CONSTRAINT SOLVING IN NON-PERMUTATIVE NOMINAL ABSTRACT SYNTAX

MATTHEW R. LAKIN

University of Cambridge Computer Laboratory, Cambridge CB3 0FD, UK, and Department of
Computer Science, University of New Mexico, Albuquerque, NM 87131
e-mail address: Matthew.Lakin@cl.cam.ac.uk

ABSTRACT. Nominal abstract syntax is a popular first-order technique for encoding, and reasoning about, abstract syntax involving binders. Many of its applications involve *constraint solving*. The most commonly used constraint solving algorithm over nominal abstract syntax is the Urban-Pitts-Gabbay *nominal unification* algorithm, which is well-behaved, has a well-developed theory and is applicable in many cases. However, certain problems require a constraint solver which respects the equivariance property of nominal logic, such as Cheney's *equivariant unification* algorithm. This is more powerful but is more complicated and computationally hard. In this paper we present a novel algorithm for solving constraints over a simple variant of nominal abstract syntax which we call *non-permutative*. This constraint problem has similar complexity to equivariant unification but without many of the additional complications of the equivariant unification term language. We prove our algorithm correct, paying particular attention to issues of termination, and present an explicit translation of name-name equivariant unification problems into non-permutative constraints.

1. INTRODUCTION

Constraint solving over the abstract syntax of programming languages is vital in many areas of logic and computer science. For example, many compiler optimisations are typically phrased as constraint problems. The abstract syntax in question often involves binding constructs, such as λ-expressions or $\forall$-quantifiers. In these cases we would want the abstract syntax encoding to respect α-equivalence of binding structures—this is known as the Barendregt variable convention [Bar84].

One approach to representing and manipulating abstract syntax with binders is *nominal abstract syntax* [GP02], which was developed as a first-order theory of abstract syntax involving bound names. The theory is based on permuting, rather than substituting, names, as this has more convenient logical properties. Bindable names in the object-language are represented by meta-level names ranged over by n (these are often called *atoms* in the literature). These names follow Gabbay's *permutative convention* [GM08] which states that

distinct meta-variables n_1 and n_2 over names always denote distinct names. The object-level binding of a name n in a term t is is represented by the abstraction term $<\!n\!>\!t$. This is not itself a binder, so $<\!n_1\!>\!n_1$ and $<\!n_2\!>\!n_2$ are considered to be distinct terms if $n_1 \neq n_2$ (though they should behave similarly as they are α-equivalent).

Logic programming over nominal abstract syntax requires a unification algorithm which can unify nominal terms modulo α-equivalence. One such algorithm is *nominal unification* [UPG04], which is a simple, well-studied constraint solving algorithm for nominal abstract syntax. The algorithm extends first-order unification to work on nominal terms modulo α-equivalence by adding *freshness* constraints which are satisfied if a given name does not appear free in a term. This allows terms involving binders to be equated by checking that they have the same topology of name bindings. Nominal unification enjoys unique, most general solutions [UPG04, Theorem 3.7] and is known to be decidable in quadratic time [LV08, LV10, CF11].

Resolution using nominal unification is the basis of the αProlog nominal logic programming language [CU04]. However, here we encounter a problem—resolution using nominal unification is incomplete for nominal logic [Pit03]. The issue is that nominal unification does not respect the *equivariance* property of nominal logic, that is, closure under name-permutations. A standard example, taken from [Che10], concerns capture-avoiding substitution over λ-terms encoded in nominal abstract syntax. Writing $\mathtt{subst}(M, M', n)$ to represent the capture-avoiding substitution function $M[M'/n]$, the following two rewrite rules implement the case when M is a variable.

$$\mathtt{subst}(\mathtt{var}(n), M', n) \to M' \qquad\qquad \mathtt{subst}(\mathtt{var}(n'), M', n) \to \mathtt{var}(n')$$

From the first rule we infer that $\mathtt{subst}(\mathtt{var}(n), \mathtt{var}(n'), n) \to \mathtt{var}(n')$. However, nominal unification cannot compute a substitution for M' such that

$$\mathtt{subst}(\mathtt{var}(n), M', n) =_\alpha \mathtt{subst}(\mathtt{var}(n'), \mathtt{var}(n), n').$$

In nominal logic this equation holds modulo a permutation, and hence nominal unification does not suffice for complete proof search in all cases.

A workaround is to define a well-formedness condition on αProlog programs to isolate those which can be executed correctly using nominal unification [UC05]. A more general solution is to use a more powerful constraint solving algorithm which takes equivariance into account, such as Cheney's *equivariant unification* algorithm [Che10]. Equivariant unification generalises the term language of nominal unification to include name variables A which stand for unknown names (and which do not follow the permutative convention) and permutation variables Q which stand for unknown permutations. The syntax of names and permutations in equivariant unification is non-trivial—for example, the terms involved in a swapping may themselves contain nested swappings! Furthermore, these compound name expressions may appear in abstraction position, so we can write terms such as

$$<\!(((Q \circ Q')\, A)\, (Q^{-1}\, A))\, n\!>\!(Q'\, n)$$

even though the meaning of such a term is by no means obvious. The equivariant unification algorithm uses "permutation graphs" to solve generalised equality and freshness constraints. This constraint problem is NP-hard [Che10], and there are no longer unique, most general solutions. However, the main issue with equivariant unification are that the term language is complicated (see below), making it difficult to implement the algorithm and interpret the resulting answers.

1.1. **Contributions.** In this paper we present an alternative to equivariant unification, in the form of a constraint algorithm over *non-permutative nominal abstract syntax* (NPNAS). This is a mild generalisation of standard nominal abstract syntax and is a syntactic subset of the equivariant unification term language. We use the term *non-permutative* because no meta-variables, not even those representing names in binding position, follow the permutative convention. The contributions of this paper are as follows:

- the presentation of non-permutative nominal abstract syntax as a simple yet powerful extension of existing nominal techniques;
- a novel decision procedure for solving equality and freshness constraints over non-permutative nominal terms, and a proof of its correctness; and
- a reduction of name-name equivariant unification problems to non-permutative nominal constraints.

We do not address the fact that equivariant unification is NP-hard, since the NPNAS constraint problem is also NP-hard. However, the simplicity of the NPNAS term language and algorithm offer practical advantages over equivariant unification when it comes to implementing a nominal logical programming or rewriting system. The constraint solving algorithm described in this paper can be used to implement sound and complete resolution over inductive definitions involving binders, as described in [Lak10]. This is an important result given the ubiquity of binders in logic and computer science.

The rest of this paper is organised as follows. Section 2 presents important background on nominal abstract syntax, nominal unification and equivariant unification. In Section 3 we present the syntax of NPNAS terms and constraints and define their semantics. We present a constraint transformation algorithm in Section 4 and use it to derive a correct decision procedure in Section 5, paying particular attention to termination. We address the relationship between NPNAS and equivariant unification in Section 6 by defining an explicit translation of name-name equivariant unification problems into NPNAS. We discuss related and future work in Section 7 and conclude in Section 8.

2. Background

This section presents the basics of nominal abstract syntax, nominal terms, nominal unification and equivariant unification. This suffices to demonstrate the relationship of the non-permutative terms and constraints studied in this paper to existing work. We refer the reader to [Pit03] for a full introduction to nominal logic, to [UPG04] for details on nominal unification and [Che10] for an in-depth treatment of equivariant unification.

2.1. **Nominal abstract syntax.** As is standard in the world of nominal techniques, we use *nominal signatures* [UPG04] to specify binding structures in the object-language.

Definition 2.1 (Nominal signatures)**.** A nominal signature Σ consists of:

- a finite set $\mathbb{N}_\Sigma$ of *name sorts*, ranged over by N;
- a finite set $\mathbb{D}_\Sigma$ of *data sorts*, disjoint from $\mathbb{N}_\Sigma$ and ranged over by D; and

- a finite set $\mathbb{C}_\Sigma$ of *constructors* $K : T \to D$, where the argument type $T \in Ty_\Sigma$ is generated by the following grammar.

$$
\begin{array}{rcll}
T \in Ty_\Sigma & ::= & D & \text{(data sorts)} \\
& | & N & \text{(name sorts)} \\
& | & [N]T & \text{(name abstractions)} \\
& | & T * \cdots * T & \text{(tuples)} \\
& | & \texttt{unit} & \text{(unit)}
\end{array}
$$

For simplicity we will assume that Σ is such that every type $T \in Ty_\Sigma$ is *inhabited* by some ground tree (defined below). This property of nominal signatures can be checked straightforwardly—see [Lak10, Section 3.3.2] for details.

In standard approaches to nominal abstract syntax [GP02], bindable names are represented explicitly in the syntax of ground syntax trees. We fix a countably infinite set *Name* of names to stand for object-language names which may be bound. The meta-variable n ranges *permutatively* over these. We assume the existence of a total function *sort* which maps every name n to a name sort $N \in \mathbb{N}_\Sigma$ such that there are infinitely many names assigned to every name sort. We say that $n \in Name(N)$ if $sort(n) = N$.

Definition 2.2 (Ground trees). We write $Tree_\Sigma$ for the set of all syntax trees over the nominal signature Σ. We refer to these as *ground trees* following the terminology of [Lak10], though they are often referred to as (ground) nominal terms. With names (and unit) as our building blocks, we define classes $g \in Tree_\Sigma(T)$ of syntax trees of the various types by constructor application, tupling and name abstraction, as follows.

$$
\frac{sort(n) = N}{n \in Tree_\Sigma(N)} \qquad \overline{() \in Tree_\Sigma(\texttt{unit})} \qquad \frac{g_1 \in Tree_\Sigma(T_1) \quad \cdots \quad g_k \in Tree_\Sigma(T_k)}{(g_1, \ldots, g_k) \in Tree_\Sigma(T_1 * \cdots * T_k)}
$$

$$
\frac{g \in Tree_\Sigma(T) \qquad (K : T \to D) \in \Sigma}{K\, g \in Tree_\Sigma(D)} \qquad \frac{sort(n) = N \qquad g \in Tree_\Sigma(T)}{\texttt{<}n\texttt{>}g \in Tree_\Sigma([N]T)}
$$

The abstraction $\texttt{<}n\texttt{>}g$ represents a term with a bound name. This term-former is not regarded as a binder, which means that, for distinct names n and n', we regard $\texttt{<}n\texttt{>}n$ and $\texttt{<}n'\texttt{>}n'$ as distinct ground trees. This is a consequence of following Gabbay's *permutative convention* [GM08].

Definition 2.3 (Permutations and permutation actions). Let *Perm* be the set of all finite permutations over *Name*, that is, the set of all bijections π such that $\pi(n) = n$ for all but finitely many n. Any element of *Perm* can be represented as a finite list of name-swappings of the form $(n\, n')$. The *action* of a permutation π on a ground tree g is to rename all names appearing in g (including those in abstraction position) according to π. This is defined as follows.

$$
\pi \cdot n \triangleq \pi(n) \qquad \pi \cdot () \triangleq () \qquad \pi \cdot (g_1, \ldots, g_k) \triangleq (\pi \cdot g_1, \ldots, \pi \cdot g_k)
$$

$$
\pi \cdot (K\, g) \triangleq K\, (\pi \cdot g) \qquad \pi \cdot (\texttt{<}n\texttt{>}g) \triangleq \texttt{<}\pi(n)\texttt{>}(\pi \cdot g)
$$

Using the definition of permutation action, Figure 1 defines a type-directed equality relation between two ground trees and a freshness relation between a name and a ground tree. The equality relation corresponds $(g =_\alpha g' : T)$ to α-equivalence [GP02]. This definition of α-equivalence paraphrases that of [Bar84], as shown by Gabbay and Pitts [GP02,

$$\frac{sort(n) = N}{n =_\alpha n : N} \qquad \overline{() =_\alpha () : \texttt{unit}} \qquad \frac{g_1 =_\alpha g_1' : T_1 \qquad \cdots \qquad g_k =_\alpha g_k' : T_k}{(g_1, \ldots, g_k) =_\alpha (g_1', \ldots, g_k') : T_1 * \cdots * T_k}$$

$$\frac{g =_\alpha g' : T \qquad (K : T \to D) \in \Sigma}{K\, g =_\alpha K\, g' : D} \qquad \frac{g =_\alpha g' : T \qquad sort(n) = N}{\texttt{<n>}g =_\alpha \texttt{<n>}g' : \texttt{<N>}T}$$

$$\frac{n \ne n' \qquad n \mathbin{\#} g' \qquad g =_\alpha (n\,n') \cdot g' : T}{\texttt{<n>}g =_\alpha \texttt{<n'>}g' : [N]T} \qquad \frac{n \ne n'}{n \mathbin{\#} n'} \qquad \overline{n \mathbin{\#} ()}$$

$$\frac{n \mathbin{\#} g_1 \qquad \cdots \qquad n \mathbin{\#} g_k}{n \mathbin{\#} (g_1, \ldots, g_k)} \qquad \frac{n \mathbin{\#} g}{n \mathbin{\#} K\, g} \qquad \overline{n \mathbin{\#} \texttt{<n>}g} \qquad \frac{n \ne n' \qquad n \mathbin{\#} g}{n \mathbin{\#} \texttt{<n'>}g}$$

Figure 1: Equality and freshness for ground trees

Proposition 2.2]. The freshness relation $(n \mathbin{\#} g)$ holds when the name n is not free in the ground tree g. We write $FN(g)$ for the set of names n which appear in g and are such that $n \mathbin{\#} g$ holds.

Finally, a relation $R \subseteq Tree_\Sigma(T_1) * \cdots * Tree_\Sigma(T_k)$ is *equivariant* if, for all permutations π, $R(g_1, \ldots, g_k)$ holds iff $R(\pi \cdot g_1, \ldots, \pi \cdot g_k)$ holds. It is not hard to show that the equality and freshness relations from Figure 1 are both equivariant.

2.2. **Nominal unification.** *Nominal unification* [UPG04] is a simple, well-studied constraint solving algorithm which extends first-order unification to work on ground trees with binders modulo α-conversion. The language of ground trees from Definition 2.2 is extended to include metavariables with suspended permutations πX. When X is instantiated by a substitution σ the permutation must be applied to produce the result $\pi \cdot (\sigma(X))$. Problems consist of equality $(t = t')$ and freshness $(n \mathbin{\texttt{\#}} t')$ constraints. These are solved in the context of a *freshness environment* ∇ of freshness assumptions $n \mathbin{\texttt{\#}} X$ between a name and a meta-variable which constrain the free names in an unknown term: if $(n \mathbin{\texttt{\#}} X) \in \nabla$ then X cannot be replaced by any term which has a free occurrence of n.

2.3. **Equivariant unification.** As mentioned above, *equivariant unification* [Che10] considerably extends the term language of nominal unification, with complex permutation expressions involving unknown names and unknown permutations. For the purposes of this paper it suffices to consider equivariant unification problems involving only terms of a fixed name sort N, ranged over by a. These are generated by the grammar below. We write n where [Che10] uses a, for consistency with the rest of this paper.

Vertices	v, w	::=	n	(name)
			A	(name variable)
Name-terms	a, b	::=	$\Pi \cdot v$	(suspended permutation)
Permutation-terms	Π	::=	ι	(identity)
			$(a\,b)$	(swap)
			Q	(permutation variable)

"Vertices" is a term used in [Che10], where equivariant unification problems are represented as "permutation graphs". A compound permutation expression Π may be an unknown permutation Q, a swapping $(a\,a')$ or an explicit permutation composition or inversion. Equivariant unification name-terms may be either concrete names n or name-variables A. We abbreviate $\iota \cdot v$ as just v in most cases. We write $\overline{n}; \overline{A}; \overline{Q} \vdash a\ ok$ to mean that $\overline{n} \supseteq names(a)$, $\overline{A} \supseteq namevars(a)$ and $\overline{Q} \supseteq pvars(a)$, and extend this definition to other elements of equivariant unification syntax in the obvious way.

The semantics of equivariant unification problems was defined in [Che10] and we briefly summarise the relevant details here. We concern ourselves with ground valuations θ applied to name-terms, in particular, the portion of the valuation that provides values for name variables A and permutation variables Q, in terms of ground names n.

- If $A \in dom(\theta)$ then $\theta(A) = n$, for some $n \in Name$.
- If $Q \in dom(\theta)$ then $\theta(Q)$ is a *ground permutation*, which can be represented as a finite (possibly empty) list of name-swappings $(n\,n')$.

The semantics of name-name equivariant unification problems is as follows, after [Che10].

- $\theta \models a \approx b$ iff $\theta(a) =_\alpha \theta(b) : N$, using the rules for α-equivalence from Figure 1.
- $\theta \models a \# b$ iff $\theta(a) \not\# \theta(b)$, using the rules for freshness from Figure 1.
- If S is a finite set of equivariant unification constraints c (referred to as a *problem*) then $Sat(S) = \{\theta \mid \forall c \in S.\ \theta \models c\}$.

