Unranked Nominal Unification[⋆]

Besik Dundua[1,3], Temur Kutsia[2], and Mikheil Rukhaia[3]

[1] Kutaisi International University, Kutaisi, Georgia
[2] RISC, Johannes Kepler University Linz, Austria
[3] Institute of Applied Mathematics, Tbilisi State University, Georgia
bdundua@gmail.com, kutsia@risc.jku.at, mrukhaia@logic.at

Abstract. In this paper we define an unranked nominal language, an extension of the nominal language with tuple variables and term tuples. We define the unification problem for unranked nominal terms and present an algorithm solving the unranked nominal unification problem.

1 Introduction

Solving equations between logic terms is a fundamental problem with many important applications in mathematics, computer science, and artificial intelligence. It is needed to perform an inference step in reasoning systems and logic programming, to match a pattern to an expression in rule-based and functional programming, to extract information from a document, to infer types in programming languages, to compute critical pairs while completing a rewrite system, to resolve ellipsis in natural language processing, etc. Unification and matching are well-known techniques used in these tasks.

Unification (as well as matching) is a quite well-studied topic for the case when the equality between function symbols is precisely defined. This is the standard setting. There is quite some number of unification algorithms whose complexities range from exponential [34] to linear [31]. Besides, many extensions and generalizations have been proposed. Those relevant to our interests are equational unification (more precisely, associative unification with unit element) (see, e.g., [7]), word unification [10,17,29], and sequence unification [22,25,26]. There are some good surveys on unification [8,11,18,19].

Nominal logic [12,33] extends first-order logic with primitives for renaming via name-swapping, for freshness of names, and for name-binding. Such kind of constructs are important in meta-programming and meta-deduction. Nominal logic provides a simple formalism for reasoning about abstract syntax modulo α-equivalence. A nominal term $a.t$ is an example of abstraction, binding every occurrence of atom a in t. Term equality ($t \approx t'$) in nominal language is considered modulo renaming of bound variables (atoms), i.e., it is α-equivalence, formalized inside the language itself. The α-equivalence is a meta-relation in first-order syntax, but it is formulated at the object level in nominal languages.

[⋆] Supported by the Austrian Science Fund (FWF), project 28789-N32, and by the Shota Rustaveli National Science Foundation, grant YS-19-367.

For such formulations it is important to explicitly define which atom can be considered as a new atom for a given term. This relation ($a\#t$), called freshness relation, is also formulated on the object level in nominal languages.

Solving equations between nominal terms needs a special unification algorithm [39], which is first-order, but can be also seen from the higher-order perspective via mapping from/to higher-order pattern unification [28]. The standard nominal language contains fixed-arity symbols and one kind of variable, corresponding to individual variables from first-order syntax. In nominal languages, a unification problem, e.g. $a.x \approx^? b.y$, is solved by a pair $\langle\{b\#x\},\ \{y \mapsto (a\,b)\cdot x\}\rangle$. The first component of the solution, the freshness constraint $b\#x$, requires that b should not occur free in any possible instantiation of x. The second component, the substitution, tells us that the solution must replace the variable y with the term $(a\,b)\cdot x$. The latter means that atoms a and b are swapped in every possible instantiation of x.

As we mentioned above, the constructs provided by nominal logic are important for meta-deduction. However, this formalism, as well as many representation formats for formalized mathematics typically do not provide a structural analog for ellipses ($\ldots$) which are commonly used in mathematical texts [14, 15]. In the literature, the latter problem has been addressed by permitting unranked (also known as variadic, flexary, or flexible arity) symbols in the language, introducing sequences in the meta-level, and extending the language with sequence variables, see, e.g., [15, 20, 21, 24].

In this paper we present a combination of these two approaches, extending nominal languages by unranked symbols and studying the fundamental computational mechanism for them: unification. However, unlike the above mentioned unranked languages, where sequences are introduced in the meta-level, nominal syntax allows us to introduce their analogs in the object level. This is done by generalizing already existing syntactic constructs, pairs, to arbitrary tuples. They should be flat, which is achieved by imposing a special α-equivalence rule for them.

Term pairs, which are a part of nominal syntax in some papers (e.g., [2, 39]) have been extended to term tuples in [3, 4], but our approach differs in that we additionally introduce variables that can be instantiated by tuples (*tuple variables*, that resemble sequence variables), and the mentioned notion of flatness.

The paper is organized as follows: In Section 2, we define the language. The unification rules and a strategy that guarantees soundness and tries to minimize redundant computations are discussed in Section 3. Some terminating fragments of unranked nominal unification are introduced in Section 4. In Section 5, we discuss related problems and explain our design choices. Section 6 concludes.

2 Unranked Nominal Language

In our signature we have pairwise disjoint sets of atoms ($a, b, \ldots$), function symbols ($f, g, \ldots$), individual variables ($x, y, \ldots$), tuple variables ($X, Y, \ldots$), and the tuple constructor $\langle\rangle$. *Permutations* are a finite (possibly empty) sequence of swappings, which are pairs of atoms $(a\,b)$. We use π to denote permutations, write *id* for the identity (empty) permutation and $\pi_1 \circ \pi_2$ for concatenating two permutations.

An *unranked nominal term* ($t, s, \ldots$, shortly a term) is either an individual term r, a tuple variable with a suspended permutation $\pi \cdot X$, or a possibly empty tuple of terms $\langle t_1, \ldots, t_n \rangle$:

$$t ::= r \mid \pi \cdot X \mid \langle t_1, \ldots, t_n \rangle, \quad n \geq 0,$$

where *individual terms* are defined by the grammar

$$r ::= a \mid a.t \mid \pi \cdot x \mid f\langle t_1, \ldots, t_n \rangle, \quad n \geq 0.$$

We will write $t\colon \iota$ to indicate that t is an individual term. The terms $\pi \cdot x$ and $\pi \cdot X$ are called *suspensions*. We skip π if $\pi = id$. The inverse of a permutation π, denoted by π^{-1}, is obtained by reversing the list of swappings from π.

Permutation action on terms is defined as follows:

- $id \cdot a = a$.
- $((a_1\, a_2) \circ \pi) \cdot a = \begin{cases} a_1, & \text{if } \pi \cdot a = a_2, \\ a_2, & \text{if } \pi \cdot a = a_1, \\ \pi \cdot a, & \text{otherwise.} \end{cases}$
- $\pi \cdot (a.t) = (\pi \cdot a).(\pi \cdot t)$.
- $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$ and $\pi \cdot (\pi' \cdot X) = (\pi \circ \pi') \cdot X$.
- $\pi \cdot (f\langle t_1, \ldots, t_n \rangle) = f(\pi \cdot \langle t_1, \ldots, t_n \rangle)$, and
- $\pi \cdot \langle t_1, \ldots, t_n \rangle = \langle \pi \cdot t_1, \ldots, \pi \cdot t_n \rangle$.

The disagreement set of two permutations π and π' is defined as $ds(\pi, \pi') ::= \{a \mid \pi \cdot a \neq \pi' \cdot a\}$. Further, we often omit expressions like $a \neq b$, assuming that atoms differ by their names.

Substitution is a mapping from individual variables to individual terms and from tuple variables to tuples such that all but finitely many individual variables are mapped to themselves, and all but finitely many tuple variables are mapped to singleton tuples consisting of that variable only (i.e., mapping X to $\langle X \rangle$). They are usually written as finite sets, e.g., $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n, X_1 \mapsto \langle t_{11}, \ldots, t_{1n_1} \rangle, \ldots, X_m \mapsto \langle t_{m1}, \ldots, t_{mn_m} \rangle]$. We use the letter σ for substitutions in general and ε for the identity substitution, i.e., $\varepsilon(x) = x$ and $\varepsilon(X) = \langle X \rangle$ for all individual and tuple variables x and X.

Application of a substitution σ to a term t is defined as follows:

- $a\sigma = a$.
- $(a.t)\sigma = a.t\sigma$.
- $(f\langle t_1, \ldots, t_n \rangle)\sigma = f\langle t_1, \ldots, t_n \rangle\sigma$.
- $\langle t_1, \ldots, t_n \rangle\sigma = \langle t_1\sigma, \ldots, t_n\sigma \rangle$, where nested tuples are flattened.
- $(\pi \cdot x)\sigma = \pi \cdot \sigma(x)$ and $(\pi \cdot X)\sigma = \pi \cdot \sigma(X)$, where π acts on $\sigma(x)$ and $\sigma(X)$ as permutation action.

