concatenating the sequences $\boldsymbol{s}$ and $\boldsymbol{t}$. Then $\boldsymbol{u} \lesssim^? \mathbb{A}_{2n+2m}$ is the DMAT problem corresponding to the instance G of GRAPH 3-COLORABILITY.

Correctness of the reduction. So far we have only described the construction and the intuition behind it. It is easy to see that the size of $\boldsymbol{u}$ is $O(m+n)$, since π_C, τ_i, and σ_j each have representations consisting of at most three transpositions. We now show carefully that the reduction is correct, that is, G has a 3-coloring $\boldsymbol{c} \in C^n$ if and only if $\boldsymbol{u}$ has a distinct valuation θ. The backward direction is easy, since (as outlined above) it is easy to show that any solution θ making $\boldsymbol{s}$ and $\boldsymbol{t}$ distinct corresponds to a 3-coloring $c_i = \theta(X_i)$.

The difficulty is showing that $\boldsymbol{u}$ is not over-constrained: that is, if $\boldsymbol{c}$ is a 3-coloring then the valuation $\theta(X_i) = c_i$ makes $\boldsymbol{u}$ distinct. Suppose $\boldsymbol{c}$ is a 3-coloring and $\theta(X_i) = c_i$. We need to show that $i \neq j$ implies $\theta(u_i) \neq \theta(u_j)$ for each $i, j \in \{1, \ldots, |\boldsymbol{u}|\}$. Assume $i, j \in \{1, \ldots, |\boldsymbol{u}|\}$ and $i \neq j$. Suppose without loss of generality that $i < j$. There are three cases.

If i is even or $j > i+1$, then $u_i = \rho \cdot X_k$ and $u_j = \rho' \cdot X_{k'}$ for some permutations ρ, ρ' and $X_k, X_{k'}$, and $\rho \cdot C$ and $\rho' \cdot C$ are distinct, so

$$\theta(u_i) = \rho \cdot c_k \neq \rho' \cdot c_{k'} = \theta(u_j)$$

If i is odd and $i+1 = j$ and $j \leq 2n$, then j is even; set $k = j/2$. Then $u_i = \tau_k \cdot X_k, u_j = \tau_k \circ \pi_C \cdot X_k$, and we have

$$\theta(u_i) = \tau_k \cdot c_k \neq \tau_k \circ \pi_C \cdot c_k = \theta(u_j)$$

since $\pi_C \cdot c_k \neq c_k$.

If i is odd and $j = i+1$ and $2n+1 \leq i$, then j and $j-2n$ are even; set $k = (j-2n)/2$. Then $u_i = \sigma_k \cdot X_{e_k^s}$, $u_j = \sigma_k \cdot X_{e_k^t}$, and

$$\theta(u_i) = \sigma_k \cdot c_{e_k^s} \neq \sigma_k \cdot c_{e_k^t} = \theta(u_j)$$

where $c_{e_k^s} \neq c_{e_k^t}$ since c is a 3-coloring. So, in any case, $\theta(u_i) \neq \theta(u_j)$. QED.

Corollary 3. EVMAT, EVSAT *and* EVSAT* *are* **NP***-complete.*

4 A tractable special case

There are several special cases of equivariant satisfiability or matching that are tractable. We present a one such special case, a simple syntactic restriction that guarantees that equivariant satisfiability can be reduced to nominal unification. We describe some additional special cases at the end of this section.

Before defining nominal unification, we first need to extend permutation action to terms and define substitutions and renamings. Permutations act on terms as follows:

$$\pi \cdot_{\mathbb{T}} (a) = \pi \cdot_{\mathbb{A}} a \qquad \pi \cdot_{\mathbb{T}} (\pi' \cdot X) = (\pi \circ \pi') \cdot X$$

and act componentwise on sequences of terms. A substitution is a function $\sigma : \mathbb{V} \to \mathbb{T}$ from variables to terms, extended as follows to $\sigma_{\mathbb{T}} : \mathbb{T} \to \mathbb{T}$:

$$\sigma_{\mathbb{T}}(a) = a \qquad \sigma_{\mathbb{T}}(\pi \cdot X) = \pi \cdot_{\mathbb{T}} \sigma(X)$$

and extended componentwise to $\sigma_{\mathbb{T}^n} : \mathbb{T}^n \to \mathbb{T}^n$. Note that substitutions may activate delayed permutation actions:

$$((a\ b) \cdot X, (a\ c) \cdot Y)[X := a, Y := (b\ c) \cdot Z] = (b, (a\ c)(b\ c) \cdot Z)\ .$$

Moreover, note that $\pi \cdot (\sigma(x)) = \sigma(\pi \cdot x)$, for x a term or sequence.