Name-name equivariant unification problems are known to be NP-complete, whereas full equivariant unification (at an arbitrary type T) is known to be NP-hard, but not necessarily NP-complete [Che10].

The additional constructs supported by equivariant unification give it the power to solve equations modulo a permutation. One can compute whether there exists a permutation π such that $(\pi \cdot t) =_\alpha t'$ holds by choosing a fresh permutation variable Q and solving the equivariant unification problem $\{(Q \cdot t) \approx t'\}$. This suffices to allow complete matching and proof-search in nominal logic programming.

3. Syntax and semantics of non-permutative constraints

In this section we present the syntax of non-permutative nominal terms and constraints over these. We also define a semantics for non-permutative nominal constraints.

Schematic terms are used in informal mathematics as templates which may be used to produce a (potentially infinite) set of ground instances, quotiented by α-equivalence. To permit this, they contain variables which are instantiated with (α-equivalence classes of) ground terms according to certain rules. We fix a countably infinite set *Var* of variables as placeholders for unknown α-equivalence classes. We will use various symbols, typically x, y, etc., to range *non-permutatively* over these.

Definition 3.1 (Non-permutative nominal terms and atomic constraints)**.** The sets $Term_\Sigma$ of (schematic) non-permutative nominal terms t and $Constr_\Sigma$ of atomic constraints over the

$$\dfrac{x \in dom(\Delta) \qquad \Delta(x) = T}{\Delta \vdash_{\Sigma} x : T} \qquad \dfrac{\Delta \vdash_{\Sigma} t : T \qquad (K : T \to D) \in \Sigma}{\Delta \vdash_{\Sigma} K\,t : D} \qquad \dfrac{}{\Delta \vdash_{\Sigma} \mathtt{()} : \mathtt{unit}}$$

$$\dfrac{\Delta \vdash_{\Sigma} t_1 : T_1 \qquad \cdots \qquad \Delta \vdash_{\Sigma} t_k : T_k}{\Delta \vdash_{\Sigma} (t_1, \ldots, t_k) : T_1 * \cdots * T_k} \qquad \dfrac{\Delta \vdash_{\Sigma} x : N \qquad \Delta \vdash_{\Sigma} t : T}{\Delta \vdash_{\Sigma} \mathtt{<}x\mathtt{>}t : [N]T}$$

$$\dfrac{\Delta \vdash_{\Sigma} t : T \qquad \Delta \vdash_{\Sigma} t' : T}{\Delta \vdash_{\Sigma} t = t' \ ok} \qquad \dfrac{\Delta \vdash_{\Sigma} x : N \qquad \Delta \vdash_{\Sigma} t : T}{\Delta \vdash_{\Sigma} x \,\#\, t \ ok}$$

Figure 2: Typing rules for non-permutative nominal terms and atomic constraints

nominal signature Σ are defined as follows.

$$
\begin{array}{rclll}
t \in \mathit{Term}_{\Sigma} & ::= & x & & \text{(variable)} \\
 & | & \mathtt{<}x\mathtt{>}t & & \text{(abstraction)} \\
 & | & K\,t & & \text{(data)} \\
 & | & (t, \ldots, t) & & \text{(tuple)} \\
 & | & \mathtt{()} & & \text{(unit)} \\
c \in \mathit{Constr}_{\Sigma} & ::= & t = t & & \text{(equality constraint)} \\
 & | & x \,\#\, t & & \text{(freshness constraint)}
\end{array}
$$

Our main departure from traditional approaches to nominal abstract syntax is that there are no permutative names in the syntax—all object-level names are represented by non-permutative variables at the meta-level, even those which appear in binding position. Since the variables are non-permutative, distinct variables may be instantiated with the same ground tree, which we call *aliasing*. As we shall see, the fact that bound names may be aliased means that a schematic term can be instantiated to multiple different α-equivalence classes in general. The abstraction term-former is *not* a binder, so there are no meta-level binding constructs in schematic terms. Hence it is meaningless to define α-equivalence on schematic terms directly.

Remark 3.2 (Omission of name-constants)**.** Unlike the equivariant unification term language, we have not included explicit permutative name-constants in the grammar of non-permutative nominal terms from Definition 3.1. We omit them in part because they do not add expressive power—a combination of non-permutative variables and name inequality (freshness) constraints can be used to imitate permutative name-constants in constraints, as mentioned in Remark 6.12. Furthermore, permutative name-constants are not needed to achieve a sound and complete encoding of inductive definitions over terms involving binders, as shown in [LP09, Lak10].

We let Δ range over typing environments, which are finite partial functions from *Var* to Ty_{Σ} which assign types to variables. We write $dom(\Delta)$ for the domain of definition of Δ. Figure 2 provides rules which define typing judgements of the form $\Delta \vdash_{\Sigma} t : T$ for terms and $\Delta \vdash_{\Sigma} c \ ok$ for atomic constraints. Note that if $\Delta \vdash_{\Sigma} t : N$ holds then t must be a variable x such that $\Delta(x) = N$ for some name sort $N \in \mathbb{N}_{\Sigma}$.

Definition 3.3 (Non-permutative nominal constraint problems). Let $\overline{c}$ range over finite conjunctions $c_1 \; \& \; \cdots \; \& \; c_k$ of atomic constraints, where each $c_i \in Constr_\Sigma$. Then, a non-permutative nominal constraint problem in $Prob_\Sigma$ has the form $\exists \Delta(\overline{c})$, and is well-formed (written $\emptyset \vdash_\Sigma \exists \Delta(\overline{c}) \; ok$) iff $\Delta \vdash_\Sigma c \; ok$ holds for all $c \in \overline{c}$.

Having specified the syntax of non-permutative terms and constraint problems, we now turn to their semantics. We will give a semantics to atomic constraints and constraint problems in terms of instantiations of their variables with α-equivalence classes of ground abstract syntax trees. To do so, we must first define a notion of ground trees quotiented by α-equivalence.

Definition 3.4 (α-trees). Let $\alpha\text{-}Tree_\Sigma(T)$ be the set of all $=_\alpha$-equivalence classes of ground trees of type T, which we call α-*trees*. We let $\mathfrak{a}$ range over α-trees. If $g \in Tree_\Sigma(T)$ then we write $[g]_\alpha$ for the set $\{g' \mid g =_\alpha g' : T\}$ of all ground trees which are α-equivalent to g. If $g \in Tree_\Sigma(T)$ then $[g]_\alpha \in \alpha\text{-}Tree_\Sigma(T)$. In the special case where $g \in \alpha\text{-}Tree_\Sigma(N)$ it follows that $g = [n]_\alpha = \{n\}$, for some $n \in Name(N)$ (since constructors cannot produce trees of name sorts).

α-trees represent the object-language terms quotiented by α-equivalence which are so frequently used in informal mathmatical parlance. They will form the basis for the semantics of non-permutative nominal constraints. We now extend the standard notions of "free names" and "freshness" from nominal abstract syntax to α-trees.

Definition 3.5 (Free names and freshness for α-trees). Suppose that $\mathfrak{a} \in \alpha\text{-}Tree_\Sigma(T)$. Then, we write $FN(\mathfrak{a})$ for the finite set $FN(g)$ for some/any[1] ground tree $g \in Tree_\Sigma(T)$ such that $\mathfrak{a} = [g]_\alpha$. Furthermore, if $\mathfrak{a} \in \alpha\text{-}Tree_\Sigma(N)$ then we know that $\mathfrak{a} = [n]_\alpha$ for some $n \in Name(N)$. Then, we write $\mathfrak{a} \mathrel{\#} \mathfrak{a}'$ and say "$\mathfrak{a}$ is fresh for $\mathfrak{a}'$" iff $n \notin FN(\mathfrak{a}')$.

We now describe the instantiation of schematic terms, which involves replacing the variables with α-trees to produce specific ground instances.

Definition 3.6 (α-tree valuations). An α-*tree valuation* V is a finite partial function which maps variables to α-trees. We write $dom(V)$ for the domain of the partial function. Given a type environment Δ we write $\alpha\text{-}Tree_\Sigma(\Delta)$ for the set of all α-tree valuations V such that $dom(V) = dom(\Delta)$ and $V(x) \in \alpha\text{-}Tree_\Sigma(\Delta(x))$ for all $x \in dom(V)$. This ensures that the valuation respects types.

Using the proof techniques from [Pit06] we can show that there exists an instantiation operation $[\![t]\!]_V$ which respects both types and α-equivalence classes, i.e. if $\Delta \vdash_\Sigma t : T$ and $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ then $[\![t]\!]_V \in \alpha\text{-}Tree_\Sigma(T)$ holds, where $[\![t]\!]_V$ is as follows.

$$[\![x]\!]_V = V(x)$$
$$[\![t]\!]_V = [g]_\alpha \implies [\![K\,t]\!]_V = [K\,g]_\alpha$$
$$[\![()]\!]_V = [()]_\alpha$$
$$[\![t_1]\!]_V = [g_1]_\alpha \wedge \cdots \wedge [\![t_k]\!]_V = [g_k]_\alpha \implies [\![(t_1, \ldots, t_k)]\!]_V = [(g_1, \ldots, g_k)]_\alpha$$
$$V(x) = [n]_\alpha \wedge [\![t]\!]_V = [g]_\alpha \implies [\![\texttt{<}x\texttt{>}t]\!]_V = [\texttt{<}n\texttt{>}g]_\alpha$$

Since variables stand for unknown α-trees, not unknown trees, we see that schematic terms describe an α-tree as opposed to a tree. Precisely which one depends on how the variables

---

[1]Some/any properties are characteristic of nominal techniques for representing abstract syntax—see [Pit06] for a rigorous mathematical treatment.

in the term are instantiated. This reflects the common practice of leaving α-equivalence implicit and using representatives to stand in place of the whole class [Bar84, Convention 2.1.13].

Applying an α-tree valuation to a schematic term is "possibly-capturing" with regard to the binders in the underlying language of ground abstract syntax trees, even when in abstraction position. For example, given distinct variables x, y, z we cannot regard the schematic terms `<x>z` and `<y>z` as equivalent because if we let $V = \{x \mapsto [n]_\alpha, y \mapsto [n']_\alpha, z \mapsto [n]_\alpha\}$ (with $n \neq n'$) then we get

$$[\![\texttt{<x>z}]\!]_V = [\texttt{<n>n}]_\alpha \neq [\texttt{<n'>n}]_\alpha = [\![\texttt{<y>z}]\!]_V.$$

Barendregt [Bar84] does not draw a distinction between names and schematic variables, but in our presentation there is a clear distinction between non-permutative variables x and permutative names n. Even when in abstraction position, we allow non-permutative variables to be *aliased*, that is, we allow syntactically distinct variables to be assigned with the same underlying name. We can model names which behave permutatively by imposing additional constraints that the variables must be mutually distinct [Lak10, Section 6.4].

Lemma 3.7 (Substitution lemma). *If $\Delta \vdash_\Sigma t':T$ and $\Delta \vdash_\Sigma x = t$ ok and $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ all hold then $V(x) = [\![t]\!]_V$ implies $[\![t']\!]_V = [\![t'[t/x]]\!]_V$.* $\square$

We are now in a position to define the semantics of non-permutative nominal constraint problems, in terms of satisfying ground instantiations by α-tree valuations.

Definition 3.8 (Satisfaction of atomic constraints). For an atomic constraint c such that $\Delta \vdash_\Sigma c$ ok and an α-tree valuation $V \in \alpha\text{-}Tree_\Sigma(\Delta)$, we write $V \models c$ to mean that "V satisfies c", which is defined by cases on c.

$$\frac{[\![t]\!]_V = [\![t']\!]_V}{V \models t = t'} \qquad\qquad \frac{V(x) \not\# [\![t]\!]_V}{V \models x \,\#\, t}$$

In the case of freshness constraints, this relation is well-defined by virtue of the points noted in Definition 3.5.

Definition 3.9 (Satisfiable constraint problems). For a constraint problem $\exists\Delta(\overline{c})$ such that $\emptyset \vdash_\Sigma \exists\Delta(\overline{c})$ ok, we say that $\models \exists\Delta(\overline{c})$ holds iff there exists a valuation $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ such that $V \models c$ for all $c \in \overline{c}$.

For example, suppose that we have variables x and y of some name sort N. Then, the α-tree constraint problem `<x>y=<y>x` is satisfied by any valuation V such that $V(x) = V(y)$, as both sides of the equality constraint are then instantiated to the same α-equivalence class $[\texttt{<n>n}]_\alpha$. Note that if we used permutative names n_x and n_y then the corresponding constraint problem `<nx>ny=<ny>nx` would *not* be satisfiable, because the two terms are ground but are not in the same α-equivalence class. This corresponds to the non-permutative constraint problem `<x>y = <y>x & x # y` where we simulate permutative behaviour by adding appropriate freshness constraints. This constraint problem is also unsatisfiable, because the first constraint `<x>y = <y>x` is only satisfiable by a valuation V if $V(x) = V(y)$. However, no such valuation can satisfy the freshness constraint $x \,\#\, y$.

Definition 3.10. We write NonPermSat for the decision problem $\{(\Delta, \overline{c}) \mid \models \exists\Delta(\overline{c})\}$.

It is trivial to see that NonPermSat is decidable because it is a syntactic subset of the equivariant unification problem, which was shown to be decidable by Cheney in his

thesis [Che04, Chapter 7]. In previous work [LP09] we showed that NonPermSat is is NP-hard by a reduction of graph 3-colourability. In Section 6 of this paper we present an alternative proof of NP-hardness by defining an encoding of name-name equivariant unification problems (which are known to be NP-complete [Che10]) into NPNAS. It follows from [Che10, Theorem 1] that NonPermSat is in NP and hence that NonPermSat is NP-complete.

Remark 3.11 (Permutations and equivariance). This section has hardly mentioned name-permutations, which are a staple of most nominal techniques for abstract syntax involving binders (as discussed in Section 2.1). They are not required because α-equivalence is handled by the explicit use of α-equivalence classes of ground trees in the semantics. However, it is not hard to show that the semantics of non-permutative nominal constraints is equivariant, that is, that (for any c) the set $\{V \mid V \models c\}$ is closed under name-permutations.

4. Constraint transformation algorithm

We define a non-deterministic transition relation, $\longrightarrow$, which transforms a single constraint problem into a finite, non-empty set of constraint problems. Figure 3 presents transition rules for the transformation relation. To save space, we write $x \mathbin{\#} x_{1..k}$ for the conjunction $x \mathbin{\#} x_1 \mathbin{\&} \cdots \mathbin{\&} x \mathbin{\#} x_k$. We also write $\texttt{<}x_{1..k}\texttt{>}t$ as a shorthand for the iterated abstraction term $\texttt{<}x_1\texttt{>}\cdots\texttt{<}x_k\texttt{>}t$, where the list of abstractions may be empty unless explicitly stated otherwise. Note that Figure 3 does not contain explicit rules for handling constraints of the form $\texttt{<}x_{1..k}\texttt{><}x\texttt{>}t = \texttt{<}y_{1..k}\texttt{><}y\texttt{>}t'$ or $x \mathbin{\#} \texttt{<}y_{1..k}\texttt{><}y\texttt{>}t'$ because the additional abstraction term-former is implicitly folded into the initial abstraction list during pattern-matching.

Rules (F1)–(F3) and (E1)–(E3) deal with unit, data and tuple terms in the usual way: the only difference is that we work within nested abstractions. The abstractions do not play any part in these six rules, except that the lists on both sides of an equality constraint must be of the same length. The rules (F5) and (E5) dispose of trivial constraints: in the case of (F5), two names of different sorts will always be fresh for each other and in the case of (E5), any term is equal to itself.

The most interesting rules are (F4) and (E4), which deal with the scopes of bound names with respect to the nested abstractions. We first consider (F4). In order for x to not appear free anywhere in $\texttt{<}y_{1..k}\texttt{>}y$, either x should map to the same name as one of the abstracted variables $y_1, \ldots, y_k$ or x should be distinct from all of the abstracted variables and be constrained to be fresh for the unknown term y. Unlike in nominal unification, transforming a freshness constraint with this rule may produce new equality constraints to solve.

Rule (E4) deals with equality constraints between variables of some name sort N. We handle these constraints by noting that the way to resolve the binding scope of the names x and y is to start at the innermost binding occurrence and work towards the outside. Therefore, it should be the case that either x and y both unify with the innermost binder (x_k and y_k respectively), or that they should both be distinct from the innermost binder and unify with the next one moving outwards (i.e. x_{k-1} and y_{k-1}), and so on, or that x and y should be distinct from all of the potential binders and equal to each other. This method of dealing with equality constraints between bound names seems more natural than existing methods based on name-swapping.