For instance, for a substitution

$$\sigma = [x \mapsto f\langle b.b\rangle,\ X \mapsto \langle a, f\langle c.c, (a\,c)\cdot Z\rangle\rangle,\ Y \mapsto \langle\rangle]$$

we have

$$a.f\langle (a\,b)\cdot x,\ (a\,b)\cdot X,\ (a\,b)\cdot Y\rangle\sigma = a.f\langle f\langle a.a\rangle,\ b,\ f\langle c.c, (a\,b)(a\,c)\cdot Z\rangle\rangle.$$

A freshness environment (denoted by ∇) is a list of freshness constraints $a\#x$ and $a\#X$, meaning that the instantiations of x and X cannot contain free occurrences of a. The flatness property of tuples is formalized by the axiom $\approx$-flat. (where $n \geq 0, k \geq 0, m \geq 0$)

The equivalence ($\approx$) is defined by the following rules:

$$\frac{}{\nabla \vdash \langle\rangle \approx \langle\rangle}\approx\text{-unit} \qquad \frac{}{\nabla \vdash a \approx a}\approx\text{-atom}$$

$$\frac{}{\nabla \vdash \langle t_1,\ldots,t_n,\langle t'_1,\ldots,t'_k\rangle,t''_1,\ldots,t''_m\rangle \approx \langle t_1,\ldots,t_n,t'_1,\ldots,t'_k,t''_1,\ldots,t''_m\rangle}\approx\text{-flat}$$

$$\frac{\nabla \vdash t_1 \approx t'_1 \quad \ldots \quad \nabla \vdash t_n \approx t'_n}{\nabla \vdash \langle t_1,\ldots,t_n\rangle \approx \langle t'_1,\ldots,t'_n\rangle}\approx\text{-tuple}$$

$$\frac{\nabla \vdash \langle t_1,\ldots,t_n\rangle \approx \langle t'_1,\ldots,t'_n\rangle}{\nabla \vdash f\langle t_1,\ldots,t_n\rangle \approx f\langle t'_1,\ldots,t'_n\rangle}\approx\text{-application}$$

$$\frac{t \approx t'}{\nabla \vdash a.t \approx a.t'}\approx\text{-abst.1} \qquad \frac{\nabla \vdash t \approx (a\ b)\cdot t' \quad \nabla \vdash a\#t'}{\nabla \vdash a.t \approx b.t'}\approx\text{-abst.2}$$

$$\frac{a\#x \in \nabla \text{ for all } a \in ds(\pi,\pi')}{\nabla \vdash \pi\cdot x \approx \pi'\cdot x}\approx\text{-susp.1}$$

$$\frac{a\#X \in \nabla \text{ for all } a \in ds(\pi,\pi')}{\nabla \vdash \pi\cdot X \approx \pi'\cdot X}\approx\text{-susp.2}$$

and the freshness predicate ($\#$) is defined by:

$$\frac{}{\nabla \vdash a\#\langle\rangle}\#\text{-unit} \qquad \frac{\nabla \vdash a\#t_1 \quad \ldots \quad \nabla \vdash a\#t_n}{\nabla \vdash a\#\langle t_1,\ldots,t_n\rangle}\#\text{-tuple}$$

$$\frac{}{\nabla \vdash a\#b}\#\text{-atom} \qquad \frac{\nabla \vdash a\#\langle t_1,\ldots,t_n\rangle}{\nabla \vdash a\#f\langle t_1,\ldots,t_n\rangle}\#\text{-application}$$

$$\frac{}{\nabla \vdash a\#a.t}\#\text{-abst.1} \qquad \frac{\nabla \vdash a\#t}{\nabla \vdash a\#b.t}\#\text{-abst.2}$$

$$\frac{(\pi^{-1}\cdot a\#x) \in \nabla}{\nabla \vdash a\#\pi\cdot x}\#\text{-susp.1} \qquad \frac{(\pi^{-1}\cdot a\#X) \in \nabla}{\nabla \vdash a\#\pi\cdot X}\#\text{-susp.2}$$

Proposition 1. *Given a freshness context ∇, a permutation π, an atom a and a term t, we have:*

(1) If $\nabla \vdash a\#\pi \cdot t$ then $\nabla \vdash \pi^{-1} \cdot a\#t$.
(2) If $\nabla \vdash \pi \cdot a\#t$ then $\nabla \vdash a\#\pi^{-1} \cdot t$.
(3) If $\nabla \vdash a\#t$ then $\nabla \vdash \pi \cdot a\#\pi \cdot t$.

Proof. Using induction on the structure of t and the fact that $\pi \cdot a = b$ iff $a = \pi^{-1} \cdot b$. The last statement is the consequence of (2) and the fact that permutations are bijections on atoms. □

The size of a term t, denoted by $|t|$, is defined by:

$$|\pi \cdot x| = |\pi \cdot X| = |a| = |\langle\rangle| = 1, \qquad |a.t| = 1 + |t|,$$
$$|f\langle t_1, \ldots, t_n\rangle| = 1 + |\langle t_1, \ldots, t_n\rangle|, \qquad |\langle t_1, \ldots, t_n\rangle| = 1 + |t_1| + \ldots + |t_n|.$$

Further, we define the size of an equation as $|t \approx t'| = |t| + |t'|$ and the size of a freshness constraint as $|a\#t| = |t|$.

Theorem 1. *$\approx$ is an equivalence relation.*

Proof. Similar to the corresponding results from [5, 39].

- Reflexivity is by a simple induction on the structure of terms.
- Transitivity is by an induction on the size of terms using the properties:
  - permutations can be moved from one side of the freshness relation to the other by forming the inverse permutation (Proposition 1).
  - the freshness relation is preserved under $\approx$ and permutation actions.
- Symmetry is by a simple induction on the structure of terms using the Proposition 1 and preservation of freshness under alpha-equivalence.

□

3 Unification

An unranked nominal unification problem P is a finite set of equational $t \approx^? t'$ or freshness problems $a\#^?t$. Tuples occurring in the unification problem are always flattened (e.g. after substitution application, etc.). A solution for P is a pair (∇, σ) such that for all problems $t \approx^? t'$ in P we have $\nabla \vdash \sigma(t) \approx \sigma(t')$ and for all problems $a\#^?t$ in P we have $\nabla \vdash a\#\sigma(t)$.

To describe the unification algorithm we use so called labeled transformation of unification problems: $P \overset{\sigma}{\Longrightarrow} P'$ and $P \overset{\nabla}{\Longrightarrow} P'$ which are given below (note that if in $\approx$-susp.1,2 $\pi = \pi'$, then $ds(\pi, \pi')$ and thus $\{a\#^?x, X \mid a \in ds(\pi, \pi')\}$ is empty; $V(t)$ denotes the set of variables occurring in t):

($\approx^?$-atom) $\quad \{a \approx^? a\} \cup P \overset{\varepsilon}{\Longrightarrow} P.$

($\approx^?$-unit) $\quad \{\langle\rangle \approx^? \langle\rangle\} \cup P \overset{\varepsilon}{\Longrightarrow} P.$

($\approx^?$-function) $\quad \{f\langle t_1, \ldots, t_n\rangle \approx^? f\langle t'_1, \ldots, t'_m\rangle\} \cup P \overset{\varepsilon}{\Longrightarrow}$
$\{\langle t_1, \ldots, t_n\rangle \approx^? \langle t'_1, \ldots, t'_m\rangle\} \cup P.$

($\approx^?$ -abst.1) $\{a.t \approx^? a.t'\} \cup P \overset{\varepsilon}{\Longrightarrow} \{t \approx^? t'\} \cup P.$

($\approx^?$ -abst.2) $\{a.t \approx^? b.t'\} \cup P \overset{\varepsilon}{\Longrightarrow} \{t \approx^? (a\ b) \cdot t', a\#^? t'\} \cup P.$

($\approx^?$ -susp.1) $\{\pi \cdot x \approx^? \pi' \cdot x\} \cup P \overset{\varepsilon}{\Longrightarrow} \{a\#^? x \mid a \in ds(\pi, \pi')\} \cup P.$

($\approx^?$ -susp.2) $\{\pi \cdot X \approx^? \pi' \cdot X\} \cup P \overset{\varepsilon}{\Longrightarrow} \{a\#^? X \mid a \in ds(\pi, \pi')\} \cup P.$

($\approx^?$ -tuple) $\{\langle t, t_1, \ldots, t_n\rangle \approx^? \langle t', t'_1, \ldots, t'_m\rangle\} \cup P \overset{\varepsilon}{\Longrightarrow}$
$\{t \approx t', \langle t_1, \ldots, t_n\rangle \approx^? \langle t'_1, \ldots, t'_m\rangle\} \cup P,$
where t and t' are not tuple variables.