A term s (or sequence $\boldsymbol{s}$) is a *renaming* of another term t (sequence $\boldsymbol{t}$) if $s = \rho(t)$ (or $\boldsymbol{s} = \rho(\boldsymbol{t})$) for some *invertible* substitution ρ. Note that invertible substitutions may involve swapping: for example, $[X := \pi \cdot Y, Y := X]$ has inverse $[X := Y, Y := \pi^{-1} \cdot X]$. Two terms s, t (or sequences $\boldsymbol{s}, \boldsymbol{t}$) *unify* if there is an idempotent substitution σ such that $\sigma(s) = \sigma(t)$ (or $\sigma(\boldsymbol{s}) = \sigma(\boldsymbol{t})$). For example, $(a\ b) \cdot X$ unifies with $(b\ c) \cdot X$ with substitution $[X := d]$, for any $d \notin \{a, b, c\}$. The algorithm of Urban et al. decides a more general case of nominal unification, and finds unique MGUs (up to renaming) when they exist. Although their algorithm is not polynomial time as presented, a polynomial-time algorithm can be obtained by modifying the quadratic unification algorithm of Martelli and Montanari [10]; further improvements may be possible.

We say $\boldsymbol{s}$ is *pure* if no atoms appear in $\boldsymbol{s}$: that is, $\boldsymbol{s}$ is a list of variables with suspended permutation id. We say $\boldsymbol{s}$ is *semi-pure* if it is a renaming of a pure $\boldsymbol{s}'$. For example, (X, Y, X) is pure and $((a\ b) \cdot X, Y, (c\ a)(c\ b)(c\ a) \cdot X)$ is semi-pure. We say $\boldsymbol{s}, \boldsymbol{t}$ are *variable-disjoint* when $V(\boldsymbol{s}) \cap V(\boldsymbol{t}) = \emptyset$.

Theorem 2. *If $\boldsymbol{s}$ is semi-pure and $\boldsymbol{s}, \boldsymbol{t}$ are variable-disjoint, then $\boldsymbol{s} \sim ?\ \boldsymbol{t}$ can be decided in polynomial time.*

Proof. We show this in two steps. First, assuming $\boldsymbol{s}$ is pure, we show that deciding $\boldsymbol{s} \sim ?\ \boldsymbol{t}$ reduces to nominal unification. Second, we show that if $\boldsymbol{s}$ is semi-pure and $\boldsymbol{s}'$ is a pure renaming of $\boldsymbol{s}$, then $\boldsymbol{s} \sim ?\ \boldsymbol{t}$ is satisfiable if and only if $\boldsymbol{s}' \sim ?\ \boldsymbol{t}$ is.

For the first part, if $\boldsymbol{s}$ and $\boldsymbol{t}$ have a nominal unifier, note that any unifier has a ground instance, any instance of a unifier is also a unifier, and any ground substitution is a valuation. So we can find a valuation θ such that $\theta(\boldsymbol{s}) = \theta(\boldsymbol{t})$; hence, $\mathsf{id} \cdot \theta(\boldsymbol{s}) = \theta(\boldsymbol{t})$ so $\theta(\boldsymbol{s}) \sim \theta(\boldsymbol{t})$. Conversely, suppose that $\pi \cdot \theta(\boldsymbol{s}) = \theta(\boldsymbol{t})$. Let θ' be defined as follows:

$$\theta'(X) = \begin{cases} \pi \cdot \theta(X) : X \in V(\boldsymbol{s}) \\ \theta(X) \quad\ : \text{otherwise} \end{cases}$$

Since $\boldsymbol{s}, \boldsymbol{t}$ are variable-disjoint, θ' agrees with θ on $V(\boldsymbol{t})$ so $\theta(\boldsymbol{t}) = \theta'(\boldsymbol{t})$. Also, since $\boldsymbol{s}$ is pure, we know $\boldsymbol{s} = (X_1, \ldots, X_n)$ for $\{X_1, \ldots, X_n\} = V(\boldsymbol{s})$ (where some

of the X_i may be repeated). Hence

$$\begin{aligned}\theta'(\boldsymbol{s}) &= (\theta'(X_1), \ldots, \theta'(X_n)) = (\pi \cdot \theta(X_1), \ldots, \pi \cdot \theta(X_n)) \\ &= \pi \cdot \theta(\boldsymbol{s}) = \theta(\boldsymbol{t}) = \theta'(\boldsymbol{t})\end{aligned}$$

So $\theta'(\boldsymbol{s}) = \theta'(\boldsymbol{t})$ and θ' is a nominal unifier of $\boldsymbol{s}, \boldsymbol{t}$. The existence of a nominal unifier can be decided in polynomial time by nominal unification.