The final two rules, (E6) and (E7), eliminate variables from the problem by substituting throughout the remaining constraints. They use a notion of substitution $\overline{c}[t/x]$ which replaces all occurrences of the variable x in $\overline{c}$ by the term t. These substitutions are capturing with respect to the abstraction term-former. Rule (E6) is the standard variable elimination rule from first-order (syntactic) unification. The side-condition $x \notin vars(t)$ on this rule enforces the *occurs check* which is necessary to avoid cyclic substitutions. The side-condition $x \in vars(\overline{c})$ ensures that this rule can only be invoked once per variable, which is necessary for termination.

Rule (E7) deals with equality constraints of the form $<x_{1..k}>x\texttt{=}<y_{1..k}>t$, where $k > 0$, the occurs check succeeds and t is not a variable, i.e. t is some compound term. This includes the case where there are more initial nested abstractions on one side than the other. We cannot simply substitute t for x here because of the preceding abstractions: x might need to be instantiated with a term syntactically different from t. For example, to satisfy the constraints $(x \mathbin{\#} y) \mathbin{\&} (<x>x^* \texttt{=} <y>K\,y)$ it is clear that x^* must be mapped to $K\,x$, not $K\,y$. This is where swappings are necessary in both nominal and equivariant unification: however, this is not an elegant solution when bound names are represented using variables, because the potential for aliasing means that the result of a "variable swapping" such as $(x\,y) \cdot z$ is not unique.

Since we cannot make progress using a swapping, we note that the side-condition that t may not be a variable means that we know the outermost constructor of t. This allows us to impose some structure on the unknown term represented by x by narrowing [AEH00]. The rules from Figure 4 define a narrowing relation which factors out this common functionality at unit, tuple, data and abstraction types.

The intuitive reading of $[\Delta; t] \Longmapsto [\Delta^*; t^*]$ is that the term t^* represents a "pattern" for terms with the same outermost constructor as t. The subterms of t^* are variables which stand for the (as-yet unknown) subterms of the term referred to by the variable x. The extra type environment Δ^* is needed to ensure that the variables used to create t^* do not appear elsewhere in the constraint problem. They must also be mutually distinct, as in rules (N3) and (N4). This gives rise to the following lemma, which is proved by cases on the narrowing rules.

Lemma 4.1 (Narrowing and typing)**.** *If $[\Delta; t] \Longmapsto [\Delta^*; t^*]$ and $\Delta \vdash_\Sigma t\!:\!T$ then $dom(\Delta) \cap dom(\Delta^*) = \emptyset$ and $\Delta^* \vdash_\Sigma t^*\!:\!T$.* $\qquad\square$

The narrowing procedure is lazy in the sense that each narrowing step using rule (E7) does not replicate the entire structure of the term t but just its outermost constructor. If there is further structure on one side, rule (E7) may be applied repeatedly. There is no rule for narrowing against variables because they have no internal structure to copy. Constraints of the form $<x_{1..k}>x \texttt{=} <y_{1..k}>y$ are simply left alone when $\Delta(x)$ is not a name sort—this is in direct contrast to nominal unification. It is not immediately obvious that this is correct, and we will address this point in the proof of Lemma 5.28 below.

Remark 4.2 (Relationship to existing algorithms)**.** Since NonPermSat is a syntactic subset of the full equivariant unification problem studied by Cheney, the problems considered here could be handled using a subset of the rules presented in [Che10] for solving "general nominal unification" problems. The main difference is that our rules use the lazy narrowing approach to delay case analysis until the body of the abstractions is itself a variable of name sort. We also note that a simplified narrowing-based approach was used to solve swapping-free equivariant matching problems in polynomial time in [Che10, Section 5.2].

(F1): $\exists\Delta((x \mathbin{\#} \mathord{<}y_{1..k}\mathord{>}()) \mathbin{\&} \overline{c}) \longrightarrow \exists\Delta(\overline{c})$

(F2): $\exists\Delta((x \mathbin{\#} \mathord{<}y_{1..k}\mathord{>}K\,t) \mathbin{\&} \overline{c}) \longrightarrow \exists\Delta((x \mathbin{\#} \mathord{<}y_{1..k}\mathord{>}t) \mathbin{\&} \overline{c})$

(F3): $\exists\Delta((x \mathbin{\#} \mathord{<}y_{1..k}\mathord{>}(t_1, \ldots, t_j)) \mathbin{\&} \overline{c})$
$\qquad \longrightarrow \exists\Delta((x \mathbin{\#} \mathord{<}y_{1..k}\mathord{>}t_1) \mathbin{\&} \cdots \mathbin{\&} (x \mathbin{\#} \mathord{<}y_{1..k}\mathord{>}t_j) \mathbin{\&} \overline{c})$

(F4): $\exists\Delta((x \mathbin{\#} \mathord{<}y_{1..k}\mathord{>}y) \mathbin{\&} \overline{c})$

$$\longrightarrow \begin{cases} \exists\Delta((x = y_1) \mathbin{\&} \overline{c}) & \text{if } \Delta(x) = \Delta(y_1). \\ \exists\Delta((x \mathbin{\#} y_1) \mathbin{\&} (x = y_2) \mathbin{\&} \overline{c}) & \text{if } \Delta(x) = \Delta(y_2). \\ \cdots & \cdots \\ \exists\Delta((x \mathbin{\#} y_{1..k-1}) \mathbin{\&} (x = y_k) \mathbin{\&} \overline{c}) & \text{if } \Delta(x) = \Delta(y_k). \\ \exists\Delta((x \mathbin{\#} y_{1..k}) \mathbin{\&} (x \mathbin{\#} y) \mathbin{\&} \overline{c}) \end{cases}$$
$\qquad$ if $k > 0$.

(F5): $\exists\Delta((x \mathbin{\#} y) \mathbin{\&} \overline{c}) \longrightarrow \exists\Delta(\overline{c})$
$\qquad$ if $\Delta(x) = N$, $\Delta(y) = N'$ and $N \neq N'$.

(E1): $\exists\Delta((\mathord{<}x_{1..k}\mathord{>}() = \mathord{<}y_{1..k}\mathord{>}()) \mathbin{\&} \overline{c}) \longrightarrow \exists\Delta(\overline{c})$

(E2): $\exists\Delta((\mathord{<}x_{1..k}\mathord{>}K\,t = \mathord{<}y_{1..k}\mathord{>}K\,t') \mathbin{\&} \overline{c}) \longrightarrow \exists\Delta((\mathord{<}x_{1..k}\mathord{>}t = \mathord{<}y_{1..k}\mathord{>}t') \mathbin{\&} \overline{c})$

(E3): $\exists\Delta((\mathord{<}x_{1..k}\mathord{>}(t_1, \ldots, t_j) = \mathord{<}y_{1..k}\mathord{>}(t'_1, \ldots, t'_j)) \mathbin{\&} \overline{c})$
$\qquad \longrightarrow \exists\Delta((\mathord{<}x_{1..k}\mathord{>}t_1 = \mathord{<}y_{1..k}\mathord{>}t'_1) \mathbin{\&} \cdots \mathbin{\&} (\mathord{<}x_{1..k}\mathord{>}t_j = \mathord{<}y_{1..k}\mathord{>}t'_j) \mathbin{\&} \overline{c})$

(E4): $\exists\Delta((\mathord{<}x_{1..k}\mathord{>}x = \mathord{<}y_{1..k}\mathord{>}y) \mathbin{\&} \overline{c})$

$$\longrightarrow \begin{cases} \exists\Delta((x = x_k) \mathbin{\&} (y = y_k) \mathbin{\&} \overline{c}) & \text{if } \Delta(x) = \Delta(x_k). \\ \exists\Delta((x \mathbin{\#} x_k) \mathbin{\&} (x = x_{k-1}) \mathbin{\&} (y \mathbin{\#} y_k) \mathbin{\&} (y = y_{k-1}) \mathbin{\&} \overline{c}) & \text{if } \Delta(x) = \Delta(x_{k-1}). \\ \cdots & \cdots \\ \exists\Delta((x \mathbin{\#} x_{k..2}) \mathbin{\&} (x = x_1) \mathbin{\&} (y \mathbin{\#} y_{k..2}) \mathbin{\&} (y = y_1) \mathbin{\&} \overline{c}) & \text{if } \Delta(x) = \Delta(x_1). \\ \exists\Delta((x \mathbin{\#} x_{k..1}) \mathbin{\&} (y \mathbin{\#} y_{k..1}) \mathbin{\&} (x = y) \mathbin{\&} \overline{c}) \end{cases}$$
$\qquad$ if $k > 0$ and $\Delta(x) = N$, for some N.

(E5): $\exists\Delta((x = x) \mathbin{\&} \overline{c}) \longrightarrow \exists\Delta(\overline{c})$

(E6): $\left. \begin{array}{l} \exists\Delta((x = t) \mathbin{\&} \overline{c}) \\ \exists\Delta((t = x) \mathbin{\&} \overline{c}) \end{array} \right\} \longrightarrow \exists\Delta((x = t) \mathbin{\&} \overline{c}[t/x])$
$\qquad$ if $x \notin vars(t)$ and $x \in vars(\overline{c})$.

(E7): $\left. \begin{array}{l} \exists\Delta((\mathord{<}x_{1..k}\mathord{>}x = \mathord{<}y_{1..k}\mathord{>}t) \mathbin{\&} \overline{c}) \\ \exists\Delta((\mathord{<}y_{1..k}\mathord{>}t = \mathord{<}x_{1..k}\mathord{>}x) \mathbin{\&} \overline{c}) \end{array} \right\}$
$\qquad \longrightarrow \exists\Delta, \Delta^*((x = t^*) \mathbin{\&} (\mathord{<}x_{1..k}\mathord{>}t^* = \mathord{<}y_{1..k}\mathord{>}t) \mathbin{\&} \overline{c}[t^*/x])$
$\qquad$ if t is not a variable, $x \notin vars(t)$, $k > 0$ and $[\Delta; t] \Longmapsto [\Delta^*; t^*]$.

Figure 3: Constraint transformation rules

$$(N1) \; \frac{}{[\Delta; ()] \Longmapsto [\emptyset; ()]} \qquad\qquad (N2) \; \frac{x \notin dom(\Delta) \qquad (K : T \to D) \in \Sigma}{[\Delta; K\,t] \Longmapsto [\{x : T\}; K\,x]}$$

$$(N3) \; \frac{\Delta \vdash_\Sigma (t_1, \ldots, t_k) : T_1 * \cdots * T_k \qquad x_1 \neq \cdots \neq x_k \notin dom(\Delta)}{[\Delta; (t_1, \ldots, t_k)] \Longmapsto [\{x_1 : T_1, \ldots, x_k : T_k\}; (x_1, \ldots, x_k)]}$$

$$(N4) \; \frac{\Delta \vdash_\Sigma \mathord{<}x\mathord{>}t : [N]T \qquad x' \neq x'' \notin dom(\Delta)}{[\Delta; \mathord{<}x\mathord{>}t] \Longmapsto [\{x' : N, x'' : T\}; \mathord{<}x'\mathord{>}x'']}$$

Figure 4: Narrowing rules

We conclude this section with the straightforward result that well-formedness of constraint problems is preserved by the transformation rules. The proof is by cases on the transformation rules from Figure 3. In the case for rule (E7) we require Lemma 4.1 to deduce that the narrowing step preserves well-formedness.

Lemma 4.3 (Preservation of well-formedness). *If $\emptyset \vdash_\Sigma \exists\Delta(\overline{c})$ ok and $\exists\Delta(\overline{c}) \longrightarrow \exists\Delta'(\overline{c}')$ then $\emptyset \vdash_\Sigma \exists\Delta'(\overline{c}')$ ok. Furthermore, $\Delta' \supseteq \Delta$.* □

5. A CORRECT DECISION PROCEDURE

We now present an algorithm for deciding satisfiability of non-permutative nominal constraint problems. We begin by considering the correctness of individual transformation rules from the previous section and prove that, with careful consideration of termination, the transformation rules can be used to give a correct decision procedure for NonPermSat.

5.1. **Soundness and completeness of transformations.** We first prove soundness and completeness results for the individual constraint transformation rules from Figure 3. We begin by stating a lemma which relates substitution and constraint satisfaction, which will be needed for the cases for rules (E6) and (E7) which involve substitution.

Lemma 5.1 (Substitution property of satisfaction). *Suppose that $\emptyset \vdash_\Sigma \exists\Delta, x{:}T(\overline{c})$ ok, $\Delta \vdash_\Sigma t{:}T$, $V \in \alpha\text{-}Tree_\Sigma(\Delta, x{:}T)$ and $V(x) = [\![t]\!]_V$. Then $V \models \overline{c}[t/x]$ iff $V \models \overline{c}$.* □

We now prove that the transformation rules are *sound*, i.e. that the transformation steps do not introduce any additional satisfying valuations to the problem.

Theorem 5.2 (Soundness of transformations). *Suppose that $\emptyset \vdash_\Sigma \exists\Delta(\overline{c})$ ok, $\exists\Delta(\overline{c}) \longrightarrow \exists\Delta'(\overline{c}')$ and $V' \models \overline{c}'$ all hold, where $V' \in \alpha\text{-}Tree_\Sigma(\Delta')$. Then $V \models \overline{c}$ holds, where V is the restriction of V' to $dom(\Delta)$.*

Proof. By case analysis on the transformation rule used to derive $\exists\Delta(\overline{c}) \longrightarrow \exists\Delta'(\overline{c}')$. The cases for rules (F1)–(F4) and (E1)–(E4) are straightforward, using standard facts about the definition of constraint satisfaction. The case for (E5) follows because $V \models x = x$ holds for any V and x. Similarly, the case for (F5) follows because $V \models x \mathrel{\#} y$ holds for any V, x and y if $\Delta(x)$ and $\Delta(y)$ are different name sorts. The case for (E6) relies on Lemma 5.1. The remaining case, for (E7), is dealt with in detail below.

(E7): In this case we have $\overline{c} = (\mathord{<}x_{1..k}\mathord{>}x = \mathord{<}y_{1..k}\mathord{>}t) \mathrel{\&} \overline{c}^*$ and furthermore that $\overline{c}' = (x = t^*) \mathrel{\&} (\mathord{<}x_{1..k}\mathord{>}t^* = \mathord{<}y_{1..k}\mathord{>}t) \mathrel{\&} \overline{c}[t^*/x]$, where t is not a variable, $x \notin vars(t)$, $k > 0$ and $[\Delta; t] \Longmapsto [\Delta^*; t^*]$. Furthermore, $\Delta' = \Delta, \Delta^*$. By assumption we get that $V' \models x = t^*$, $V' \models \mathord{<}x_{1..k}\mathord{>}t^* = \mathord{<}y_{1..k}\mathord{>}t$ and $V' \models \overline{c}^*[t^*/x]$ all hold, for some $V' \in \alpha\text{-}Tree_\Sigma(\Delta, \Delta^*)$. From $V' \models x = t^*$ we know that $V'(x) = [\![t^*]\!]_{V'}$, (since $x \notin vars(t)$) and then by Lemma 5.1 and $V' \models \overline{c}^*[t^*/x]$ we get that $V' \models \overline{c}^*$ holds. Furthermore, we know that $\mathord{<}x_{1..k}\mathord{>}t^* = \mathord{<}y_{1..k}\mathord{>}t$ is $(\mathord{<}x_{1..k}\mathord{>}x = \mathord{<}y_{1..k}\mathord{>}t)[t^*/x]$, because $x \notin vars(t)$. Therefore, by $V' \models \mathord{<}x_{1..k}\mathord{>}t^* = \mathord{<}y_{1..k}\mathord{>}t$ and Lemma 5.1 we can show that $V' \models \mathord{<}x_{1..k}\mathord{>}x = \mathord{<}y_{1..k}\mathord{>}t$ holds. Thus we get that $V' \models \overline{c}$ holds, and hence that $V \models \overline{c}$, as required. □

Next we prove that the constraint transformation rules are *complete*, i.e. that every satisfying valuation is preserved across some transformation of a constraint problem. We present some preliminary lemmas concerning narrowing before moving on to the main proof.