($\approx^?$ -proj.1) $\{\langle \pi \cdot X, t_1, \ldots, t_n\rangle \approx^? \langle t'_1, \ldots, t'_m\rangle\} \cup P \overset{\sigma}{\Longrightarrow}$
$\{\langle t_1\sigma, \ldots, t_n\sigma\rangle \approx^? \langle t'_1\sigma, \ldots, t'_m\sigma\rangle\} \cup P\sigma,$
where $\sigma = [X \mapsto \langle\rangle]$.

($\approx^?$ -proj.2) $\{\langle t_1, \ldots, t_n\rangle \approx^? \langle \pi \cdot X, t'_1, \ldots, t'_m\rangle\} \cup P \overset{\sigma}{\Longrightarrow}$
$\{\langle t_1\sigma, \ldots, t_n\sigma\rangle \approx^? \langle t'_1\sigma, \ldots, t'_m\sigma\rangle\} \cup P\sigma,$
where $\sigma = [X \mapsto \langle\rangle]$.

($\approx^?$ -widen.1) $\{\langle \pi \cdot X, t_1, \ldots, t_n\rangle \approx^? \langle t, t'_1, \ldots, t'_m\rangle\} \cup P \overset{\sigma}{\Longrightarrow}$
$\{\langle X', t_1\sigma, \ldots, t_n\sigma\rangle \approx^? \langle t'_1\sigma, \ldots, t'_m\sigma\rangle\} \cup P\sigma,$
where $\sigma = [X \mapsto \pi^{-1} \cdot \langle t, X'\rangle], X \notin V(t)$.

($\approx^?$ -widen.2) $\{\langle t, t_1, \ldots, t_n\rangle \approx^? \langle \pi \cdot X, t'_1, \ldots, t'_m\rangle\} \cup P \overset{\sigma}{\Longrightarrow}$
$\{\langle t_1\sigma, \ldots, t_n\sigma\rangle \approx^? \langle X', t'_1\sigma, \ldots, t'_m\sigma\rangle\} \cup P\sigma,$
where $\sigma = [X \mapsto \pi^{-1} \cdot \langle t, X'\rangle], X \notin V(t)$.

($\approx^?$ -var.1) $\{\pi \cdot x \approx^? t : \iota\} \cup P \overset{\sigma}{\Longrightarrow} P\sigma,$
where $\sigma = [x \mapsto \pi^{-1} \cdot t], x \notin V(t)$.

($\approx^?$ -var.2) $\{t : \iota \approx^? \pi \cdot x\} \cup P \overset{\sigma}{\Longrightarrow} P\sigma,$
where $\sigma = [x \mapsto \pi^{-1} \cdot t], x \notin V(t)$.

($\approx^?$ -var.3) $\{\pi \cdot X \approx^? t\} \cup P \overset{\sigma}{\Longrightarrow} P\sigma,$
where $\sigma = [X \mapsto \langle \pi^{-1} \cdot t\rangle], X \notin V(t)$.

($\approx^?$ -var.4) $\{t \approx^? \pi \cdot X\} \cup P \overset{\sigma}{\Longrightarrow} P\sigma,$
where $\sigma = [X \mapsto \langle \pi^{-1} \cdot t\rangle], X \notin V(t)$.

($\#^?$-atom) $\{a\#^? b\} \cup P \overset{\emptyset}{\Longrightarrow} P.$

($\#^?$-unit) $\{a\#^? \langle\rangle\} \cup P \overset{\emptyset}{\Longrightarrow} P.$

(#$^?$-tuple) $\{a\#^? \langle t_1,\dots,t_n\rangle\} \cup P \overset{\emptyset}{\Longrightarrow} \{a\#^? t_1,\dots,a\#^? t_n\} \cup P.$

(#$^?$-function) $\{a\#^? f\langle t_1,\dots,t_n\rangle\} \cup P \overset{\emptyset}{\Longrightarrow} \{a\#^? \langle t_1,\dots,t_n\rangle\} \cup P.$

(#$^?$-abst.1) $\{a\#^? a.t\} \cup P \overset{\emptyset}{\Longrightarrow} P.$

(#$^?$-abst.2) $\{a\#^? b.t\} \cup P \overset{\emptyset}{\Longrightarrow} \{a\#^? t\} \cup P.$

(#$^?$-susp.1) $\{a\#^? \pi\cdot x\} \cup P \overset{\nabla}{\Longrightarrow} P,$ where $\nabla = \{\pi^{-1}\cdot a\#x\}.$

(#$^?$-susp.2) $\{a\#^? \pi\cdot X\} \cup P \overset{\nabla}{\Longrightarrow} P,$ where $\nabla = \{\pi^{-1}\cdot a\#X\}.$

The naive algorithm, as presented in [28], is divided into two phases: first apply as many $\overset{\sigma}{\Longrightarrow}$ transformations as possible. It might cause branching due to tuple variables. On some branches, there might be no equational problems left. We expand them by $\overset{\nabla}{\Longrightarrow}$ transformations as long as possible. If we do not end up with the empty problem, then halt with failure, otherwise from the sequence of transformations $P \overset{\sigma_1}{\Longrightarrow} \cdots \overset{\sigma_n}{\Longrightarrow} P' \overset{\nabla_1}{\Longrightarrow} \cdots \overset{\nabla_m}{\Longrightarrow} \emptyset$ construct the solution $(\nabla_1 \cup \cdots \cup \nabla_m, \sigma_n \circ \cdots \circ \sigma_1)$. Some branches might directly lead to failure after application of $\overset{\sigma}{\Longrightarrow}$ rules. Some branches might cause more and more branching, leading to infinite sets of solutions. Employing some fair strategy of search tree development, we can have a complete method to enumerate them.

Example 1. We give examples of some unification problems and their solutions:

– Problem: $\{f\langle a.\langle X,x,Y\rangle\rangle \approx^? f\langle b.\langle f\langle X\rangle,x,b,c\rangle\rangle\}$.
 Solution: $(\emptyset, [X\mapsto\langle\rangle, x\mapsto f\langle\rangle, Y\mapsto\langle f\langle\rangle,a,c\rangle])$.

– Problem: $\{a.b.f\langle X,b\rangle \approx^? b.a.f\langle a,X\rangle\}$.
 Solution: $(\emptyset, [X\mapsto\langle\rangle])$.

– Problem: $\{f\langle X,a\rangle \approx^? f\langle a,Y\rangle\}$.
 Solutions: $(\emptyset, [X\mapsto\langle\rangle, Y\mapsto\langle\rangle])$ and $(\emptyset, [X\mapsto\langle a,Z\rangle, Y\mapsto\langle Z,a\rangle])$.

 If instead of Y we had X, then there would be infinitely many solutions: $(\emptyset,[X\mapsto\langle\rangle])$, $(\emptyset,[X\mapsto\langle a\rangle])$, $(\emptyset,[X\mapsto\langle a,a\rangle])$, ….

– Problem: $\{a.f\langle X,a\rangle \approx^? b.f\langle b,X\rangle\}$.
 Solution: $(\emptyset, [X\mapsto\langle\rangle])$.