For the second part, note that since $\boldsymbol{s}$ is semi-pure, there exists a pure $\boldsymbol{s}'$ and invertible ρ such that $\rho(\boldsymbol{s}) = \boldsymbol{s}'$. Since $\boldsymbol{s}, \boldsymbol{t}$ are variable-disjoint, we may choose $\boldsymbol{s}', \rho$ such that $\boldsymbol{s}', \boldsymbol{t}$ are also variable-disjoint and ρ fixes all the variables $V(\boldsymbol{t})$ of $\boldsymbol{t}$. Since $\rho(X) = X$ whenever $X \in V(\boldsymbol{t})$, we also have $\rho(\boldsymbol{t}) = \boldsymbol{t}$. We will show that $\boldsymbol{s}' \sim^? \boldsymbol{t}$ is satisfiable if and only if $\boldsymbol{s} \sim^? \boldsymbol{t}$ is; since the former can be decided efficiently, so can the latter. Assume $\boldsymbol{s}' \sim^? \boldsymbol{t}$ is satisfiable, and suppose $\pi \cdot \theta(\boldsymbol{s}') = \theta(\boldsymbol{t})$. Let $\theta' = \theta \circ \rho$. Then

$$\pi \cdot \theta'(\boldsymbol{s}) = \pi \cdot \theta \circ \rho(\boldsymbol{s}) = \pi \cdot \theta(\boldsymbol{s}') = \theta(\boldsymbol{t}) = \theta \circ \rho(\boldsymbol{t}) = \theta'(\boldsymbol{t})$$

so $\boldsymbol{s} \sim^? \boldsymbol{t}$ has a solution. A symmetric argument (using the equation $\rho^{-1}(\boldsymbol{s}') = \boldsymbol{s}$) shows that if $\boldsymbol{s} \sim^? \boldsymbol{t}$ has a solution then so does $\boldsymbol{s}' \sim^? \boldsymbol{t}$. QED.

Remark 2. Theorem 2 can be generalized to unification over full nominal terms, in which case pure terms are simply first-order terms with no atoms, abstractions, or swaps. Suppose we have a purely first-order logic program P (i.e., a set of first-order Horn clauses). Since the variables of program clauses are always freshened prior to attempting resolution, resolution behaves the same using equivariant unification as nominal unification, so for atomic A, $P \vdash A$ can be derived using equivariant unification if any only if $P \vdash A$ can also be derived using nominal unification. Similarly, suppose we have a purely first-order term rewriting system R. Then $s \to_R t$ using equivariant matching if and only if $s \to_R t$ using nominal matching. These results can be generalized to permit program clauses with semi-pure heads and unrestricted bodies, and rewriting rules with semi-pure left-hand sides and arbitrary right-hand sides. So broad classes of nominal logic programs and rewrite systems (including all first-order logic programs and rewrite systems) can be executed efficiently without sacrificing completeness.

Remark 3. There are other tractable special cases, but they depend on aspects of nominal logic beyond the scope of this paper. First, equivariant matching is tractable when both terms are free of swaps but may contain abstractions and atoms. The algorithm for this case is a straightforward generalization of nominal matching, based on the insight that $\langle a \rangle X \lesssim^? \langle b \rangle b$ precisely when $X = a$ and $\langle a \rangle X \lesssim^? \langle b \rangle c$ (where $b \neq c$) precisely when $a \# X$. Another well-behaved case is when the two terms are variable-disjoint and one term has *empty support*; this means that no atom a occurs outside the scope of an abstraction of a in s, and that every atom a appearing in s is fresh for every variable not enclosed by an abstraction of a in s. For example, $(Y, \langle b \rangle X)$ has empty support assuming $b \# Y$, whereas $(a, \langle b \rangle X)$ does not since a appears unbound. In this case, equivariant unification reduces to nominal unification. Note that this is a generalization of Theorem 2 since pure terms have empty support. Many interesting nominal logic programs can be expressed using head clauses with empty syntactic support.

5 Related and future work

Permutations of variables arise in natural ways in first-order and higher-order pattern unification. In first-order unification, any two MGUs for a given problem are equivalent up to permuting their free variables. In higher-order unification, the cases that cause problems involve free variables applied to arbitrary lists of bound and free variables, and this case is avoided by the higher-order pattern restriction that free variables are only applied to lists of *distinct* bound variables [11]. Consequently, whenever two subterms $X\ x_1 \cdots\ x_n$, $Y\ y_1 \cdots\ y_m$ are to be unified (where X, Y are free and $\bar{x}, \bar{y}$ are bound variables), there is always a partial permutation relating the variables $\bar{x}$ and $\bar{y}$. Then all the nondeterministic choices in Huet's full higher-order semi-unification algorithm can be avoided; unification can be performed efficiently, and MGUs are unique when they exist.