Lemma 5.3 (Possibility of narrowing). *If t is not a variable and $\Delta \vdash_\Sigma t : T$ then there exist Δ^* and t^* such that $[\Delta; t] \Longmapsto [\Delta^*; t^*]$ holds.*

Proof. By cases on the narrowing rules from Figure 4, which cover all syntactic cases for t except for when t is a variable. In each case, the types from the typing assumption $\Delta \vdash_\Sigma t : T$ match those required by the appropriate narrowing rule, so we can apply that rule to find Δ^* and t^* such that $[\Delta; t] \Longmapsto [\Delta^*; t^*]$, as required. $\qquad\square$

Lemma 5.4 (Narrowing and satisfaction). *Suppose that $\Delta \vdash_\Sigma <x_{1..k}>x = <y_{1..k}>t$ ok and that $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ is such that $V \models <x_{1..k}>x = <y_{1..k}>t$ holds. If $[\Delta; t] \Longmapsto [\Delta^*; t^*]$ then there exists $V^* \in \alpha\text{-}Tree_\Sigma(\Delta, \Delta^*)$ which agrees with V on $dom(\Delta)$ and is such that $V^*(x) = [\![t^*]\!]_{V^*}$.*

Proof. From $[\Delta; t] \Longmapsto [\Delta^*; t^*]$ and Lemma 4.1 we get that $dom(\Delta) \cap dom(\Delta^*) = \emptyset$. Thus it follows that there exist a family of valuations $V^* \in \alpha\text{-}Tree_\Sigma(\Delta, \Delta^*)$ which agree with V on $dom(\Delta)$, and it just remains to show that $V^*(x) = [\![t^*]\!]_{V^*}$ holds for some such V^*. We prove this by cases on the narrowing rule used to derive $[\Delta; t] \Longmapsto [\Delta^*; t^*]$. In each case, $V \models <x_{1..k}>x = <y_{1..k}>t$ implies that the outermost term-former of $V^*(x)$ is the same as that of $[\![t]\!]_{V^*}$, and since $[\Delta; t] \Longmapsto [\Delta^*; t^*]$ this is also the same as the outermost term-former of $[\![t^*]\!]_{V^*}$. Thus we can choose a valuation V^* which instantiates the variables in $dom(\Delta^*)$ such that $V^*(x) = [\![t^*]\!]_{V^*}$ holds, as required. $\qquad\square$

Definition 5.5 (Successor sets). We write $succ(\exists\Delta(\overline{c}))$ for the *successor set* of $\exists\Delta(\overline{c})$, which we define as the set $\{\exists\Delta'(\overline{c'}) \mid \exists\Delta(\overline{c}) \longrightarrow \exists\Delta'(\overline{c'})\}$.

Theorem 5.6 (Completeness of transformations). *Suppose that $\emptyset \vdash_\Sigma \exists\Delta(\overline{c})$ ok, $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ and $V \models \overline{c}$ all hold, and that $succ(\exists\Delta(\overline{c})) \neq \emptyset$. Then there exists $\exists\Delta'(\overline{c'}) \in succ(\overline{c})$ and $V' \in \alpha\text{-}Tree_\Sigma(\Delta')$ such that $V' \models \overline{c'}$, where V and V' agree on $dom(\Delta)$.*

Proof. Since $succ(\exists\Delta(\overline{c})) \neq \emptyset$ it follows that $\overline{c}$ matches the left-hand side of one of the constraint transformation rules from Figure 3 and satisfies any side-conditions. Then, the proof is by case analysis on $\overline{c}$. The cases of $\overline{c}$ which match rules (F1)–(F4) and (E1)–(E4) are straightforward and follow from standard properties of constraint satisfaction.

If $\overline{c}$ is $(x = x) \, \& \, \overline{c}$ the transition uses rule (E5) and the result is trivial by assumption, the problem on the right-hand side being a subset of the problem on the left-hand side. If $\overline{c}$ is $(x \# y) \, \& \, \overline{c}$, where x and y are distinct variables of different name sorts, then (F5) applies and again the result follows trivially. We invoke Lemma 5.1 in the case for (E6). The case for (E7) is less straightforward, and we give details for this below.

Case $\overline{c} = (<x_{1..k}>x = <y_{1..k}>t) \, \& \, \overline{c}^*$: We also assume that t is not a variable, $x \notin vars(t)$, and $k > 0$ all hold. Furthermore, we assume that $V \in \alpha\text{-}Tree_\Sigma(\Delta)$, where $V \models <x_{1..k}>x = <y_{1..k}>t$ and $V \models \overline{c}^*$ both hold. Since t is not a variable, by Lemma 5.3 we get that $[\Delta; t] \Longmapsto [\Delta^*; t^*]$ holds for some Δ^* and t^*. We can then match against rule (E7) and get that $\overline{c'} = (x = t^*) \, \& \, (<x_{1..k}>t^* = <y_{1..k}>t) \, \& \, \overline{c}^*[t^*/x]$ and $\Delta' = \Delta, \Delta^*$. By Lemma 5.4 there exists $V' \in \alpha\text{-}Tree_\Sigma(\Delta')$ which agrees with V on $dom(\Delta)$ and is such that $V'(x) = [\![t^*]\!]_{V'}$. It follows that $V' \models x = t^*$ and $V' \models <x_{1..k}>t^* = <y_{1..k}>t$ both hold. Finally, we can use Lemma 5.1 to deduce that $V' \models \overline{c}^*[t^*/x]$, and hence that $V' \models \overline{c'}$ holds, as required. $\qquad\square$

The following corollary follows immediately from Theorem 5.2 and Theorem 5.6, and summarises the results of this section.

Corollary 5.7 (Soundness and completeness of transformations). *Assume $\emptyset \vdash_\Sigma \exists\Delta(\overline{c})$ ok and $succ(\exists\Delta(\overline{c})) = \{\exists\Delta, \Delta_1(\overline{c}_1), \ldots, \exists\Delta, \Delta_k(\overline{c}_k)\}$ both hold. Then, for any $V \in \alpha\text{-}Tree_\Sigma(\Delta)$, $V \models \overline{c}$ iff there exists a valuation V' which extends V to $dom(\Delta, \Delta_i)$ and is such that $V' \models \overline{c}_i$, for some $i \in \{1, \ldots, k\}$.* $\qquad\square$

5.2. **Termination.** A key feature of any decision procedure is that it must always terminate, but here we run into a problem: for some constraint problems it is possible to get infinite reduction sequences. For example, given a datatype `nat` for natural numbers, if $n > 0$ then the constraint problem

$$\exists\Delta((\texttt{<}x_{1..k}\texttt{>}x = \texttt{<}y_{1..k}\texttt{>}\texttt{S}\,y) \,\&\, (\texttt{<}y_{1..k}\texttt{>}y = \texttt{<}x_{1..k}\texttt{>}\texttt{S}\,x))$$

can be reduced to

$$\exists\Delta, x'\texttt{:nat}, y'\texttt{:nat}((x = \texttt{S}\,x') \,\&\, (y = \texttt{S}\,y') \,\&\, (\texttt{<}x_{1..k}\texttt{>}x' = \texttt{<}y_{1..k}\texttt{>}\texttt{S}\,y') \,\&\, (\texttt{<}y_{1..k}\texttt{>}y' = \texttt{<}x_{1..k}\texttt{>}\texttt{S}\,x')).$$

There is clearly the possibility of divergence as we have recovered a variant of the original problem. In this section we attempt to address this problem by defining a decidable test on constraint problems and proving that this test allows us to avoid reducing may-divergent problems. We begin by introducing some terminology—we say that a constraint problem $\exists\Delta(\overline{c})$ is

- *terminal* (written $\exists\Delta(\overline{c}) \not\longrightarrow$) if there does not exist a constraint problem $\exists\Delta'(\overline{c}')$ such that $\exists\Delta(\overline{c}) \longrightarrow \exists\Delta'(\overline{c}')$.
- a $\longrightarrow$-*normal form* of $\exists\Delta^*(\overline{c}^*)$ if there exists a finite transformation sequence from $\exists\Delta^*(\overline{c}^*)$ to $\exists\Delta(\overline{c})$ and $\exists\Delta(\overline{c})$ is terminal.
- *strongly normalising* if all transformation sequences starting from $\exists\Delta(\overline{c})$ eventually reach a terminal constraint problem.
- *may-divergent* if there exists an infinite transformation sequence starting from $\exists\Delta(\overline{c})$.

Our termination check will involve translating elements of $Prob_\Sigma$ into a subset of $Prob_\Sigma$ which corresponds to first-order unification problems. Since first-order unification is known to be decidable, we can check whether the first-order unification problem that underlies a given non-permutative constraint problem is satisfiable. From this we will deduce whether the non-permutative constraint problem is strongly normalising.

We refer to this abstraction interpretation process as "first-order reduction", and begin by reducing nominal signatures and types to first-order versions.

Definition 5.8 (Reducing nominal signatures). For every nominal signature Σ we write $\Sigma^\flat$ for the underlying first-order signature, which has $\mathbb{N}_{\Sigma^\flat} \triangleq \emptyset$ and $\mathbb{D}_{\Sigma^\flat} \triangleq \mathbb{D}_\Sigma$. Furthermore, if $(K\!:\!T \to D) \in \Sigma$ then $(K\!:\!T^\flat \to D) \in \Sigma^\flat$ holds, where $T^\flat$ is defined as follows.

$$N^\flat \triangleq \texttt{unit} \qquad ([N]T)^\flat \triangleq \texttt{unit} * (T^\flat) \qquad D^\flat \triangleq D \qquad \texttt{unit}^\flat \triangleq \texttt{unit}$$

$$(T_1 * \cdots * T_k)^\flat \triangleq (T_1{}^\flat) * \cdots * (T_k{}^\flat).$$

Note that we reduce all name sorts to the unit type and all abstraction type-formers $[N]T$ to the product type $\texttt{unit} * (T^\flat)$. Thus we lose all information on the types of object-level names but retain some information on the locations of abstractions in the original type.

Definition 5.9 (Reducing type environments). For any type environment Δ we write $\Delta^\flat$ for the type environment such that

- $dom(\Delta^\flat) = \{x \mid x \in dom(\Delta) \text{ and } \Delta(x) \text{ is } not \text{ a name sort}\}$; and
- $\Delta^\flat(x) = (\Delta(x))^\flat$ for all $x \in dom(\Delta^\flat)$.

Note that $dom(\Delta^\flat) \subseteq dom(\Delta)$ by definition. We now define a similar first-order reduction operation on non-permutative nominal terms—this definition includes a type environment Δ as a parameter, which is necessary to decide how to handle variables when they are encountered during the reduction process. This definition is related to the morphism between nominal and first-order terms defined in [CF11].

Definition 5.10 (Reducing terms). For a term t, let Δ be any type environment such that $\Delta \vdash_\Sigma t : T$ holds for some T. Then, define the first-order reduction of t under Δ, $t^\flat_\Delta$, as follows.

$$x^\flat_\Delta \triangleq \begin{cases} () & \text{if } \Delta(x) \text{ is a name sort} \\ x & \text{if } \Delta(x) \text{ is } not \text{ a name sort} \end{cases} \qquad (\texttt{<}x\texttt{>}t)^\flat_\Delta \triangleq ((), t^\flat_\Delta) \qquad ()^\flat_\Delta \triangleq ()$$

$$(K\, t)^\flat_\Delta \triangleq K\, (t^\flat_\Delta) \qquad\qquad (t_1, \ldots, t_k)^\flat_\Delta \triangleq ((t_1{}^\flat_\Delta), \ldots, (t_k{}^\flat_\Delta))$$

Mirroring Definition 5.8, we turn any variables of name sort into unit terms and translate abstraction term-formers into a pair consisting of unit and the reduced abstraction body. This will be convenient later on because it ensures that the "size" of a reduced term $t^\flat_\Delta$ (defined below) is the same as the size of the original term t. It is straightforward to show that if $\Delta \vdash_\Sigma t : T$ then $\Delta^\flat \vdash_{\Sigma^\flat} t^\flat_\Delta : T^\flat$. We also get a "weakening" result: if $\Delta \vdash_\Sigma t : T$ and $\Delta' \supseteq \Delta$ then $t^\flat_{\Delta'} = t^\flat_\Delta$. We now extend the definitions to constraint problems.

Definition 5.11 (Reducing constraint problems). The reduction $\overline{c}^\flat_\Delta$ of constraints $\overline{c}$ under Δ, where $\Delta \vdash_\Sigma \overline{c}\ ok$, is defined as follows.

$$\overline{c}^\flat_\Delta \quad \triangleq \quad \{t_1{}^\flat_\Delta = t_2{}^\flat_\Delta \mid (t_1 = t_2) \in \overline{c}\}.$$

For a constraint problem $\exists\Delta(\overline{c})$, we write $(\exists\Delta(\overline{c}))^\flat$ for the corresponding reduced constraint problem $\exists\Delta^\flat(\overline{c}^\flat_\Delta)$. It is trivial to show that if $\emptyset \vdash_\Sigma \exists\Delta(\overline{c})\ ok$ holds then $\emptyset \vdash_{\Sigma^\flat} (\exists\Delta(\overline{c}))^\flat\ ok$ also holds.

When reducing constraint problems we discard any freshness constraints, since these are not present in first-order unification problems. This is not an issue since freshness constraints cannot cause may-divergence in our constraint transformation rules. For equality constraints, we simply apply the first-order reduction defined in Definition 5.10 to both terms separately.

In order to reason about the satisfaction of reduced constraint problems we must define first-order reductions of ground trees, α-trees and α-tree valuations.

Definition 5.12 (Reducing ground trees)**.** We define the first-order reduction $g^\flat$ of a ground tree g as follows.

$$n^\flat \triangleq () \qquad\qquad ()^\flat \triangleq () \qquad\qquad (\texttt{<}n\texttt{>}g)^\flat \triangleq ((), g^\flat) \qquad\qquad (K\,g)^\flat \triangleq K\,(g^\flat)$$

$$(g_1, \ldots, g_k)^\flat \triangleq (g_1{}^\flat, \ldots, g_k{}^\flat).$$

It is easy to show that $g \in Tree_\Sigma(T)$ implies $g^\flat \in Tree_{\Sigma^\flat}(T^\flat)$ for any ground tree g. Regarding α-equivalence, if $g \in Tree_\Sigma(T)$ then $[g^\flat]_\alpha = \{g^\flat\}$ and $[g^\flat]_\alpha \in \alpha\text{-}Tree_{\Sigma^\flat}(T^\flat)$. Furthermore, if $g_1 =_\alpha g_2{:}T$ then $g_1{}^\flat = g_2{}^\flat$, since the first-order reduction process erases all names. This property means that we can define the first-order reduction of α-trees as follows: $[g]_\alpha{}^\flat = [g^\flat]_\alpha = \{g^\flat\}$. In turn, this allows us to define a first-order reduction operation on α-tree valuations.

Definition 5.13 (Reducing α-tree valuations)**.** If $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ then we write $V^\flat$ for the valuation which has $dom(V^\flat) = dom(\Delta^\flat)$ (and hence $dom(V^\flat) \subseteq dom(V)$) and is such that $V^\flat(x) = (V(x))^\flat$ for all $x \in dom(V^\flat)$. It is straightforward to show that if $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ then $V^\flat \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)$.

We will use W to range over valuations in $\alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)$ if the starting α-tree valuation in $\alpha\text{-}Tree_\Sigma(\Delta)$ is irrelevant. Now, a key task is to show that satisfaction is preserved by the process of reduction to first-order form.

Lemma 5.14 (Reduction and valuation)**.** *For any Σ, V, Δ, t and T, if $\Delta \vdash_\Sigma t{:}T$ and $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ then $([\![t]\!]_V)^\flat = [\![t^\flat_\Delta]\!]_{V^\flat}$.*

Proof. The proof is by induction on the structure of t. In the base case where t is a variable which is not of name sort, we use the defining properties of $V^\flat$. $\qquad\square$

Lemma 5.15 (Reduction and satisfaction)**.** *For any Σ, Δ, $\overline{c}$ and V, if $\Delta \vdash_\Sigma \overline{c}$ ok and $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ then $V \models \overline{c}$ implies $V^\flat \models \overline{c}^\flat_\Delta$.*

Proof. Since Definition 5.11 discards all freshness constraints in $\overline{c}$ and translates all freshness constraints, it suffices to show that, for any equality constraint $(t_1 = t_2) \in \overline{c}$, if $[\![t_1]\!]_V = [\![t_2]\!]_V$ then $[\![t_1{}^\flat_\Delta]\!]_{V^\flat} = [\![t_2{}^\flat_\Delta]\!]_{V^\flat}$. If we assume that $[\![t_1]\!]_V = [\![t_2]\!]_V$ then $([\![t_1]\!]_V)^\flat = ([\![t_2]\!]_V)^\flat$, and by Lemma 5.14 we get that $[\![t_1{}^\flat_\Delta]\!]_{V^\flat} = [\![t_2{}^\flat_\Delta]\!]_{V^\flat}$, as required. $\qquad\square$

We can now prove an important result about the satisfaction of non-permutative constraint problems and state its corollary.