– Problem: $\{a.f\langle X,a\rangle \approx^? b.f\langle b,Y\rangle\}$.
 Solutions: $(\emptyset, [X\mapsto\langle\rangle, Y\mapsto\langle\rangle])$ and $(\{b\#Z\}, [X\mapsto\langle a,Z\rangle, Y\mapsto\langle (a\,b)\cdot Z, b\rangle])$.

– Problem: $\{a.f\langle X,c\rangle \approx^? b.f\langle c,Y\rangle\}$.
 Solutions: $(\emptyset, [X\mapsto\langle\rangle, Y\mapsto\langle\rangle])$ and $(\{b\#Z\}, [X\mapsto\langle c,Z\rangle, Y\mapsto\langle (a\,b)\cdot Z, c\rangle])$.

– Problem: $\{f\langle X,Y\rangle \approx^? f\langle a,b,X\rangle, b\#X\}$.
 Solutions: $(\emptyset, [X\mapsto\langle\rangle, Y\mapsto\langle a,b\rangle])$ and $(\emptyset, [X\mapsto\langle a\rangle, Y\mapsto\langle b,a\rangle])$.

Without $b\#X$, the problem $\{f\langle X,Y\rangle \approx^? f\langle a,b,X\rangle\}$ has infinitely many solutions: $(\emptyset,[X \mapsto \langle\rangle, Y \mapsto \langle a,b\rangle])$, $(\emptyset,[X \mapsto \langle a\rangle, Y \mapsto \langle b,a\rangle])$, $(\emptyset,[X \mapsto \langle a,b\rangle, Y \mapsto \langle a,b\rangle])$, $(\emptyset,[X \mapsto \langle a,b,a\rangle, Y \mapsto \langle b,a\rangle])$, $(\emptyset,[X \mapsto \langle a,b,a,b\rangle, Y \mapsto \langle a,b\rangle])$, $(\emptyset,[X \mapsto \langle a,b,a,b,a\rangle, Y \mapsto \langle b,a\rangle])$, ...

The naive algorithm, described above, can be non-terminating even when there is a finite number of solutions. This is illustrated by the following example.

Example 2. Let us consider the following unification problem:

$$\begin{aligned}
&\{a.f\langle X,a\rangle \approx^? b.f\langle b,X\rangle\} \Longrightarrow_{\approx^?\textsf{-abst.2}}\\
&\{f\langle X,a\rangle \approx^? f\langle a,(a\,b)\cdot X\rangle, a\#^? f\langle b,X\rangle\} \Longrightarrow_{\approx^?\textsf{-function}}\\
&\{\langle X,a\rangle \approx^? \langle a,(a\,b)\cdot X\rangle, a\#^? f\langle b,X\rangle\}
\end{aligned}$$

Now, we can apply $\approx^?$ -proj.1 rule (followed by $\approx^?$ -atom and several $\overset{\nabla}{\Longrightarrow}$ transformations) to obtain a solution $(\emptyset,[X \mapsto \langle\rangle])$.

When we apply $\approx^?$ -widen.1 rules, we get non-terminating branch:

$$\begin{aligned}
&\{\langle X,a\rangle \approx^? \langle a,(a\,b)\cdot X\rangle, a\#^? f\langle b,X\rangle\} \overset{[X\mapsto\langle a,X_1\rangle]}{\Longrightarrow}\\
&\{\langle X_1,a\rangle \approx^? \langle b,(a\,b)\cdot X_1\rangle, a\#^? f\langle b,a,X_1\rangle\} \overset{[X_1\mapsto\langle b,X_2\rangle]}{\Longrightarrow}\\
&\{\langle X_2,a\rangle \approx^? \langle a,(a\,b)\cdot X_2\rangle, a\#^? f\langle b,a,b,X_2\rangle\} \Longrightarrow \cdots
\end{aligned}$$

But we could apply $\overset{\nabla}{\Longrightarrow}$ transformations on a subset $\{a\#^? f\langle b,a,X_1\rangle\}$, obtain $\{a\#^? a, a\#^? X_1\}$ and stop with failure since $a\#^? a$ has no solution.

An obvious attempt to fix the problem for such cases would be to delay application of the $\approx^?$ -widen.1 and $\approx^?$ -widen.2 rules until all possible $\overset{\nabla}{\Longrightarrow}$ transformations are applied. But we should be careful not to remove freshness constraints from the problems too early.

Consider the following example: $\{a \approx^? x, a\#^? x\}$. If we apply the $\#^?$-susp.1 rule first and then $\approx^?$ -var.2, we will obtain a wrong solution $(\{a\#x\},[x \mapsto a])$. Thus, we should delay application of the $\#^?$-susp.1 and $\#^?$-susp.2 rules until all possible $\overset{\sigma}{\Longrightarrow}$ transformations are applied.

The discussion above leads to the following strategy S:

– first apply as many $\overset{\sigma}{\Longrightarrow}$ transformations as possible except the $\approx^?$ -proj.1,2 and $\approx^?$ -widen.1,2 rules.
– if no other $\overset{\sigma}{\Longrightarrow}$ transformation is possible, $\approx^?$ -proj.1,2 and $\approx^?$ -widen.1,2 rules can be applied in parallel. However, before the $\approx^?$ -widen.1 and $\approx^?$ -widen.2 rules, one should apply as many $\overset{\nabla}{\Longrightarrow}$ transformations as possible except the $\#^?$-susp.1 and $\#^?$-susp.2 rules.
– use $\#^?$-susp.1 and $\#^?$-susp.2 rules if no other rules are applicable.
– If there is at least one equational problem in P such that no transformation rule is applicable on it, immediately halt the development of that branch with failure.

Theorem 2. *Given a unification problem P, if the unranked unification algorithm with the strategy S fails on P, then P has no solution; and if it succeeds on P, then the result is a unifier.*

Proof. The idea is that mixing equational and freshness rules except $\#^?$-susp.1,2 does not cause soundness problems, since those freshness rules do not affect variables. Parallel application of widening and projection rules together with the iteration of the strategy make sure that no solution is lost (cf. Theorem 51 in [22]).

For a unification problem P, the strategy S fails on P if there is at least one of the following pairs $a \approx^? b$, $b \approx^? a$, $a \approx^? \langle\rangle$, $\langle\rangle \approx^? a$, $a\#^?a$ in P', obtained by applying simplification rules to P, or there is an occurs check violation in $\approx^?$-var rules. Clearly, P has no solution in these cases.

If the strategy S succeeds on P, then we get a result $(\nabla_1 \cup \cdots \cup \nabla_n, \sigma_1 \circ \cdots \circ \sigma_m)$. The proof continues by simple induction on transitions with the following induction hypothesis:

– If $P \overset{\sigma}{\Longrightarrow} P'$ and (∇, σ') is a unifier for P', then $(\nabla, \sigma \circ \sigma')$ is a unifier for P.
– If $P \overset{\nabla}{\Longrightarrow} P'$ and (∇', σ) is a unifier for P', then $(\nabla \cup \nabla', \sigma)$ is a unifier for P. □

Example 3. We demonstrate how the strategy works on a unification problem.

$$\{a.f\langle X, x, Y, f\langle y, x\rangle\rangle \approx^? b.f\langle g\langle X\rangle, x, b, Z, f\langle g\langle X\rangle, y\rangle\rangle\} \qquad \Longrightarrow_{\approx^?\text{-abst.2}}$$

$$\begin{aligned}&\{f\langle X, x, Y, f\langle y, x\rangle\rangle \approx^? \\ &\quad f\langle g\langle (a\,b)\cdot X\rangle, (a\,b)\cdot x, a, (a\,b)\cdot Z, f\langle g\langle (a\,b)\cdot X\rangle, (a\,b)\cdot y\rangle\rangle, \\ &\quad a\#^? f\langle g\langle X\rangle, x, b, Z, f\langle g\langle X\rangle, y\rangle\rangle\} \qquad \Longrightarrow_{\approx^?\text{-function}}\end{aligned}$$

$$\begin{aligned}&\{\langle X, x, Y, f\langle y, x\rangle\rangle \approx^? \\ &\quad \langle g\langle (a\,b)\cdot X\rangle, (a\,b)\cdot x, a, (a\,b)\cdot Z, f\langle g\langle (a\,b)\cdot X\rangle, (a\,b)\cdot y\rangle\rangle, \\ &\quad a\#^? f\langle g\langle X\rangle, x, b, Z, f\langle g\langle X\rangle, y\rangle\rangle\} \qquad \overset{[X\mapsto\langle\rangle]}{\Longrightarrow}_{\approx^?\text{-proj.1}}\end{aligned}$$