An alternative view of equivariant satisfiability to the one taken in this paper is as the search for a solution for the equation $P \cdot \boldsymbol{s}(\bar{X}) = \boldsymbol{t}(\bar{X})$ (over a permutation variable P and atom variables $\bar{X}$). In light of this fact, prior work on satisfiability for equations over groups may be relevant to equivariant unification. Many mathematicians from Frobenius onward have studied the problem of solving (and counting the solutions to) specific group equations such as $P^n = \mathsf{id}$ [5]. Albert and Lawrence studied elementary unification in varieties of nilpotent groups [1]. They showed that MGUs may not exist in that setting, but are unique when they do, and described a polynomial time algorithm that computes a MGU or determines that none exists for a specific problem. Goldmann and Russell [7] showed that for finite groups, solving systems of equations (possibly involving constants) is polynomial time if the group is Abelian, otherwise **NP**-complete. They also showed that solving a single group equation is **NP**-complete if the group is non-solvable and in **P** if it is nilpotent; the complexity of solvable but non-nilpotent groups is not settled. Engebretsen et al. [4] showed that approximating the number of solutions to a single group equation to within $|G| - \epsilon$ is **NP**-hard for any $\epsilon > 0$.

Our first proof of **NP**-completeness for equivariant satisfiability reduced from Goldmann and Russell's single-equation group satisfiability problem for non-solvable groups (since full finite symmetric groups are not solvable). That approach required several intermediate reductions and showed only to the weaker result that EVSAT is **NP**-complete, leaving the complexity of equivariant matching unresolved. Except for Goldmann and Russell's work, we have not found any of the above research on unification and satisfaction for group equations to be applicable to equivariant unification.

There are two immediate directions for future work. First, we are developing practical algorithms for equivariant matching and unification for use in resolution and term rewriting in αProlog, a logic programming language based on nominal logic [3]. Second, in this paper we asserted without proof that equivariant unification is necessary and sufficient for complete nominal resolution. Though this seems clear, it requires proof. We plan to present practical equivariant unification and matching algorithms and prove that equivariant unification is sound and complete for nominal resolution in future work.

6 Conclusions

Equivariant satisfiability and matching, or deciding whether two terms involving swapping can be made equal "up to a permutation", are important decision problems for automated reasoning, logic programming, and term rewriting in nominal logic. We have shown that both are **NP**-complete. We have also found an interesting tractable special case, for which nominal unification suffices. Consequently, first-order logic programs and term rewriting systems can be run efficiently on nominal terms. Only those programs or rewrite systems that actually use the novel features of nominal logic need pay for them.

References

1. Michael H. Albert and John Lawrence. Unification in varieties of groups: nilpotent varieties. *Canadian Journal of Mathematics*, 46(6):1135–1149, 1994.
2. Peter J. Cameron. *Permutation groups*, volume 45 of *London Mathematical Society Student Texts*. Cambridge University Press, 1999.
3. J. Cheney and C. Urban. System description: Alpha-Prolog, a fresh approach to logic programming modulo alpha-equivalence. In J. Levy, M. Kohlhase, J. Niehren, and M. Villaret, editors, *Proc. 17th Int. Workshop on Unification, UNIF'03*, pages 15–19, Valencia, Spain, June 2003. Departamento de Sistemas Informaticos y Computacion, Universidad Politecnica de Valencia. Technical Report DSIC-II/12/03.
4. Lars Engebretsen, Jonas Holmerin, and Alexander Russell. Inapproximability results for equations over finite groups. *Theoretical Computer Science*, 312(1):17–45, 2004.
5. H. Finkelstein. Solving equations in groups: a survey of Frobenius' Theorem. *Periodica Mathematica Hungarica*, 9(3):187–204, 1978.
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. Mikael Goldmann and Alexander Russell. The complexity of solving equations over finite groups. *Information and Computation*, 178:253–262, 2002.
8. Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *Proc. 14th Symp. on Logic in Computer Science*, pages 204–213. IEEE, July 1999.
9. Gerard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–67, 1975.
10. A. Martelli and U. Montanari. An efficient unification algorithm. *Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.
11. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Logic and Computation*, 1(4):497–536, 1991.
12. Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI '89)*, pages 199–208. ACM Press, 1989.
13. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 183:165–193, 2003.
14. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
15. C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. In M. Baaz, editor, *Computer Science Logic and 8th Kurt Gödel Colloquium (CSL'03 & KGC)*, volume 2803 of *Lecture Notes in Computer Science*, pages 513–527, Vienna, Austria, 2003. Springer-Verlag.