Theorem 5.16 (Reduction and satisfiability)**.** *For any Σ, Δ and $\overline{c}$, if $\emptyset \vdash_\Sigma \exists\Delta(\overline{c})$ ok then $\models \exists\Delta(\overline{c})$ implies $\models (\exists\Delta(\overline{c}))^\flat$.*

Proof. If $\models \exists\Delta(\overline{c})$ then there exists a valuation $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ such that $V \models \overline{c}$. By Lemma 5.15 it follows that $V^\flat \models \overline{c}^\flat_\Delta$, where $V^\flat \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)$. Thus we have $\models (\exists\Delta(\overline{c}))^\flat$, as required. $\qquad\square$

Corollary 5.17 (Reduction and unsatisfiability)**.** *For any Σ, Δ and $\overline{c}$, if $\emptyset \vdash_\Sigma \exists\Delta(\overline{c})$ ok then $\not\models (\exists\Delta(\overline{c}))^\flat$ implies $\not\models \exists\Delta(\overline{c})$.* $\qquad\square$

Corollary 5.17 tells us that if the first-order reduction $(\exists \Delta(\overline{c}))^{\flat}$ is *unsatisfiable* then the original problem $\exists \Delta(\overline{c})$ is unsatisfiable. This is one of the properties that we require of a correct termination check for non-permutative constraint problems. It just remains to show that if the first-order reduction $(\exists \Delta(\overline{c}))^{\flat}$ is *satisfiable* then $\exists \Delta(\overline{c})$ is strongly normalising.

We begin by showing that satisfaction of reduced constraint problems is preserved by substitution, in the following sense.

Lemma 5.18 (Satisfaction and substitution). *For any Σ, Δ, $\overline{c}$, x, t and W, suppose that $\Delta \vdash_{\Sigma} \overline{c}$ ok and $\Delta \vdash_{\Sigma} x = t$ ok and $W \in \alpha\text{-}Tree_{\Sigma^{\flat}}(\Delta^{\flat})$. Then, if $W \models \overline{c}_{\Delta}^{\flat}$ and $W \models x_{\Delta}^{\flat} = t_{\Delta}^{\flat}$ then $W \models (\overline{c}[t/x])_{\Delta}^{\flat}$.*

Proof. If $W \models x_{\Delta}^{\flat} = t_{\Delta}^{\flat}$ then we know that $[\![x_{\Delta}^{\flat}]\!]_W = [\![t_{\Delta}^{\flat}]\!]_W$. By Definition 5.11 it suffices to prove that a similar substitution property holds for all equality constraints $(t_1 = t_2) \in \overline{c}$. We assume that $\Delta \vdash_{\Sigma} t_1 : T$ and $\Delta \vdash_{\Sigma} t_2 : T$ (for some T) and that $\Delta \vdash_{\Sigma} x = t$ ok and $W \in \alpha\text{-}Tree_{\Sigma^{\flat}}(\Delta^{\flat})$, and show that $[\![t_1{}_{\Delta}^{\flat}]\!]_W = [\![t_2{}_{\Delta}^{\flat}]\!]_W$ and $[\![x_{\Delta}^{\flat}]\!]_W = [\![t_{\Delta}^{\flat}]\!]_W$ imply $[\![(t_1[t/x])_{\Delta}^{\flat}]\!]_W = [\![(t_2[t/x])_{\Delta}^{\flat}]\!]_W$. We now perform a case split: if T is a name sort, the result follows from the fact that $t'_{\Delta}^{\flat} = (t'[t/x])_{\Delta}^{\flat}$ if t and x are both of name sort; otherwise it follows from Lemma 3.7 and the fact that $(t'[t/x])_{\Delta}^{\flat} = t'_{\Delta}^{\flat}[t_{\Delta}^{\flat}/x]$. $\square$

We can now show that solutions to reduced problems are preserved by reduction of the original problem.

Lemma 5.19 (Preservation of reduced solutions). *If $\emptyset \vdash_{\Sigma} \exists \Delta(\overline{c})$ ok and $W \in \alpha\text{-}Tree_{\Sigma^{\flat}}(\Delta^{\flat})$ and $W \models \overline{c}_{\Delta}^{\flat}$ and $\exists \Delta(\overline{c}) \longrightarrow \exists \Delta'(\overline{c}')$ then there exists a valuation $W' \in \alpha\text{-}Tree_{\Sigma^{\flat}}(\Delta'^{\flat})$ which agrees with W on $dom(\Delta^{\flat})$ and is such that $W' \models \overline{c}'_{\Delta'}^{\flat}$.*

Proof. We assume that $\emptyset \vdash_{\Sigma} \exists \Delta(\overline{c})$ ok and $W \in \alpha\text{-}Tree_{\Sigma^{\flat}}(\Delta^{\flat})$ and $W \models \overline{c}_{\Delta}^{\flat}$ and $\exists \Delta(\overline{c}) \longrightarrow \exists \Delta'(\overline{c}')$ all hold, and proceed by case analysis on the constraint transformation rule used to derive $\exists \Delta(\overline{c}) \longrightarrow \exists \Delta'(\overline{c}')$.

The cases for rules (F1)–(F3) and (F5) are straightforward since $\overline{c}'_{\Delta'}^{\flat} = \overline{c}_{\Delta}^{\flat}$. In the case for rule (E1) the new constraints in $\overline{c}'_{\Delta'}^{\flat}$ are all trivially satisfied. The cases for rules (E2) and (E3) follow directly from the semantics of non-permutative constraints, defined in terms of $[\![t]\!]_V$. The case for (E5) follows because $\overline{c}'_{\Delta'}^{\flat}$ is obtained from $\overline{c}_{\Delta}^{\flat}$ simply by deleting a constraint. In the cases for (F4) and (E4) the additional constraints in $\overline{c}'_{\Delta'}^{\flat}$ are all simple equality constraints in involving tuples and unit, which are trivially satisfied. The cases for (E6) and (E7) both rely on Lemma 5.18 to deal with substitution. In the case of (E7) we also use the fact that the variables in the "patterns" generated by the narrowing rules from Figure 4 always use fresh variables. This allows us to safely extend W to produce a larger valuation W'. $\square$

In order to prove a termination result we will need to define some kind of size metric on constraint problems, by interpreting them into a well-founded set.

Definition 5.20 (Sizes of ground trees). Let $\mathbb{N}_+$ be the set $\{n \in \mathbb{N} \mid n \geq 1\}$. Then we define a size function $size(g)$ which maps from ground trees into $\mathbb{N}_+$, as follows.

$$size(n) \triangleq 1 \qquad size(K\,g) \triangleq 1 + size(g) \qquad size(\texttt{<}x\texttt{>}g) \triangleq 2 + size(g)$$

$$size(\texttt{()}) \triangleq 1 \qquad size((g_1, \ldots, g_k)) \triangleq 1 + \sum_{i \in \{1,\ldots,k\}} size(g_i).$$

Since $g =_\alpha g'$ implies that $size(g) = size(g')$, the above definition induces a well-defined size function on α-trees: $size([g]_\alpha) = size(g)$ for some/any representative g of the α-equivalence class. Furthermore, $size(g) \geq 1$ for all g.

Definition 5.21 (Sizes of terms and atomic constraints)**.** If $\Delta \vdash_\Sigma t{:}T$ (for some type T) then we define a function $\lceil t \rceil : (\alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)) \to \mathbb{N}_+$ as follows.

$$\lceil x \rceil(W) \triangleq \begin{cases} 1 & \text{if } \Delta(x) = N, \text{ for some } N \\ size(W(x)) & \text{otherwise} \end{cases} \qquad \lceil \texttt{<}x\texttt{>}t \rceil(W) \triangleq 2 + \lceil t \rceil(W)$$

$$\lceil () \rceil(W) \triangleq 1 \qquad \lceil K\,t \rceil(\triangleq)1 + \lceil t \rceil(W) \qquad \lceil (t_1, \ldots, t_k) \rceil(W) \triangleq 1 + \sum_{i \in \{1,\ldots,k\}} \lceil t_i \rceil(W).$$

We now extend this definition to atomic constraints—if $\Delta \vdash_\Sigma c\ ok$ we define a function $\lceil c \rceil : (\alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)) \to \mathbb{N}_+$, as follows.

$$\lceil t = t' \rceil(W) \triangleq \lceil t \rceil(W) + \lceil t' \rceil(W) \qquad\qquad \lceil x \mathbin{\#} t' \rceil(W) \triangleq \lceil t' \rceil(W).$$

Note that $\lceil t \rceil(W) \geq 1$ and $\lceil c \rceil(W) \geq 1$ both always hold.

Lemma 5.22 (Equality constraints and sizes)**.** *For all Σ, Δ, t, t' and W, if $\Delta \vdash_\Sigma t = t'\ ok$ and $W \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)$ and $W \models t^\flat_\Delta = t'^\flat_\Delta$ then $\lceil t \rceil(W) = \lceil t' \rceil(W)$.*

Proof. If $W \models t^\flat_\Delta = t'^\flat_\Delta$ then $[\![t^\flat_\Delta]\!]_W = [\![t'^\flat_\Delta]\!]_W$. Hence $size([\![t^\flat_\Delta]\!]_W) = size([\![t'^\flat_\Delta]\!]_W)$. It is straightforward to show that $size([\![t^\flat_\Delta]\!]_W) = \lceil t \rceil(W)$ (the crucial cases are for names and abstractions) and hence we get that $\lceil t \rceil(W) = \lceil t' \rceil(W)$, as required. $\qquad\square$

Definition 5.23 (Solved variables)**.** We say that a variable x is *solved* in $\exists\Delta(\overline{c})$ iff there is precisely one occurrence of x in $\overline{c}$, where that occurrence is in a constraint of the form $x = t$ or $t = x$. We say that a variable is *unsolved* when it is not solved.

Definition 5.24 (Measure on constraint problems)**.** Write $\mathcal{M}$ for the set of finite multisets of elements of $\mathbb{N}_+$ and write $\{\!\{f(x) \mid x \in S, P(x)\}\!\}$ for the *multiset* of values $f(x)$ where $x \in S$ and x satisfies the property $P(x)$. Now, we begin by defining two intermediate measure functions from constraint problems $\exists\Delta(\overline{c})$ into $\mathcal{M}$, each parameterised by a valuation $W \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)$.

$$\begin{aligned} \mu_1(W)(\exists\Delta(\overline{c})) &\triangleq \{\!\{ \lceil x \rceil(W) \mid x \in dom(\Delta), x \text{ is } unsolved \}\!\} \\ \mu_2(W)(\exists\Delta(\overline{c})) &\triangleq \{\!\{ \lceil c \rceil(W) \mid c \in \overline{c} \}\!\} \end{aligned}$$

We now define a measure function μ on constraint problems in terms of μ_1, μ_2 and a valuation $W \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)$.

$$\mu(W)(\exists\Delta(\overline{c})) \triangleq (\mu_1(W)(\exists\Delta(\overline{c})), \mu_2(W)(\exists\Delta(\overline{c})))$$

Since μ_1 and μ_2 are functions into $\mathcal{M}$ it follows that μ is a function into $\mathcal{M} \times \mathcal{M}$. There is a well-founded ordering $\prec_\mathcal{M}$ on $\mathcal{M}$ induced by the usual well-founded ordering $<$ on natural numbers, via the multiset ordering construction from [DM79]. From the lexicographic product of $\prec_\mathcal{M}$ with itself we derive a well-founded ordering $\prec_{\mathcal{M} \times \mathcal{M}}$ on $\mathcal{M} \times \mathcal{M}$, which we will use to provide a well-founded ordering on the results of the μ function.

We now have the necessary tools to show that transformation of any constraint problem whose first-order reduction is satisfiable will terminate. The proof uses the standard strategy

of interpreting constraint problems in a set equipped with a well-founded ordering, using the measure function defined above.

Theorem 5.25 (Termination). *If $\emptyset \vdash_\Sigma \exists\Delta(\overline{c})$ ok holds and $\models (\exists\Delta(\overline{c}))^\flat$ then there is no infinite sequence of $\longrightarrow$ transformations starting from $\exists\Delta(\overline{c})$.*

Proof. We will proceed by showing that

$$\exists\Delta(\overline{c}) \longrightarrow \exists\Delta'(\overline{c}') \implies \mu(W')(\exists\Delta'(\overline{c}')) \prec_{\mathcal{M}\times\mathcal{M}} \mu(W)(\exists\Delta(\overline{c})) \tag{5.1}$$

holds, for any $W \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)$ such that $W \models \overline{c}_\Delta^\flat$ and any $W' \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta'^\flat)$ which agrees with W on $dom(\Delta^\flat)$ and is such that $W' \models \overline{c}_{\Delta'}^\flat$.

This suffices to prove termination because (by Lemma 5.19) solutions to reduced problems are preserved by the transformation rules. By assumption, if $\models (\exists\Delta(\overline{c}))^\flat$ holds then there exists some $W \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)$ such that $W \models \overline{c}_\Delta^\flat$. Therefore, for any solution W of the reduced problem $(\exists\Delta(\overline{c}))^\flat$ we get a family of derived solutions W' for the reduced problem $(\exists\Delta'(\overline{c}'))^\flat$, each of which produces a strictly smaller value for $\mu(W')(\exists\Delta'(\overline{c}'))$ in the well-founded ordering $\prec_{\mathcal{M}\times\mathcal{M}}$. If we repeat this argument along the transformation sequence it follows that the chain must eventually terminate.

To prove that (5.1) holds, we proceed by case analysis on the rule used to derive $\exists\Delta(\overline{c}) \longrightarrow \exists\Delta'(\overline{c}')$. We will present the case for rule (E7) in full, as it is the most involved.

(E7): We assume that $\overline{c} = (\texttt{<}x_{1..k}\texttt{>}x\,\texttt{=}\,\texttt{<}y_{1..k}\texttt{>}t)$ & $\overline{c}^*$ and that $W \models \overline{c}_\Delta^\flat$, where $\overline{c}_\Delta^\flat = (\texttt{<}x_{1..k}\texttt{>}x\,\texttt{=}\,\texttt{<}y_{1..k}\texttt{>}t)_\Delta^\flat$ & $\overline{c}_\Delta^{*\flat}$ and $W \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta^\flat)$. We also assume that $\overline{c}' = (x\,\texttt{=}\,t^*)$ & $(\texttt{<}x_{1..k}\texttt{>}x\,\texttt{=}\,\texttt{<}y_{1..k}\texttt{>}t)$ & $(\overline{c}^*[t^*/x])$ and $\Delta' = \Delta, \Delta^*$ both hold, where $[\Delta;t] \longmapsto [\Delta^*;t^*]$ holds. Furthermore, we let $W' \in \alpha\text{-}Tree_{\Sigma^\flat}(\Delta'^\flat)$ be such that $W' \models \overline{c}_{\Delta'}^\flat$ holds and such that W and W' agree on $dom(\Delta^\flat)$. By assumption, $x \notin vars(t)$ and since $dom(\Delta^*) \cap dom(\Delta) = \emptyset$ by definition, it follows that rule (E7) changes x from unsolved to solved and replaced it with unsolved variables $dom(\Delta^*)$. Thus we must show that $\lceil z\rceil(W') < \lceil x\rceil(W)$ for all $z \in dom(\Delta^*)$.

Now, since $W' \models \overline{c}_{\Delta'}^\flat$ we know that $W' \models x_{\Delta'}^\flat \,\texttt{=}\, t^{*\flat}_{\Delta'}$ holds, and by Lemma 5.22 we get that $\lceil x\rceil(W') = \lceil t^*\rceil(W')$ holds. Since t is not a variable we know that $\Delta(x)$ cannot be a name sort and hence $x \in dom(\Delta^\flat)$. Thus we get that $W(x) = W'(x)$ and it follows that $\lceil x\rceil(W) = \lceil x\rceil(W')$. Therefore we know that $\lceil x\rceil(W) = \lceil t^*\rceil(W')$. It is easy to see that $\lceil z\rceil(W') < \lceil t^*\rceil(W')$ holds for all $z \in dom(\Delta^*)$, from which it follows that $\lceil z\rceil(W') < \lceil x\rceil(W)$ holds for all $z \in dom(\Delta^*)$. Hence we have shown that $\mu_1(W')(\exists\Delta'(\overline{c}')) \prec_{\mathcal{M}} \mu_1(W)(\exists\Delta(\overline{c}))$ holds, from which it follows that $\mu(W')(\exists\Delta'(\overline{c}')) \prec_{\mathcal{M}\times\mathcal{M}} \mu(W)(\exists\Delta(\overline{c}))$, as required.