Note, that at this point $\approx^?$-widen.1 is not applicable because of occurs check: $X \in V(g\langle (a\,b)\cdot X\rangle)$.

$$\begin{aligned}&\{\langle x, Y, f\langle y, x\rangle\rangle \approx^? \langle g\langle\rangle, (a\,b)\cdot x, a, (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle, \\ &\quad a\#^? f\langle g\langle\rangle, x, b, Z, f\langle g\langle\rangle, y\rangle\rangle\} \qquad \Longrightarrow_{\approx^?\text{-tuple}}\end{aligned}$$

$$\begin{aligned}&\{x \approx^? g\langle\rangle, \langle Y, f\langle y, x\rangle\rangle \approx^? \langle (a\,b)\cdot x, a, (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle, \\ &\quad a\#^? f\langle g\langle\rangle, x, b, Z, f\langle g\langle\rangle, y\rangle\rangle\} \qquad \overset{[x\mapsto g\langle\rangle]}{\Longrightarrow}_{\approx^?\text{-var.1}}\end{aligned}$$

$$\begin{aligned}&\{\langle Y, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle g\langle\rangle, a, (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle, \\ &\quad a\#^? f\langle g\langle\rangle, g\langle\rangle, b, Z, f\langle g\langle\rangle, y\rangle\rangle\}.\end{aligned}$$

Now, we have two branches: (a) continue again with $\approx^?$ -proj.1, followed by $\approx^?$ -tuple and several $\overset{\nabla}{\Longrightarrow}$ transformations (given also below), leading to the failure; and (b) continue with the $\overset{\nabla}{\Longrightarrow}$ transformations followed by $\approx^?$ -widen.1:

$$\begin{aligned}
&\{\langle Y, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle g\langle\rangle, a, (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle,\\
&\quad a\#^? f\langle g\langle\rangle, g\langle\rangle, b, Z, f\langle g\langle\rangle, y\rangle\rangle\} && \Longrightarrow_{\#^?\text{-function}}\\
&\{\langle Y, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle g\langle\rangle, a, (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle,\\
&\quad a\#^? \langle g\langle\rangle, g\langle\rangle, b, Z, f\langle g\langle\rangle, y\rangle\rangle\} && \Longrightarrow_{\#^?\text{-tuple}}\\
&\{\langle Y, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle g\langle\rangle, a, (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle,\\
&\quad a\#^? g\langle\rangle, a\#^? b, a\#^? Z, a\#^? f\langle g\langle\rangle, y\rangle\} && \Longrightarrow_{\#^?\text{-function, tuple}}\\
&\{\langle Y, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle g\langle\rangle, a, (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle,\\
&\quad a\#^? g\langle\rangle, a\#^? b, a\#^? Z, a\#^? y\} && \Longrightarrow_{\#^?\text{-function, unit, atom}}\\
&\{\langle Y, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle g\langle\rangle, a, (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle,\\
&\quad a\#^? Z, a\#^? y\} && \overset{[Y\mapsto\langle g\langle\rangle, Y_1\rangle]}{\Longrightarrow}_{\approx^?\text{-widen.1}}\\
&\{\langle Y_1, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle a, (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle,\\
&\quad a\#^? Z, a\#^? y\} && \overset{[Y_1\mapsto\langle a, Y_2\rangle]}{\Longrightarrow}_{\approx^?\text{-widen.1}}\\
&\{\langle Y_2, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle,\\
&\quad a\#^? Z, a\#^? y\}.
\end{aligned}$$

Note that before the last $\approx^?$ -widen.1 rule application we should have the $\approx^?$ -proj.1 branch again leading to the failure.

Now, at this point we have several options:

(1) apply $\approx^?$ -proj.1 rule

$$\begin{aligned}
&\{\langle Y_2, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle,\\
&\quad a\#^? Z, a\#^? y\} && \overset{[Y_2\mapsto\langle\rangle]}{\Longrightarrow}_{\approx^?\text{-proj.1}}\\
&\{\langle f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle, a\#^? Z, a\#^? y\} && \overset{[Z\mapsto\langle\rangle]}{\Longrightarrow}_{\approx^?\text{-proj.2}}\\
&\{\langle f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle, a\#^? \langle\rangle, a\#^? y\} && \Longrightarrow_{\approx^?\text{-tuple, function}}\\
&\{\langle y, g\langle\rangle\rangle \approx^? \langle g\langle\rangle, (a\,b)\cdot y\rangle, a\#^? \langle\rangle, a\#^? y\} && \Longrightarrow_{\approx^?\text{-tuple}}\\
&\{y \approx^? g\langle\rangle,\ g\langle\rangle \approx^? (a\,b)\cdot y,\ \langle\rangle \approx^? \langle\rangle,\ a\#^? \langle\rangle,\ a\#^? y\} && \overset{[y\mapsto g\langle\rangle]}{\Longrightarrow}_{\approx^?\text{-var.1}}\\
&\{g\langle\rangle \approx^? g\langle\rangle, \langle\rangle \approx^? \langle\rangle,\ a\#^? \langle\rangle,\ a\#^? g\langle\rangle\} && \Longrightarrow_{\approx^?\text{-function, unit}}\\
&\{a\#^? \langle\rangle, a\#^? g\langle\rangle\} && \Longrightarrow_{\#^?\text{-function, unit}}\\
&\emptyset.
\end{aligned}$$

and we obtain the solution $(\emptyset, [X \mapsto \langle\rangle, x \mapsto g\langle\rangle, Y \mapsto \langle g\langle\rangle, a\rangle, Z \mapsto \langle\rangle, y \mapsto g\langle\rangle])$. Note, that application of $\approx^?$ -widen.2 instead of $\approx^?$ -proj.2 rule in this branch will lead to failure.

(2) applying $\approx^?$ -proj.2 rule first is similar to (1), obtaining the same solution.

(3) apply $\approx^?$ -widen.1 rule

$$
\begin{array}{ll}
\{\langle Y_2, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle (a\,b)\cdot Z, f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle, & \\
\quad a\#^? Z, a\#^? y\} & \overset{[Y_2 \mapsto \langle (a\,b)\cdot Z, Y_3\rangle]}{\Longrightarrow}_{\approx^?\text{-widen.1}} \\
\{\langle Y_3, f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle, a\#^? Z, a\#^? y\} & \overset{[Y_3 \mapsto \langle\rangle]}{\Longrightarrow}_{\approx^?\text{-proj.1}} \\
\{\langle f\langle y, g\langle\rangle\rangle\rangle \approx^? \langle f\langle g\langle\rangle, (a\,b)\cdot y\rangle\rangle, a\#^? Z, a\#^? y\} & \Longrightarrow_{\approx^?\text{-tuple, function}} \\
\{y \approx^? g\langle\rangle, g\langle\rangle \approx^? (a\,b)\cdot y, \langle\rangle \approx^? \langle\rangle, a\#^? Z, a\#^? y\} & \overset{[y \mapsto g\langle\rangle]}{\Longrightarrow}_{\approx^?\text{-var.1}} \\
\{g\langle\rangle \approx^? g\langle\rangle, \langle\rangle \approx^? \langle\rangle, a\#^? Z, a\#^? g\langle\rangle\} & \Longrightarrow_{\approx^?\text{-function, unit}} \\
\{a\#^? Z, a\#^? g\langle\rangle\} & \Longrightarrow_{\#^?\text{-function, unit}} \\
\{a\#^? Z\} & \overset{\{a\#Z\}}{\Longrightarrow}_{\#^?\text{-susp.2}} \\
\emptyset &
\end{array}
$$

and we obtain the solution $(\{a\#Z\}, [X \mapsto \langle\rangle, x \mapsto g\langle\rangle, Y \mapsto \langle g\langle\rangle, a, (a\,b)\cdot Z\rangle, y \mapsto g\langle\rangle])$. Note, that application of $\approx^?$ -widen.1 again instead of $\approx^?$ -proj.1 rule in this branch will lead to failure.