In the case for rule (E6), one unsolved variable becomes solved. In each of the remaining cases, there are no more unsolved variables in $dom(\Delta')$ than in $dom(\Delta)$. Furthermore, $\mu_2(W')(\exists\Delta'(\overline{c}'))$ is formed from $\mu_2(W)(\exists\Delta(\overline{c}))$ by removing $\lceil c^*\rceil(W)$ (for some c^*) and replacing it with zero or finitely many elements $\lceil c_i^*\rceil(W')$, where $i \in \{1,\ldots,k\}$ for some k, and where $W' = W$ since $\Delta' = \Delta$. Now we get that $\lceil c_i^*\rceil(W') < \lceil c^*\rceil(W)$ for all $i \in \{1,\ldots,k\}$. Thus we get $\mu_2(W')(\exists\Delta'(\overline{c}')) \prec_{\mathcal{M}} \mu_2(W)(\exists\Delta(\overline{c}))$, and it follows that $\mu(W')(\exists\Delta'(\overline{c}')) \prec_{\mathcal{M}\times\mathcal{M}} \mu(W)(\exists\Delta(\overline{c}))$, as required. $\square$

Thus we have shown that *(1)* if the reduced problem $(\exists\Delta(\overline{c}))^\flat$ is *satisfiable* then the original problem $\exists\Delta(\overline{c})$ is strongly normalising, and *(2)* if $(\exists\Delta(\overline{c}))^\flat$ is *unsatisfiable* then

$\exists\Delta(\overline{c})$ is unsatisfiable. These properties mean that satisfiability of reduced problems is a suitable decidable check for detecting may-divergent constraint problems.

5.3. **Soundness and completeness of the algorithm.** In this section we prove correctness of a decision procedure for non-permutative nominal constraint problems which uses the constraint transformation rules from Figure 3. We proceed by relating the syntactic forms of constraint problems in $\longrightarrow$-normal form to their satisfiability.

Definition 5.26 (Solved constraint problems). A constraint problem $\exists\Delta(\overline{c})$ is *solved* iff all constraints in $\overline{c}$ have one of the following forms.

(1) $x \mathrel{\#} y$, where x and y are distinct variables and either $\Delta(x) = \Delta(y)$ or $\Delta(y)$ is not a name sort;
(2) $x = t$, where $x \notin vars(t)$ and x does not appear elsewhere in $\overline{c}$;
(3) $<x_{1..k}>x = <y_{1..k}>y$, where $k > 0$ and $\Delta(x)(= \Delta(y))$ is not a name sort and x and y are distinct variables; or
(4) $<x_{1..k}>x = <y_{1..k}>x$, where $k > 0$ and $\Delta(x)$ is not a name sort.

Lemma 5.27. *Any* solved *constraint problem* $\exists\Delta(\overline{c})$ *is also* terminal.

Proof. By cases on the possible constraints that may appear within a solved constraint problem, according to Definition 5.26. $\qquad\square$

The relationship between terminal and solved constraints and their satisfiability is crucial to the correctness of our algorithm. We now show that once a problem has been reduced as far as possible using $\longrightarrow$ we can determine whether it is satisfiable by examining its syntax.

Lemma 5.28 (Terminal constraints and satisfiability). *Let* $\exists\Delta(\overline{c})$ *be a* terminal *constraint problem such that* $\emptyset \vdash_{\Sigma} \exists\Delta(\overline{c})$ *ok. Then* $\exists\Delta(\overline{c})$ *is satisfiable iff it is* solved.

Proof. We assume that $\emptyset \vdash_{\Sigma} \exists\Delta(\overline{c})$ ok. By inspection of the constraint transformation rules, the possible forms of constraint in a terminal constraint problem consist of the possibilities presented in Definition 5.26 as well as the following:

(5) $x \mathrel{\#} x$.
(6) $<x_{1..k}>K\, t = <y_{1..k}>K'\, t'$, where $K \neq K'$.
(7) $x = t$, where $x \in vars(t)$ and t is not x.
(8) $<x_{1..k}>x = <y_{1..k}>t$, where $k > 0$ and $x \in vars(t)$.

In particular, an equality constraint between two terms which have different numbers of outermost nested abstractions is not terminal, as it can be reduced by narrowing using rule (E7). It suffices to show that any single constraint conforming to possibilities 5–8 is unsatisfiable, and that any solved constraint problem is satisfiable. We prove these below.

Any constraint of the form 5–8 is unsatisfiable: Constraints of form 5 are not satisfiable because a name cannot be fresh for itself, and constraints of form 6 are not satisfiable because the constructors do not match. Finally, constraints of forms 7 and 8 are not satisfiable because the occurs check fails.

Any solved constraint problem is satisfiable: For a solved constraint problem $\exists\Delta(\overline{c})$ we will construct a satisfying valuation V. We write $\overline{c}_i$ for the partition of $\overline{c}$ where the constraints are all of the form $i \in \{1, 2, 3, 4\}$.

We observe that we can form a satisfying valuation $V_{3,4}$ for $\overline{c}_3 \uplus \overline{c}_4$ because the variables x and y in constraints of form 3 and x in form 4 cannot be of name sort and hence cannot

coincide with any of the abstracted variables. Therefore we can simply instantiate the abstracted variables distinctly (i.e. avoiding aliasing) and instantiate the variables within the nesting to avoid the abstracted variables and satisfy the appropriate constraints.

Now we note that if $(x \mathbin{\#} y) \in \overline{c}_1$ and $x, y \in vars(\overline{c}_3 \uplus \overline{c}_4)$ then $V_{3,4} \models x \mathbin{\#} y$ by construction. Therefore we can extend $V_{3,4}$ with additional mappings which ensure that all freshnesss in $\overline{c}_1$ are satisfied, to produce a valuation $V_{1,3,4}$ which satisfies $\overline{c}_1 \uplus \overline{c}_3 \uplus \overline{c}_4$.

Finally it is always possible to extend $V_{1,3,4}$ to a satisfying valuation V for the entire problem $\overline{c}$. We begin by providing an arbitrary instantiation for any variable $z \in vars(\overline{c}_2)$ which only appears on the right-hand side of constraints in $\overline{c}_2$ and which has not already been instantiated. This just leaves the variables x which appear on the left-hand side of the constraints in $\overline{c}_2$. By assumption on solved constraints these variables cannot appear elsewhere in $\overline{c}$ and hence cannot have been instantiated already. Hence we are free to choose instantiations for these variables which satisfy $\overline{c}_2$. Thus we get that $V \models \overline{c}$, as required. $\qquad\square$

With these results under our belt we can begin to examine the correctness of the constraint transformation algorithm. We begin by proving a soundness result: if a constraint problem has a solved $\longrightarrow$-normal form then it is satisfiable.

Theorem 5.29 (Soundness of transformation algorithm)**.** *For any constraint problem $\exists \Delta(\overline{c})$ where $\emptyset \vdash_\Sigma \exists \Delta(\overline{c})$ ok holds, if there exists a $\longrightarrow$-normal form $\exists \Delta'(\overline{c}')$ of $\exists \Delta(\overline{c})$ which is solved then $\exists \Delta(\overline{c})$ is satisfiable.*

Proof. Suppose that $\exists \Delta'(\overline{c}')$ is a $\longrightarrow$-normal form of $\exists \Delta(\overline{c})$. If $\exists \Delta'(\overline{c}')$ is solved then $\exists \Delta'(\overline{c}')$ is satisfiable by Lemma 5.28, i.e. there exists some $V' \in \alpha\text{-}Tree_\Sigma(\Delta')$ such that $V' \models \overline{c}'$. Finally, by Theorem 5.2 we get that $V \models \overline{c}$ holds (where V is the restriction of V' to $dom(\Delta)$) and hence that $\exists \Delta(\overline{c})$ is satisfiable, as required. $\qquad\square$

We now prove a partial completeness result which is *not quite* the converse of Theorem 5.29 because it only applies to strongly normalising constraint problems. The assumption that $\exists \Delta(\overline{c})$ is strongly normalising is needed to ensure that it has a $\longrightarrow$-normal form, which we then show is satisfied by V.

Theorem 5.30 (Partial completeness of transformation algorithm)**.** *Let $\exists \Delta(\overline{c})$ be a strongly normalising constraint problem such that $\emptyset \vdash_\Sigma \exists \Delta(\overline{c})$ ok holds. If $\exists \Delta(\overline{c})$ is satisfiable then there exists a $\longrightarrow$-normal form $\exists \Delta'(\overline{c}')$ of $\exists \Delta(\overline{c})$ which is solved.*

Proof. If $\exists \Delta(\overline{c})$ is satisfiable then there exists a valuation $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ such that $V \models \overline{c}$ holds. Since $\exists \Delta(\overline{c})$ is strongly normalising we know that every transformation sequence eventually terminates. Then, by Theorem 5.6 we know that there is some sequence of transformations from $\exists \Delta(\overline{c})$ which terminate at a problem $\exists \Delta'(\overline{c}')$ such that $V' \models \overline{c}'$ holds, where V' extends V to $dom(\Delta')$. Finally, by Lemma 5.28 it follows that the $\longrightarrow$-normal form $\exists \Delta'(\overline{c}')$ is solved, as required. $\qquad\square$

Now we use the termination checking procedure from Section 5.2 to close the gap in Theorem 5.30, giving us a correct decision procedure for NonPermSat.

Theorem 5.31 (Correct decision procedure)**.** *There exists a correct decision procedure for NonPermSat based on the constraint transformation rules from Figure 3.*

Proof. Using the termination check from Theorem 5.25 we can dismiss may-divergent constraint problems as unsatisfiable without having to rewrite them using the rules from Figure 3. This allows us to restrict our attention to strongly normalising constraint problems

$\exists \Delta(\overline{c})$, for which we can compute the finite set $\mathcal{S}$ of $\longrightarrow$-normal forms, that is, the set $\mathcal{S} = \{\exists \Delta'(\overline{c'}) \mid \exists \Delta(\overline{c}) \longrightarrow \cdots \longrightarrow \exists \Delta'(\overline{c'}) \not\longrightarrow\}$, in finite time. By Theorem 5.29 and Theorem 5.30, the constraint problem $\exists \Delta(\overline{c})$ is satisfiable precisely when there exists a solved constraint problem in $\mathcal{S}$, which is a decidable property of the syntax of $\mathcal{S}$. $\square$

The algorithm presented in this section decides satisfiability of non-permutative nominal constraint problems: it does not enumerate solutions. Recalling Definition 5.26 we see that solved constraint problems may contain constraints of the form $\texttt{<}x_{1..k}\texttt{>}x = \texttt{<}y_{1..k}\texttt{>}y$ or $\texttt{<}x_{1..k}\texttt{>}x\texttt{=}\texttt{<}y_{1..k}\texttt{>}x$, where $k > 0$ and where x and y are not variables of name sort. Constraints of these forms may be satisfied by infinitely many different ground valuations, as x and y may range over some recursive datatype. Since these may be the only occurrences of x and y in the constraint problem, it follows that a satisfiable non-permutative nominal constraint problem may have infinitely many satisfying ground valuations. However, Theorem 5.31 demonstrates that our algorithm only needs to check the finite number of elements of $\mathcal{S}$ to ascertain the patterns that all satisfying ground valuations must follow.

6. Encoding name-name equivariant unification

In this section we present a reduction of equivariant unification between name-terms into non-permutative nominal constraints. This will make explicit the link between equivariant unification and non-permutative nominal constraints alluded to in Section 3. It is sufficient for our purposes to consider equivariant unification between terms of name sort because that sub-problem of equivariant unification is known to be **NP**-complete [Che04, Section 7.1]. We recall the grammar of equivariant unification name-terms from Section 2.3.

Vertices	v, w	$::=$	n	(name)
		$\mid$	A	(name variable)
Name-terms	a, b	$::=$	$\Pi \cdot v$	(suspended permutation)
Permutation-terms	Π	$::=$	ι	(identity)
		$\mid$	$(a\,b)$	(swap)
		$\mid$	Q	(permutation variable)

We also refer back to Section 2.3 for the semantics of name-name equivariant unification problems. To simplify our presentation we assume (without loss of generality) that all sub-terms of the forms $\Pi^{-1} \cdot v$ and $(\Pi \circ \Pi') \cdot v$ have been expanded away by the addition of fresh name variables (to represent intermediate values) and additional equality constraints. This process is described as "phase two" of the equivariant unification algorithm [Che10, Section 4.2.2]. For example, the name-term $\Pi^{-1} \cdot v$ can be translated to the fresh name variable A, given the constraint that $\Pi \cdot A = v$ (where Π has been recursively expanded out in the same way). Furthermore, we assume that all names are of a single name sort N.

6.1. **Defining the encoding.** In order to encode equivariant unification we must use different collections of variables to represent names, name variables and permutation variables. Thus we assume that the countably infinite set of variables Var is partitioned into finitely many disjoint, countably infinite subsets Var_{Name}, Var_{Nvar}, and $Var_{Q_1}, \ldots, Var_{Q_k}$, where $Q_1, \ldots, Q_k$ is the finite set of permutation variables which appear in the problem of interest. These will be used to represent the permutative names, the name variables and the results of applying the unknown permutations $Q_1, \ldots, Q_k$ to other name-terms, respectively. For

the translation we will also need additional variables to store intermediate values—for these will use another disjoint, countably infinite set of variables Var_{Temp}. We fix bijections into these sets, as follows:

- a bijection from $Name$ to Var_{Name}, where $x_n \in Var_{Name}$ stands for $n \in Name$;
- a bijection from $Nvar$ to Var_{Nvar}, where $x_A \in Var_{Nvar}$ stands for $A \in Nvar$; and
- for each permutation variable Q a bijection from $Name \uplus Nvar$ to Var_Q, where $x_{(Q,v)} \in Var_Q$ stands for $Q \cdot v$ for $v \in (Name \uplus Nvar)$.

For the translation we fix a trivial nominal signature Σ where $\mathbb{C}_\Sigma = \mathbb{D}_\Sigma = \emptyset$ and where $\mathbb{N}_\Sigma = \{N\}$ for some fixed name type N. Given finite sets $\overline{n}$, $\overline{A}$ and $\overline{Q}$ and a finite set $\overline{x} \subset Var_{Temp}$ we define a typing environment for the corresponding non-permutative variables:

$$
\begin{aligned}
\Delta_{(\overline{n},\overline{A},\overline{Q},\overline{x})} \quad &\triangleq \quad \{x_n : N \mid n \in \overline{n}\} \uplus \{x_A : N \mid A \in \overline{A}\} \uplus \\
&\quad \{x_{(Q,v)} : N \mid Q \in \overline{Q} \wedge v \in (\overline{n} \uplus \overline{A})\} \uplus \{x : N \mid x \in \overline{x}\}
\end{aligned}
$$

where N is the single name sort from Σ. Note that $\Delta_{(\overline{n}',\overline{A}',\overline{Q}',\overline{x}')} \supseteq \Delta_{(\overline{n},\overline{A},\overline{Q},\overline{x})}$ holds if $\overline{n}' \supseteq \overline{n}$, $\overline{A}' \supseteq \overline{A}$, $\overline{Q}' \supseteq \overline{Q}$ and $\overline{x}' \supseteq \overline{x}$ all hold. The following rules specify the translation of an equivariant unification name-term a into a variable x and associated constraints $\overline{c}'$ in NPNAS, which involve the new variables $\overline{x}'$. We write this as $\mathtt{tr}(a)_{\overline{x}} = \exists \overline{x}'(x \text{ where } \overline{c}')$.

$$
\overline{\mathtt{tr}(\iota \cdot v)_{\overline{x}} = \exists \emptyset (x_v \text{ where } \emptyset)} \qquad\qquad \overline{\mathtt{tr}(Q \cdot v)_{\overline{x}} = \exists \emptyset (x_{(Q,v)} \text{ where } \emptyset)}
$$

$$
\frac{\mathtt{tr}(a)_{\overline{x}} = \exists \overline{x}_1 (x_a \text{ where } \overline{c}_1) \qquad \mathtt{tr}(b)_{(\overline{x} \uplus \overline{x}_a)} = \exists \overline{x}_2 (x_b \text{ where } \overline{c}_2) \qquad z \notin (\overline{x} \uplus \overline{x}_1 \uplus \overline{x}_2)}{\mathtt{tr}((a\, b) \cdot v)_{\overline{x}} = \exists (\overline{x}_1 \uplus \overline{x}_2 \uplus \{z\})(z \text{ where } \overline{c}_1 \, \& \, \overline{c}_2 \, \& \, (\texttt{<}x_a\texttt{><}x_b\texttt{>}z \texttt{ = <}x_b\texttt{><}x_a\texttt{>}x_v))}
$$

Here, and throughout, $\overline{x} \subset Var_{Temp}$ is a finite set of temporary variables which have already been used and must be *avoided* in the rest of the translation. The following result states the semantics of the "swapping" construction.