(4) applying $\approx^?$ -widen.2 rule first is similar to (3), obtaining the equivalent solution $(\{a\#Y_2\}, [X \mapsto \langle\rangle, x \mapsto g\langle\rangle, Y \mapsto \langle g\langle\rangle, a, Y_2\rangle, Z \mapsto (a\,b)\cdot Y_2, y \mapsto g\langle\rangle])$ (up to renaming of the variables).

It is clear from the example above that our algorithm is not minimal in the sense that it computes the same or equivalent solutions several times. Finding restrictions to achieve minimality is a topic for further research.

4 Terminating fragments

Strategy S helps to detect failures early, trying to avoid redundant computations. However, it can not guarantee termination, even when the solution set is finite. It is not surprising, since the strategy does not provide the decision algorithm for unranked nominal unification.

In this section we consider three special cases for which the strategy terminates. They originate from (non-nominal) unranked unification problems with finite sets of most general unifiers [27] and, hence, keep the same property for nominal unranked unification. Characterizations of termination based on freshness constraints require further investigation.

The KIF fragment. In this fragment, every occurrence of tuple variables is in the last argument of a tuple. The name originates from Knowledge Interchange Format (KIF), a language designed for representing and sharing information between disparate computer systems [13]. In KIF, the variables that correspond to our tuple variables are allowed to occur only as the last arguments in terms. This is a so-called unitary fragment: solvable unification problems have a single most general unifier. This property makes it suitable for reasoning, see, e.g., [16, 20, 30]. We can simplify the widening rules for this fragment. Instead of stepwise computation of the substitution, we can at once replace a tuple variable with the entire tuple in the other side of the equation:

$$(\approx^? \text{-widen.KIF.1}) \quad \{\langle \pi \cdot X\rangle \approx^? \langle t'_1, \dots, t'_m\rangle\} \cup P \overset{\sigma}{\Longrightarrow} P\sigma, \quad \text{where } m \geq 0,$$
$$\sigma = [X \mapsto \pi^{-1} \cdot \langle t'_1, \dots, t'_m\rangle], \text{ and } X \notin V(\langle t'_1, \dots, t'_m\rangle).$$

The second widening rule is adapted analogously, and the projection rules can be dropped as they are subsumed with these KIF-specific widening rules.

Example 4. We illustrate how the KIF-adapted rules are used to solve a unification problem in this fragment.

$$\{f\langle a.f\langle a, X\rangle, g\langle x, y, X\rangle, Y\rangle \approx^? f\langle b.f\langle b, x, Y\rangle, g\langle b, Z\rangle, U\rangle\} \quad \Longrightarrow_{\approx^?\text{-function, tuple}}$$

$$\{a.f\langle a, X\rangle \approx^? b.f\langle b, x, Y\rangle, \langle g\langle x, y, X\rangle, Y\rangle \approx^? \langle g\langle b, Z\rangle, U\rangle\} \quad \Longrightarrow_{\approx^?\text{- abst.2, tuple}}$$

$$\{f\langle a, X\rangle \approx^? f\langle a, (a\,b)\cdot x, (a\,b)\cdot Y\rangle, \langle Y\rangle \approx^? \langle U\rangle, g\langle x, y, X\rangle \approx^? g\langle b, Z\rangle,\ a\#^? f\langle b, x, Y\rangle\} \quad \Longrightarrow_{\approx^?\text{-function, tuple, atom}}$$

$$\{\langle X\rangle \approx^? \langle (a\,b)\cdot x, (a\,b)\cdot Y\rangle,\ g\langle x, y, X\rangle \approx^? g\langle b, Z\rangle, \langle Y\rangle \approx^? \langle U\rangle,\ a\#^? f\langle b, x, Y\rangle\} \quad \Longrightarrow_{\approx^?\text{-function, tuple}}$$

$$\{\langle X\rangle \approx^? \langle (a\,b)\cdot x, (a\,b)\cdot Y\rangle,\ x \approx^? b,\ \langle y, X\rangle \approx^? \langle Z\rangle, \langle Y\rangle \approx^? \langle U\rangle,\ a\#^? f\langle b, x, Y\rangle\} \quad \overset{[x\mapsto b]}{\Longrightarrow}_{\approx^?\text{-var.1}}$$

$$\{\langle X\rangle \approx^? \langle a, (a\,b)\cdot Y\rangle, \langle y, X\rangle \approx^? \langle Z\rangle, \langle Y\rangle \approx^? \langle U\rangle, a\#^? f\langle b, b, Y\rangle\} \quad \Longrightarrow_{\#^?\text{-function, tuple, atom}}$$

$$\{\langle X\rangle \approx^? \langle a, (a\,b)\cdot Y\rangle, \langle y, X\rangle \approx^? \langle Z\rangle, \langle Y\rangle \approx^? \langle U\rangle, a\#^? Y\} \quad \overset{[X\mapsto\langle a,(a\,b)\cdot Y\rangle]}{\Longrightarrow}_{\approx^?\text{-widen.KIF.1}}$$

$$\{\langle y, a, (a\,b)\cdot Y\rangle \approx^? \langle Z\rangle, \langle Y\rangle \approx^? \langle U\rangle,\ a\#^? Y\} \quad \overset{[Z\mapsto\langle y,a,(a\,b)\cdot Y\rangle]}{\Longrightarrow}_{\approx^?\text{-widen.KIF.2}}$$

$$\{\langle Y\rangle \approx^? \langle U\rangle,\ a\#^? Y\} \quad \overset{[Y\mapsto\langle U\rangle]}{\Longrightarrow}_{\approx^?\text{-widen.KIF.1}}$$

$$\{a\#^? U\} \quad \overset{\{a\# U\}}{\Longrightarrow}_{\approx^?\text{-susp.2}}$$

$\emptyset$.

Hence, the algorithm returns a most general unifier

$$(\{a\#U\}, [X \mapsto \langle a, (a\,b)\cdot U\rangle,\ x \mapsto b,\ Z \mapsto \langle y, a, (a\,b)\cdot U\rangle,\ Y \mapsto \langle U\rangle]).$$

Linear fragment. Unification problems in which no variable occurs more than once are called linear. Unlike the KIF fragment, here there are unification problems that have more than one, but still finitely many solutions. For the linear fragment we can simplify the unification rules. For instance, in the rules that eliminate variables (proj, widen, var), the substitution σ does not have to apply to the whole P, because the eliminated variable can not have any other occurrence in the remaining unification equations. It may occur only in the freshness constraints and it is sufficient to apply σ only to them. Besides, the occurrence check does not have to be performed. The $\approx^?$ -susp.1 and $\approx^?$ -susp.2 rules never apply. The following is an example of linear unranked nominal unification.

Example 5. Let the unification problem be $\{a.f\langle X, a\rangle \approx^? b.f\langle b, Y\rangle\}$. Then we have two derivations. The first one is:

$$\{a.f\langle X,a\rangle \approx^? b.f\langle b,Y\rangle\} \quad \Longrightarrow_{\approx^?\text{-abst.2, function}}$$
$$\{\langle X,a\rangle \approx^? \langle a,(a\,b)\cdot Y\rangle,\ a\#^? f\langle b,Y\rangle\} \quad \Longrightarrow_{\#^?\text{-function, tuple, atom}}$$
$$\{\langle X,a\rangle \approx^? \langle a,(a\,b)\cdot Y\rangle,\ a\#^? Y\} \quad \overset{[X\mapsto\langle\rangle]}{\Longrightarrow}_{\approx^?\text{-proj.1}}$$
$$\{\langle a\rangle \approx^? \langle a,(a\,b)\cdot Y\rangle,\ a\#^? Y\} \quad \Longrightarrow_{\approx^?\text{-tuple, atom}}$$
$$\{\langle\rangle \approx^? \langle (a\,b)\cdot Y\rangle,\ a\#^? Y\} \quad \overset{[Y\mapsto\langle\rangle]}{\Longrightarrow}_{\approx^?\text{-proj.2, unit}}$$
$$\{a\#^? \langle\rangle\} \quad \Longrightarrow_{\#^?\text{-unit}}$$
$$\emptyset.$$