Lemma 6.1 (Swapping constraints). *Suppose that $x, y, u, w \in dom(V)$, $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ and $\Delta(x) = \Delta(y) = N$ for some name sort N, and where $V(x) = \{n\}$ and $V(y) = \{n'\}$, for some n, n'. Then we get that $V \models \texttt{<}x\texttt{><}y\texttt{>}u \texttt{ = <}y\texttt{><}x\texttt{>}w$ iff $V(u) = (n\, n') \cdot V(w)$.* $\qquad\square$

The NPNAS translation of an equivariant unification constraint can now be defined straightforwardly. We write $\mathtt{tr}(c)_{\overline{x}} = \exists \overline{x}'(\overline{c}')$ if the equivariant unification constraint c is translated to the NPNAS problem $\overline{c}'$, involving new variables $\overline{x}'$ and avoiding $\overline{x}$.

$$
\frac{\mathtt{tr}(a)_{\overline{x}} = \exists \overline{x}_1 (x_a \text{ where } \overline{c}_1) \qquad \mathtt{tr}(b)_{(\overline{x} \uplus \overline{x}_a)} = \exists \overline{x}_2 (x_b \text{ where } \overline{c}_2)}{\mathtt{tr}(a \approx b)_{\overline{x}} = \exists (\overline{x}_1 \uplus \overline{x}_2)(\overline{c}_1 \, \& \, \overline{c}_2 \, \& \, x_a \texttt{ = } x_b)}
$$

$$
\frac{\mathtt{tr}(a)_{\overline{x}} = \exists \overline{x}_1 (x_a \text{ where } \overline{c}_1) \qquad \mathtt{tr}(b)_{(\overline{x} \uplus \overline{x}_a)} = \exists \overline{x}_2 (x_b \text{ where } \overline{c}_2)}{\mathtt{tr}(a \,\#\, b)_{\overline{x}} = \exists (\overline{x}_1 \uplus \overline{x}_2)(\overline{c}_1 \, \& \, \overline{c}_2 \, \& \, x_a \,\#\, x_b)}
$$

In order to model permutative names and permutation variables using the standard non-permutative variables from Section 3, we must impose some additional consistency constraints on the variables to ensure that they reflect the correct semantics. In particular, we will want to express pairwise *distinctness* constraints between finite sets of name variables. For a finite set $\overline{x} = \{x_1, \ldots, x_k\}$ of variables of name sort, we write $\#_{\overline{x}}$ for the set of atomic freshness constraints $\{x_i \,\#\, x_j \mid 1 \le i < j \le k\}$.

Definition 6.2 (Consistency constraints)**.** Given finite sets $\overline{n}$, $\overline{A}$ and $\overline{Q}$ we write $\mathbb{C}(\overline{n}, \overline{A}, \overline{Q})$ for the *consistency constraints* over $\overline{n}$, $\overline{A}$ and $\overline{Q}$, which are defined as follows.

$$\mathbb{C}(\overline{n}, \overline{A}, \overline{Q}) \triangleq \#_{\{x_n \mid n \in \overline{n}\}} \,\&\, \{<x_v><x_{v'}>x_v = <x_{(Q,v)}><x_{(Q,v')}>x_{(Q,v)}$$
$$\mid Q \in \overline{Q} \wedge v \neq v' \wedge \{v, v'\} \subseteq (\overline{n} \uplus \overline{A})\}$$

Consistency constraints will be crucial in the proof that our encoding of equivariant unification is correct. The first part of $\mathbb{C}(\overline{n}, \overline{A}, \overline{Q})$ requires that the variables which represent the names in $\overline{n}$ should all be distinct—this encodes the fact that names n are permutative in equivariant unification. The following result states the semantics of the $<x_a><x_b>z = <x_b><x_a>x_v$ constraints used in the second part.

Lemma 6.3 (Bijection constraints)**.** *Suppose that $x, y, x', y' \in dom(V)$, $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ and $\Delta(x) = \Delta(y) = \Delta(x') = \Delta(y') = N$ for some name sort N. Then we get that $V \models <x><y>x = <x'><y'>x'$ iff $V(x) = V(y) \Longleftrightarrow V(x') = V(y')$.* $\qquad\square$

Thus the second part of $\mathbb{C}(\overline{n}, \overline{A}, \overline{Q})$ requires that all instantiations of the variables $x_{(Q,v)}$ (which represent the application of Q to v) respect the fact that Q denotes an unknown *bijection.*

We now have all the ingredients needed to define the NPNAS translation of a name-name equivariant unification problem S, using similar notation to above.

$$\overline{n} = names(S) \qquad \overline{A} = nvars(S) \qquad \overline{Q} = pvars(S) \qquad S \equiv \{c_1, \ldots, c_k\}$$
$$\frac{\texttt{tr}(c_1)_\emptyset = \exists \overline{x}_1(\overline{c}_1) \qquad \cdots \qquad \texttt{tr}(c_k)_{(\overline{x}_1 \uplus \cdots \uplus \overline{x}_{k-1})} = \exists \overline{x}_k(\overline{c}_k)}{\texttt{tr}(S) = \exists(\overline{x}_1 \uplus \cdots \uplus \overline{x}_k)(\overline{c}_1 \,\&\, \cdots \,\&\, \overline{c}_k \,\&\, \mathbb{C}(\overline{n}, \overline{A}, \overline{Q}))}$$

Lemma 6.4 (Typing lemma for problem translation)**.** *If $\overline{n}; \overline{A}; \overline{Q} \vdash S$ ok and $\texttt{tr}(S) = \exists \overline{x}(\overline{c})$ then $\Delta_{(\overline{n}, \overline{A}, \overline{Q}, \overline{x})} \vdash_\Sigma \overline{c}$ ok.* $\qquad\square$

6.2. **Correctness of the encoding.** In this section we prove that every satisfying ground valuation θ for a name-name equivariant unification problem can be translated into an NPNAS valuation which satisfies the corresponding NPNAS constraint problem, and vice versa. We begin with the translation of ground equivariant unification (EU) valuations.

Definition 6.5 (Translating EU ground valuations)**.** Given a ground valuation θ and a finite set $\overline{n}$ of names, write $V_{(\theta, \overline{n})}$ for the NPNAS valuation where

- $dom(V_{(\theta, \overline{n})}) = \{x_n \mid n \in \overline{n}\} \uplus \{x_A \mid A \in dom(\theta)\} \uplus \{x_{(Q,v)} \mid Q \in dom(\theta) \wedge v \in (\overline{n} \uplus \overline{A})\}$;
- $\forall x_n \in dom(V_{(\theta, \overline{n})})$. $V_{(\theta, \overline{n})}(x_n) = \{n\}$;
- $\forall x_A \in dom(V_{(\theta, \overline{n})})$. $V_{(\theta, \overline{n})}(x_A) = \{\theta(A)\}$; and
- $\forall x_{(Q,v)} \in dom(V_{(\theta, \overline{n})})$. $V_{(\theta, \overline{n})}(x_{(Q,v)}) = \{\theta(Q) \cdot \theta(v)\}$.

The definition of $V_{(\theta, \overline{n})}$ uses the partitions of *Var* described above to encode the different kinds of variables and names from equivariant unification. The additional $\overline{n}$ parameter is needed because the valuation in EU does not provide instantiations for names, whereas the valuation in NPNAS must provide instantiations for the variables corresponding to those names. Note that $V_{(\theta, \overline{n})} \in \alpha\text{-}Tree_\Sigma(\Delta_{(\overline{n}, \overline{A}, \overline{Q}, \emptyset)})$, where $\overline{A} = \{A \mid A \in dom(\theta)\}$ and $\overline{Q} = \{Q \mid Q \in dom(\theta)\}$. Recalling that a "vertex" v is either a name n or a name variable A, it is straightforward to show that, for any v, if $names(v) \subseteq \overline{n}$ and $namevars(v) \subseteq dom(\theta)$ then $V_{(\theta, \overline{n})}(x_v) = \{\theta(v)\}$. We now show that translated EU valuations satisfy the appropriate consistency constraints.

Lemma 6.6 (Translated valuations and consistency constraints). *If $\overline{A} = \{A \mid A \in dom(\theta)\}$ and $\overline{Q} = \{Q \mid Q \in dom(\theta)\}$ then $V_{(\theta,\overline{n})} \models \mathbb{C}(\overline{n}, \overline{A}, \overline{Q})$.*

Proof. We must show that *(1)* if $n \neq n'$ and $\{n, n'\} \subseteq \overline{n}$ then $V_{(\theta,\overline{n})}(x_n) \neq V_{(\theta,\overline{n})}(x_{n'})$; and *(2)* if $Q \in \overline{Q}$ and $\{v, v'\} \subseteq (\overline{n} \uplus \overline{A})$ then $V_{(\theta,\overline{n})} \models \text{<}x_v\text{><}x_{v'}\text{>}x_v = \text{<}x_{(Q,v)}\text{><}x_{(Q,v')}\text{>}x_{(Q,v)}$. In both cases we use the definition of $V_{(\theta,\overline{n})}$. For the second we furthermore rely on the fact that $\theta(Q)$ is a permutation, and hence that $V_{(\theta,\overline{n})}(x_v) = V_{(\theta,\overline{n})}(x_{v'})$ iff $V_{(\theta,\overline{n})}(x_{(Q,v)}) = V_{(\theta,\overline{n})}(x_{(Q,v')})$, and the result follows by Lemma 6.3. $\square$

We now show that any solution to a problem in EU can be translated into a satisfying valuation for the corresponding NPNAS problem.

Lemma 6.7 (Problem satisfaction from EU into NPNAS). *Suppose that $\overline{n} = names(S)$, $\overline{A} = nvars(S) = \{A \mid A \in dom(\theta)\}$ and $\overline{Q} = pvars(S) = \{Q \mid Q \in dom(\theta)\}$. If $\theta \models S$ and $\text{tr}(S) = \exists \overline{x}(\overline{c})$ then there exists a valuation $V^* \in \alpha\text{-}Tree_{\Sigma}(\Delta_{(\overline{n},\overline{A},\overline{Q},\overline{x})})$ which extends $V_{(\theta,\overline{n})}$ and is such that $V^* \models \overline{c}$.*

Proof. By induction on the structure of EU name-terms a, we can show that if $\text{tr}(a)_{\overline{x}} = \exists \overline{x}'(z \text{ where } \overline{c})$ (where $\overline{A} = \{A \mid A \in dom(\theta)\}$, $\overline{Q} = \{Q \mid Q \in dom(\theta)\}$ and $\overline{n}; \overline{A}; \overline{Q} \vdash a \ ok$) then there exists a valuation $V^* \in \alpha\text{-}Tree_{\Sigma}(\Delta_{(\overline{n},\overline{A},\overline{Q},(\overline{x} \uplus \overline{x}'))})$ which extends $V_{(\theta,\overline{n})}$ and is such that $V^* \models \overline{c}$ and $V^*(z) = \{\theta(a)\}$. This relates the result of instantiating a name-term in EU to the corresponding term instantiation in NPNAS.

We can then prove a similar result for atomic constraints: if $\theta \models c$ and $\text{tr}(c)_{\overline{x}} = \exists \overline{x}'(\overline{c})$ (where $\overline{A} = \{A \mid A \in dom(\theta)\}$, $\overline{Q} = \{Q \mid Q \in dom(\theta)\}$ and $\overline{n}; \overline{A}; \overline{Q} \vdash c \ ok$) then there exists a valuation $V^* \in \alpha\text{-}Tree_{\Sigma}(\Delta_{(\overline{n},\overline{A},\overline{Q},(\overline{x} \uplus \overline{x}'))})$ which extends $V_{(\theta,\overline{n})}$ and is such that $V^* \models \overline{c}$. This uses the above result and relates constraint satisfaction in EU to constraint satisfaction in NPNAS.

Then, if $\theta \models S$ then $\theta \models c$ holds for all $c \in S$, where we suppose that $S = \{c_1, \ldots, c_k\}$. We assume that $\text{tr}(S) = \exists \overline{x}(\overline{c})$ holds, where $\text{tr}(c_i)_{(\overline{x} \uplus \overline{x}_1 \uplus \cdots \uplus \overline{x}_{i-1})} = \exists \overline{x}_i(\overline{c}_i)$ holds for all $i \in \{1, \ldots, k\}$. Using the above result about constraint satisfaction we can construct a single NPNAS valuation V^* which extends $V_{(\theta,\overline{n})}$ and is such that $V^* \models \overline{c}_i$ holds for all $i \in \{1, \ldots, k\}$. Thus we have that $V^* \models \overline{c}_1 \ \& \ \cdots \ \& \ \overline{c}_k$ holds. By Lemma 6.6 we know that $V_{(\theta,\overline{n})} \models \mathbb{C}(\overline{n}, \overline{A}, \overline{Q})$ holds: hence $V^* \models \mathbb{C}(\overline{n}, \overline{A}, \overline{Q})$ holds also. Hence we get that $V^* \models \overline{c}$ holds, as required. $\square$

We now turn to the other direction—we begin by proving that any NPNAS valuation which satisfies a set of consistency constraints can be translated back into a corresponding EU valuation.

Lemma 6.8 (Consistency constraints imply an EU valuation). *If $V \in \alpha\text{-}Tree_{\Sigma}(\Delta_{(\overline{n},\overline{A},\overline{Q},\overline{x})})$ and $V \models \mathbb{C}(\overline{n}, \overline{A}, \overline{Q})$ then there exists a permutation π_V and a ground EU valuation θ_V (with $(\overline{A} \uplus \overline{Q}) \subseteq dom(\theta_V)$) such that*
(1) $\forall n \in \overline{n}. \ V(x_n) = \{\pi_V(n)\}$;
(2) $\forall A \in \overline{A}. \ V(x_A) = \{\pi_V \cdot (\theta_V(A))\}$; and
(3) $\forall Q \in \overline{Q}. \ \forall v \in (\overline{n} \uplus \overline{A}). \ V(x_{(Q,v)}) = \{\pi_V \cdot ((\theta_V(Q))(\theta_V(v)))\}$.

Proof. We prove the three points separately.
(1) This follows from the fact that $V \models \#_{\{x_n \mid n \in \overline{n}\}}$ holds, which implies that we can fix a permutation π_V which is a bijection between $\overline{n}$ and $\{V(x_n) \mid n \in \overline{n}\}$, as required.

(2) We note that $\{x_A \mid A \in \overline{A}\} \cap \{x_n \mid n \in \overline{n}\} = \emptyset$. Thus we can construct a ground EU valuation θ_V such that $\forall A \in \overline{A}.\ V(x_A) = \{\pi_V \cdot (\theta_V(A))\}$, by setting $\theta_V(A) = \pi_V{}^{-1}(n)$ if $V(x_A) = \{n\}$.

(3) Since $V \models \mathbb{C}(\overline{n}, \overline{A}, \overline{Q})$ we know that $V \models \texttt{<}x_v\texttt{><}x_{v'}\texttt{>}x_v = \texttt{<}x_{(Q,v)}\texttt{><}x_{(Q,v')}\texttt{>}x_{(Q,v)}$ holds for all $Q \in \overline{Q}$ and all $v, v' \in (\overline{n} \uplus \overline{A})$. By Lemma 6.3 we get that there is a bijection between $\{V(x_v) \mid v \in (\overline{n} \uplus \overline{A})\}$ and $\{V(x_{(Q,x_v)}) \mid v \in (\overline{n} \uplus \overline{A})\}$. We represent this bijection as a permutation π. From the first two proof obligations we know that $V(x_v) = \{\pi_V \cdot (\theta_V(v))\}$ for all $v \in (\overline{n} \uplus \overline{A})$, and hence that $V(x_{(Q,v)}) = \{\pi \cdot (\pi_V \cdot (\theta_V(v)))\}$, i.e. that $V(x_{(Q,v)}) = \{(\pi \circ \pi_V) \cdot (\theta_V(v))\}$, for all $v \in (\overline{n} \uplus \overline{A})$. Now, we can decompose π into the form $\pi_V \circ \pi_Q \circ \pi_V{}^{-1}$ for some π_Q. Thus we get that, for all $v \in (\overline{n} \uplus \overline{A})$, $V(x_{(Q,v)}) = \{(\pi_V \circ \pi_Q \circ \pi_V{}^{-1} \circ \pi_V) \cdot (\theta_V v)\} = \{(\pi_V \circ \pi_Q) \cdot (\theta_V v)\} = \{\pi_V \cdot (\pi_Q \cdot (\theta_V(v)))\}$. Thus if we let $\theta_V(Q) = \pi_Q$ then we get that $\forall Q \in \overline{Q}.\ \forall v \in (\overline{n} \uplus \overline{A}).\ V(x_{(Q,v)}) = \{\pi_V \cdot ((\theta_V(Q))(\theta_V(v)))\}$ holds, as required. $\square$

We now show that any satisfying valuation for the NPNAS translation of an EU problem can be translated back into a solution of the original EU problem.