It leads to the solution $(\emptyset, [X \mapsto \langle\rangle, Y \mapsto \langle\rangle])$.
The second derivation is

$$\{a.f\langle X,a\rangle \approx^? b.f\langle b,Y\rangle\} \quad \Longrightarrow_{\approx^?\text{-abst.2, function}}$$
$$\{\langle X,a\rangle \approx^? \langle a,(a\,b)\cdot Y\rangle,\ a\#^? f\langle b,Y\rangle\} \quad \Longrightarrow_{\#^?\text{-function, tuple, atom}}$$
$$\{\langle X,a\rangle \approx^? \langle a,(a\,b)\cdot Y\rangle,\ a\#^? Y\} \quad \overset{[X\mapsto\langle a,X_1\rangle]}{\Longrightarrow}_{\approx^?\text{-widen.1}}$$
$$\{\langle X_1,a\rangle \approx^? \langle (a\,b)\cdot Y\rangle,\ a\#^? Y\} \quad \overset{[Y\mapsto\langle (a\,b)\cdot X_1,(a\,b)\cdot Y_1\rangle]}{\Longrightarrow}_{\approx^?\text{-widen.2}}$$
$$\{\langle a\rangle \approx^? \langle Y_1\rangle,\ a\#^? \langle (a\,b)\cdot X_1,(a\,b)\cdot Y_1\rangle\} \quad \Longrightarrow_{\#^?\text{-tuple}}$$
$$\{\langle a\rangle \approx^? \langle Y_1\rangle,\ a\#^? (a\,b)\cdot X_1, a\#^? (a\,b)\cdot Y_1\} \quad \overset{[Y_1\mapsto\langle a,Y_2\rangle]}{\Longrightarrow}_{\approx^?\text{-widen.2}}$$
$$\{\langle\rangle \approx^? \langle Y_2\rangle,\ a\#^? (a\,b)\cdot X_1, a\#^? \langle b,(a\,b)\cdot Y_2\rangle\} \quad \overset{[Y_2\mapsto\langle\rangle]}{\Longrightarrow}_{\approx^?\text{-proj.2, unit}}$$

$$\{a\#^? (a\,b)\cdot X_1, a\#^? \langle b\rangle\} \qquad \overset{\{b\#X_1\}}{\Longrightarrow}_{\#^?\text{-susp.2, tuple, atom}}$$

$$\emptyset.$$

From this derivation, we get the second unifier $(\{b\#X_1\}, \{X \mapsto \langle a, X_1\rangle, Y \mapsto \langle (a\,b)\cdot X_1, b\rangle\})$.

Matching fragment. Matching equations are those in which variables may occur only in one side, e.g., left. In this case, we can skip the occurrence check in variable elimination rules. The $\approx^?$ -susp.1,2 and $\approx^?$ -susp.2 rules never apply. If the given problem does not contain freshness constraints, they will not appear in the result either.

Example 6. The matching problem $\{f\langle X, x, Y, (c\,d)\cdot x, Z\rangle \approx^? f\langle a, b.b, c, a.a, b, d\rangle\}$ has two solutions $(\emptyset, \{X \mapsto \langle a\rangle,\ x \mapsto b.b,\ Y \mapsto \langle c\rangle,\ Z \mapsto \langle b, d\rangle\})$ and $(\emptyset, \{X \mapsto \langle a, b.b\rangle,\ x \mapsto c,\ Y \mapsto \langle a.a, b\rangle,\ Z \mapsto \langle\rangle\})$.

Theorem 3. *The strategy S for unranked unification algorithm is terminating in the KIF, linear, and matching fragments.*

Proof. For a unification problem P, the measure of the size of P is a tuple of natural numbers $(n_x, n_X, n_\approx, n_\#)$, where n_x is the number of different individual variables occurring in P, n_X is the number of different tuple variables occurring in P, $n_\approx$ is the total size of all equational problems in P and $n_\#$ is the total size of all freshness problems in P.

n_x and n_X values are decreased by the $\approx^?$ -var rules and all $\overset{\nabla}{\Longrightarrow}$ transformations are decreasing $n_\#$. Analogously, in general, all $\overset{\sigma}{\Longrightarrow}$ transformations are decreasing $n_\approx$, except the $\approx^?$ -tuple, $\approx^?$ -widen.1 and $\approx^?$ -widen.2 rules. Clearly, $\approx^?$ -tuple can be applied only finitely many times, since tuples are finite. Next, it is easy to see that in the specific cases $\approx^?$ -widen.1 and $\approx^?$ -widen.2 rules are also decreasing $n_\approx$:

– if P is a unification problem from KIF-fragment, then $\approx^?$ -widen.1,2 and $\approx^?$ -proj.1,2 rules are replaced by $\approx^?$ -widen.KIF.1,2 which are decreasing $n_\approx$ (and n_X as well).
– if P is a unification problem from linear fragment, then every tuple variable occurs only once in P, thus $\approx^?$ -widen.1,2 rules are decreasing $n_\approx$.
– if P is a matching problem, then there is no tuple variables on the other side, thus $\approx^?$ -widen.1,2 rules are decreasing $n_\approx$ in this case as well.

□

5 Discussion

Both nominal and unranked languages are important for formalizing informal mathematical practice. In nominal languages, one can represent and reason with syntax involving explicitly named bound variables. In unranked languages, one can express and formalize variadic operators and ellipsis that are ubiquitous in mathematical practice. By bringing these two formalisms together, one can get the best of both worlds, aiming at a combination of nominal and unranked logical frameworks. Methods for solving term equations, such as unification and matching, are the core computational mechanism for deduction, rewriting, and programming in such frameworks. Our work makes a step in this direction, providing a procedure that combines nominal and unranked features.

Unranked function symbols look similar to associative function symbols with unit element (A1 symbols). The associativity axiom (for a symbol f) can be expressed as $f\langle f\langle x, y\rangle, z\rangle \approx f\langle x, f\langle y, z\rangle\rangle$ and that e is the unit element of f can be written as $f\langle x, e\rangle \approx x$ and $f\langle e, x\rangle \approx x$. Then terms with nested associative symbols can be flattened, writing, e.g., $f\langle x, y, z\rangle$ for $f\langle f\langle x, y\rangle, z\rangle$. However, in equation solving, unranked and A1 symbols behave differently. Even without nominal binders and freshness atoms, unranked unification and A1-unification are different problems, which can be illustrated with the following example:

Example 7. If f and g are unranked symbols and X is a tuple variable, then the unranked unification problem $f\langle X, g\langle X, c\rangle\rangle \approx^? f\langle a, b, g\langle a, b, c\rangle\rangle$ is solved by $\{X \mapsto \langle a, b\rangle\}$. (In this language, tuples are flat.) If we assume that f and g are A1 symbols and X is an individual variable (in A1 unification problems, tuple variables do not occur), then the same problem does not have a solution: for the left hand side, $\{X \mapsto f\langle a, b\rangle\}$ gives $f\langle a, b, g\langle f\langle a, b\rangle, c\rangle\rangle$, while $\{X \mapsto g\langle a, b\rangle\}$ leads to $f\langle g\langle a, b\rangle, g\langle a, b, c\rangle\rangle$. None of them is equal to the right hand side. Even if we assume that $\langle a, b\rangle$ is a term of this language and X can be instantiated with it, we get $f\langle\langle a, b\rangle, g\langle\langle a, b\rangle, c\rangle\rangle$ as the instance of the left hand side, which is different from $f\langle a, b, g\langle a, b, c\rangle\rangle$, since tuples are not flat in these theories.