Lemma 6.9 (Problem satisfaction from NPNAS into EU). *Suppose that $\overline{n} = names(S)$, $\overline{A} = nvars(S)$ and $\overline{Q} = pvars(S)$, and that $\texttt{tr}(S) = \exists \overline{x}(\overline{c})$, $V \in \alpha\text{-}Tree_\Sigma(\Delta_{(\overline{n}, \overline{A}, \overline{Q}, \overline{x})})$ and $V \models \overline{c}$ all hold. Then there exists a ground EU valuation θ_V (with $(\overline{A} \uplus \overline{Q}) \subseteq dom(\theta_V)$) such that $\theta_V \models S$.*

Proof. The structure of this proof rather mirrors that of Lemma 6.7. We begin by showing that if $\overline{n}; \overline{A}; \overline{Q} \vdash a\ ok$, $\texttt{tr}(a)_{\overline{x}} = \exists \overline{x}'(z \text{ where } \overline{c})$, $V \in \alpha\text{-}Tree_\Sigma(\Delta_{(\overline{n}, \overline{A}, \overline{Q}, (\overline{x} \uplus \overline{x}'))})$ and $V \models \overline{c}$ all hold then $V(z) = \{\pi_V \cdot \theta_V(a)\}$ holds, for some θ_V and π_V which satisfy the three conditions stated in Lemma 6.8. This relates instantiations of translated EU name-terms in NPNAS back to instantiations of the original EU term, up to a permutation.

We proceed to prove a similar result about constraint satisfaction—if $\overline{n}; \overline{A}; \overline{Q} \vdash c\ ok$, $\texttt{tr}(c)_{\overline{x}} = \exists \overline{x}'(\overline{c})$, $V \in \alpha\text{-}Tree_\Sigma(\Delta_{(\overline{n}, \overline{A}, \overline{Q}, (\overline{x} \uplus \overline{x}'))})$ and $V \models \overline{c}$ all hold then $\theta_V \models c$, where θ_V satisfies the three conditions stated in Lemma 6.8 (for some π_V). This uses the previous result and shows that satisfaction of translated EU constraints in NPNAS can be related back to the semantics of the original constraint in EU. There is a twist here, as we must use the equivariance property of the NPNAS semantics (Remark 3.11) to strip off the unwanted permutation π_V.

Thus, if $\texttt{tr}(S) = \exists \overline{x}(\overline{c})$ then $\overline{c} \equiv (\overline{c}_1 \ \& \cdots \& \ \overline{c}_k \ \& \ \mathbb{C}(\overline{n}, \overline{A}, \overline{Q}))$, $S \equiv \{c_1, \ldots, c_k\}$ and $\overline{x} = \overline{x}_1 \uplus \cdots \uplus \overline{x}_k$ all hold, where $\overline{n} = names(S)$, $\overline{A} = nvars(S)$, $\overline{Q} = pvars(S)$ and where $\texttt{tr}(c_i)_{(\overline{x}_1 \uplus \cdots \uplus \overline{x}_{i-1})} = \exists \overline{x}_i(\overline{c}_i)$ holds for all $i \in \{1, \ldots, k\}$. It follows that $\overline{n}; \overline{A}; \overline{Q} \vdash S\ ok$ holds. We assume that $V \models \overline{c}$, i.e. that $V \models \mathbb{C}(\overline{n}, \overline{A}, \overline{Q})$ holds and that $V \models \overline{c}_i$ holds for all $i \in \{1, \ldots, k\}$. By Lemma 6.8 we get that there exists a permutation π_V and a ground EU valuation θ_V (with $(\overline{A} \uplus \overline{Q}) \subseteq dom(\theta_V)$) which satisfy the three conditions laid out in the statement of that lemma. Then, using the above result on constraint satisfaction, we can show that $\theta_V \models c_i$ holds for all $i \in \{1, \ldots, k\}$, i.e. that $\theta_V \models S$ holds, as required. $\square$

The key result of this section is the following theorem, which demonstrates the correctness of the encoding of name-name equivariant unification into non-permutative nominal constraints.

Theorem 6.10 (Correctness of encoding). *Suppose that $\overline{n}; \overline{A}; \overline{Q} \vdash S$ ok and $\mathtt{tr}(S) = \exists \overline{x}(\overline{c})$ both hold. Then, S is satisfiable iff $\exists \Delta_{(\overline{n}, \overline{A}, \overline{Q}, \overline{x})}(\overline{c})$ is satisfiable.*

Proof. For the forward direction: if $\models S$ then there exists a ground EU valuation θ such that $\theta \models S$. If $\mathtt{tr}(S) = \exists \overline{x}(\overline{c})$ then by Lemma 6.7 there exists a valuation $V^* \in \alpha\text{-}Tree_\Sigma(\Delta_{(\overline{n}, \overline{A}, \overline{Q}, \overline{x})})$ such that $V^* \models S$, from which it follows that $\models S$ holds. For the reverse direction: if $\models \overline{c}$ then there exists a valuation $V \in \alpha\text{-}Tree_\Sigma(\Delta_{(\overline{n}, \overline{A}, \overline{Q}, \overline{x})})$ such that $V \models \overline{c}$. By Lemma 6.9 we can construct a ground EU valuation θ_V such that $\theta_V \models S$, which shows that $\models S$ holds. □

Example 6.11 (A translated EU problem). As an example, consider the following equivariant unification problem which involves permutation variables and swappings.

$$S = \{(Q \cdot A) = ((Q \cdot A)(Q \cdot B)) \cdot (Q \cdot A), (Q' \cdot A) \,\#\, (Q' \cdot B)\}$$

We would expect this to be unsatisfiable because the first constraint implies that $A = B$ whereas the second implies that $A \neq B$. The translation of S into NPNAS is as follows, where z is a freshly chosen variable.

$$\langle x_{(Q,A)} \rangle \langle x_{(Q,B)} \rangle z = \langle x_{(Q,B)} \rangle \langle x_{(Q,A)} \rangle x_{(Q,A)} \tag{6.1}$$

$$\&\quad x_{(Q,A)} = z \tag{6.2}$$

$$\&\quad x_{(Q',A)} \,\#\, x_{(Q',B)} \tag{6.3}$$

$$\&\quad \langle x_A \rangle \langle x_B \rangle x_A = \langle x_{(Q,A)} \rangle \langle x_{(Q,B)} \rangle x_{(Q,A)} \tag{6.4}$$

$$\&\quad \langle x_A \rangle \langle x_B \rangle x_A = \langle x_{(Q',A)} \rangle \langle x_{(Q',B)} \rangle x_{(Q',A)} \tag{6.5}$$

To see that the NPNAS problem is also unsatisfiable, we observe that 6.1 and 6.2 imply that $x_{(Q,A)} = x_{(Q,B)}$. This fact, along with 6.4, implies that $x_A = x_B$ which, together with 6.5, implies that $x_{(Q',A)} = x_{(Q',B)}$ holds. However, this contradicts 6.3 and thus it follows that the NPNAS problem is unsatisfiable.

In solving this problem, the decision procedure outlined in the proof of Theorem 5.31 must first construct and solve the first-order reduction of the NPNAS problem, as a termination check. In this case, this is straightforward as the names are erased and the abstractions replaced by tuples as outline above. Writing $(()^k, t)$ for $((), ((), \cdots ((), t)))$ if there are k nested occurrences of $()$, this leaves the following first-order unification problem which is trivially satisfiable.

$$(((()^2, ()) = (()^2, ())) \,\&\, (() = ()) \,\&\, (((()^2, ()) = (()^2, ())) \,\&\, (((()^2, ()) = (()^2, ()))$$

Having ascertained that the problem is strongly normalising, we proceed to compute the set of $\longrightarrow$-normal forms using the reduction rules from Figure 3. There are 27 cases to check in total, since for each of 6.1, 6.4 and 6.5 there are 3 branches according to reduction rule (E4). We do not specify a particular search strategy, provided that the entire reduction space is explored. In this case, none of the $\longrightarrow$-normal forms turn out to be solved in the sense of Definition 5.26, which corresponds to the fact that the problem is unsatisfiable, as argued above.

To see that the encoding presented in this section is a polynomial time reduction, suppose that there are k_n names, k_A name variables, k_Q permutation variables, k_{swap} swappings and k_c constraints in S. Then there are $k_n(k_n - 1) + k_Q(k_n + k_A) + k_c + k_{swap}$ constraints and $k_n + k_A + k_Q(k_n + k_A) + k_{swap}$ variables in $\overline{c}$, where $\mathtt{tr}(S) = \exists \overline{x}(\overline{c})$ (for some $\overline{x}$). These are both polynomial functions of the size of S.

Note that the translation defined in this section only deals with name-name equivariant unification problems. We can use the results from this section to derive a translation of full equivariant unification into NPNAS by using the "first phase" of Cheney's algorithm [Che10, Section 4.2.1] to reduce the problem into a name-name problem (if possible) before using the algorithm described herein to translate it into NPNAS. Note, however, that this is not a polynomial time reduction because the "first phase" of Cheney's algorithm has an exponential upper bound.

Remark 6.12 (A tractable subproblem). Say that a non-permutative constraint problem $\exists\Delta(\overline{c})$ is *permutative* iff every valuation $V \in \alpha\text{-}Tree_\Sigma(\Delta)$ such that $V \models \overline{c}$ also has $V \models \#_{nvars(\Delta)}$, where $nvars(\Delta)$ is the set of all variables x in $dom(\Delta)$ such that $\Delta(x)$ is a name sort. It was shown in [Lak10, Section 6.4] that satisfiability of problems in this subset can be decided in polynomial time via translation to nominal unification. This makes sense because the non-permutative behaviour which provides the additional power in NPNAS has been disallowed.

7. Related and future work

We have already discussed at length the relationship between non-permutative nominal constraints and the equivariant unification problem. The nominal unification problem of Urban, Pitts and Gabbay [UPG04] can be thought of as a ground subproblem of equivariant unification and hence its relationship to the work reported in this paper is subsumed by that of equivariant unification.

Our algorithm bears some similarities to Huet's algorithm for higher-order unification, that is, unification for typed λ-terms [Hue75]. That algorithm ignores equations between two terms on the basis that they are always satisfiable, much as our constraint transformation procedure ignores constraints of the form $<x_{1..k}>x = <y_{1..k}>y$ (where x and y are not of name sort). There may be other parallels worthy of investigation. However, it is worth noting that higher-order unification is known to be undecidable [Gol81] whereas we have shown that a decision procedure exists for satisfiability of non-permutative nominal constraints (see Theorem 5.31).

Higher-order unification forms the basis of an alternative technique for representing abstract syntax with binders known as higher-order abstract syntax (HOAS) [PE88] which has been used in various tools for specifying, and reasoning about, formal systems with binding constructs [NM88, PS99]. These tools often exploit higher-order patterns, which are a restricted class of λ-terms for which unification (modulo $\alpha\beta_0\eta$-equivalence) is decidable [Mil91]. Higher-order pattern unification has been shown to be equivalent to nominal unification [Che05, LV08] and it follows that our non-permutative nominal constraint language subsumes higher-order pattern unification just as it does nominal unification.

Future work is needed on the relationship between non-permutative nominal constraints and the full equivariant unification problem (i.e. not just for name terms). This may involve finding an equivalent of the bijection constraint construction from Lemma 6.3 which works for variables of any type, not just name sorts. The termination checker described above could be run initially or in parallel with the constraint transformation algorithm, or omitted altogether to give a semi-decision procedure. An alternative implementation strategy could be to encode non-permutative constraint problems as boolean formulae and use a SAT solver to decide their satisfiability. With more work it may be possible to improve the

algorithm for deciding satisfiability of constraint problems in NPNAS, in particular with regard to termination and non-deterministic search. The ideal algorithm would avoid the use of name-swappings and not require a separate termination check.

A key motivation for the development of equivariant unification was to give complete implementations of nominal logic programming [CU08] and nominal rewriting [FG07] in cases where the nominal unification [UPG04] is not sufficiently powerful. In other work [LP09, Lak10] we have investigated the use of non-permutative nominal constraints in the context of the functional-logic programming language αML—further work may be to investigate the theory of rewriting over non-permutative nominal terms.

8. Conclusion

Non-permutative nominal abstract syntax is a means of encoding terms with binders without the need for globally-fresh permutative names. We have defined a semantics for equality and freshness constraints over non-permutative nominal terms, and presented an algorithm for deciding satisfiability of these constraint problems. Our constraint solving procedure is novel in that it does not use permutations, which are standard in most nominal approaches to abstract syntax. This simplifies the term language but complicates the analysis, in particular the proof of termination. Our translation of name-name equivariant unification problems into non-permutative nominal constraints is also novel and demonstrates explicitly how the additional features of equivariant unification can be encoded using just permutative variables in binding position. Studies of non-permutative nominal constraints are important from both a theoretical and a practical perspective, as this algorithm could be used instead of the more complicated equivariant unification algorithm in situations where nominal unification cannot compute all solutions.

Acknowledgements

References

[AEH00] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. *Journal of the ACM*, 47(4):776–822, 2000.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[CF11] C. Calvès and M. Fernandéz. The first-order nominal link. In M. Alpuente, editor, *Logic-Based Program Synthesis and Transformation*, volume 6564 of *Lecture Notes in Computer Science*, pages 234–248. Springer-Verlag, 2011.

[Che04] J. Cheney. *Nominal Logic Programming*. PhD thesis, Cornell University, 2004.

[Che05] J. Cheney. Relating nominal and higher-order pattern unification. In L. Vigneron, editor, *Proceedings of the 19th International Workshop on Unification (UNIF 2005)*, pages 104–119, 2005. LORIA research report A05-R-022.

[Che10] J. Cheney. Equivariant unification. *Journal of Automated Reasoning*, 45(3):267–300, 2010.

[CU04] J. Cheney and C. Urban. Alpha-Prolog: A logic programming language with names, binding and alpha-equivalence. In B. Demoen and V. Lifschitz, editors, *Proceedings of the 20th International Conference on Logic Programming (ICLP 2004)*, number 3132 in Lecture Notes in Computer Science, pages 269–283. Springer-Verlag, 2004.

[CU08] J. Cheney and C. Urban. Nominal logic programming. *ACM Transactions on Programming Languages and Systems*, 30(5):1–47, 2008.

[DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[FG07] M. Fernández and M. J. Gabbay. Nominal rewriting. *Information and Computation*, 205:917–965, 2007.

[GM08] M. J. Gabbay and A. Mathijssen. Capture-avoiding substitution as a nominal algebra. *Formal Aspects of Computing*, 20(4–5):451–479, 2008.

[Gol81] W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[GP02] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3–5):341–363, 2002.

[Hue75] G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.

[Lak10] M. R. Lakin. *An executable meta-language for inductive definitions with binders*. PhD thesis, University of Cambridge, 2010.

[LP09] M. R. Lakin and A. M. Pitts. Resolving inductive definitions with binders in higher-order typed functional programming. In G. Castagna, editor, *Proceedings of the 18th European Symposium on Programming (ESOP 2009)*, volume 5502 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 2009.

[LV08] J. Levy and M. Villaret. Nominal unification from a higher-order perspective. In A. Voronkov, editor, *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA 2008)*, volume 5117 of *Lecture Notes in Computer Science*, pages 246–260. Springer-Verlag, 2008.

[LV10] Jordi Levy and Mateu Villaret. An efficient nominal unification algorithm. In Christopher Lynch, editor, *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 209–226, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Mil91] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[NM88] G. Nadathur and D. Miller. An overview of λProlog. In R. A. Kowalski and K. A. Bowen, editors, *Proceedings of the 5th International Conference on Logic Programming (ICLP 1988)*, pages 810–827. MIT Press, 1988.

[PE88] F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 1988)*, volume 23 of *ACM SIGPLAN Notices*, pages 199–208. ACM Press, 1988.

[Pit03] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186(2):165–193, 2003.

[Pit06] A. M. Pitts. Alpha-structural recursion and induction. *Journal of the ACM*, 53(3):459–506, 2006.

[PS99] F. Pfenning and C. Schürmann. System description: Twelf—a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE 1999)*, volume 1632 of *Lecture Notes in Artifical Intelligence*, pages 202–206. Springer-Verlag, 1999.

[UC05] C. Urban and J. Cheney. Avoiding equivariance in Alpha-Prolog. In P. Urzyczyn, editor, *Proceedings of the 7th International Conference on Typed Lambda Calculus and Applications (TLCA 2005)*, number 3461 in Lecture Notes in Computer Science, pages 74–89. Springer-Verlag, 2005.

[UPG04] C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.