In recent years, equational nominal unification has been investigated e.g., in [1, 3, 6, 36, 37], but associative and associative-unit theories were not among the studied ones, although α-equivalence modulo associativity has been introduced and formalized in [2] and A-matching rules were given in [9]. One can encode flat tuples via an A1 constructor in a two-sorted language as it was shown, e.g., in [22]. Combining it with nominal techniques, we would get a nominal A1-unification problem of a special kind. However, as we have already mentioned, nominal associative unification has not been investigated so far and we would still have to develop a dedicated solving procedure for this problem. (It would look very similar to our unranked nominal unification algorithm and can be easily reconstructed along the lines of the latter.) We chose not to follow that path and, instead, stick to the unranked setting. The same approach is taken, e.g., in [15], where the authors bring various reasons in favor of the unranked (thereby called flexary) representation, among them the fact that unranked representation is often more natural for implementation. As an example, they mention implementations of type theory (e.g., the Twelf logical framework [32]), where unranked representation is preferred over associative representations both internally and at the user level. Our own experience with implementing a combination of permissive nominal unification and a restricted version of sequence unification in a mathematical assistant system [23] confirms this observation.

Nominal unification problems have been extended with context variables in [38]. Relation between context and sequence unification (without nominal terms) has been studied in [25,26]. It would be interesting to find a similar connection between our work and nominal context unification from [38], but it goes beyond the scope of this paper and can be left for future investigations.

6 Conclusion

We presented an unranked nominal language as an extension of the nominal language with tuple variables and term tuples. We developed a unification procedure for solving equality and freshness problems for unranked nominal terms and proved its soundness and completeness. The soundness property is guaranteed by a specific strategy the procedure is based on. At the same time, the strategy tries to minimize redundant computations.

Some unranked nominal unification problems have an infinite set of solutions. Our procedure, as a complete method, does not terminate for them. It may also run forever for some problems with finite set of solutions, which is not surprising since the strategy is not a decision algorithm. To address this problem, we identified three practically important finitary fragments of unranked nominal unification and proved that our procedure terminates for them.

Acknowledgements

We would like to thank the anonymous referees for their useful comments that helped us to improve our work.

References

1. M. Ayala-Rincón, W. de Carvalho Segundo, M. Fernández, and D. Nantes-Sobrinho. Nominal C-unification. In F. Fioravanti and J. P. Gallagher, editors, *Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017, Namur, Belgium, October 10-12, 2017, Revised Selected Papers*, volume 10855 of *Lecture Notes in Computer Science*, pages 235–251. Springer, 2017.
2. M. Ayala-Rincón, W. de Carvalho-Segundo, M. Fernández, D. Nantes-Sobrinho, and A. C. Rocha-Oliveira. A formalisation of nominal α-equivalence with A, C, and AC function symbols. *Theoretical Computer Science*, 781:3–23, 2019.
3. M. Ayala-Rincón, M. Fernández, and D. Nantes-Sobrinho. Fixed-point constraints for nominal equational unification. In H. Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
4. M. Ayala-Rincón, M. Fernández, and D. Nantes-Sobrinho. On nominal syntax and permutation fixed points. *Log. Methods Comput. Sci.*, 16(1), 2020.
5. M. Ayala-Rincón, M. Fernández, and A. C. Rocha-Oliveira. Completeness in PVS of a nominal unification algorithm. *ENTCS*, 323(3):57–74, 2016.

6. M. Ayala-Rincón, M. Fernández, G. F. Silva, and D. Nantes-Sobrinho. A certified functional nominal C-unification algorithm. In M. Gabbrielli, editor, *Logic-Based Program Synthesis and Transformation - 29th International Symposium, LOPSTR 2019, Porto, Portugal, October 8-10, 2019, Revised Selected Papers*, volume 12042 of *Lecture Notes in Computer Science*, pages 123–138. Springer, 2019.
7. F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.
8. F. Baader and W. Snyder. Unification theory. In Robinson and Voronkov [35], pages 445–532.
9. W. de Carvalho Segundo. *Nominal Equational Problems Modulo Associativity, Commutativity and Associativity-Commutativity.* PhD thesis, Universidade de Brasília, Brazil, 2019.
10. V. Diekert. Makanin's algorithm. *Algebraic combinatorics on words*, 90:387–442, 2002.
11. G. Dowek. Higher-order unification and matching. In Robinson and Voronkov [35], pages 1009–1062.
12. M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.
13. M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format. Version 3.0. Reference Manual. Technical Report KSL-92-86, Comp. Sci. Department, Stanford University, June 1992.
14. F. Horozal. *A Framework for Defining Declarative Languages.* PhD thesis, Jacobs University Bremen, 2014.
15. F. Horozal, F. Rabe, and M. Kohlhase. Flexary operators for formalized mathematics. In S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban, editors, *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, volume 8543 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2014.
16. I. Horrocks and A. Voronkov. Reasoning support for expressive ontology languages using a theorem prover. In J. Dix and S. J. Hegner, editors, *Foundations of Information and Knowledge Systems, 4th International Symposium, FoIKS 2006, Budapest, Hungary, February 14-17, 2006, Proceedings*, volume 3861 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2006.
17. J. Jaffar. Minimal and complete word unification. *J. ACM*, 37(1):47–85, 1990.
18. J. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J. Lassez and G. D. Plotkin, editors, *Computational Logic - Essays in Honor of Alan Robinson*, pages 257–321. The MIT Press, 1991.
19. K. Knight. Unification: A multidisciplinary survey. *ACM Comput. Surv.*, 21(1):93–124, 1989.
20. T. Kutsia. Equational prover of THEOREMA. In R. Nieuwenhuis, editor, *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings*, volume 2706 of *Lecture Notes in Computer Science*, pages 367–379. Springer, 2003.
21. T. Kutsia. Solving equations involving sequence variables and sequence functions. In B. Buchberger and J. A. Campbell, editors, *AISC 2004, Proceedings*, volume 3249 of *Lecture Notes in Computer Science*, pages 157–170. Springer, 2004.
22. T. Kutsia. Solving equations with sequence variables and sequence functions. *J. Symb. Comput.*, 42(3):352–388, 2007.
23. T. Kutsia. Unification modulo alpha-equivalence in a mathematical assistant system. RISC Report Series 20-01, RISC, Johannes Kepler University Linz, 2020.

24. T. Kutsia and B. Buchberger. Predicate logic with sequence variables and sequence function symbols. In A. Asperti, G. Bancerek, and A. Trybulec, editors, *MKM 2004, Proceedings*, volume 3119 of *Lecture Notes in Computer Science*, pages 205–219. Springer, 2004.
25. T. Kutsia, J. Levy, and M. Villaret. Sequence unification through currying. In F. Baader, editor, *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2007.
26. T. Kutsia, J. Levy, and M. Villaret. On the relation between context and sequence unification. *J. Symb. Comput.*, 45(1):74–95, 2010.
27. T. Kutsia and M. Marin. Solving, reasoning, and programming in common logic. In A. Voronkov, V. Negru, T. Ida, T. Jebelean, D. Petcu, S. M. Watt, and D. Zaharie, editors, *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012, Timisoara, Romania, September 26-29, 2012*, pages 119–126. IEEE Computer Society, 2012.
28. J. Levy and M. Villaret. Nominal unification from a higher-order perspective. *ACM Trans. Comput. Log.*, 13(2):10:1–10:31, 2012.
29. G. S. Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 145(2):147–236, 1977.
30. C. Menzel. Knowledge representation, the world wide web, and the evolution of logic. *Synth.*, 182(2):269–295, 2011.
31. M. Paterson and M. N. Wegman. Linear unification. In A. K. Chandra, D. Wotschke, E. P. Friedman, and M. A. Harrison, editors, *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, pages 181–186. ACM, 1976.
32. F. Pfenning and C. Schürmann. System description: Twelf - A meta-logical framework for deductive systems. In H. Ganzinger, editor, *Automated Deduction - CADE-16, 16th International Conference on Automated Deduction, Trento, Italy, July 7-10, 1999, Proceedings*, volume 1632 of *Lecture Notes in Computer Science*, pages 202–206. Springer, 1999.
33. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Inf. Comput.*, 186(2):165–193, 2003.
34. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
35. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
36. M. Schmidt-Schauß, T. Kutsia, J. Levy, and M. Villaret. Nominal unification of higher order expressions with recursive let. In M. V. Hermenegildo and P. López-García, editors, *Logic-Based Program Synthesis and Transformation - 26th International Symposium, LOPSTR 2016, Edinburgh, UK, September 6-8, 2016, Revised Selected Papers*, volume 10184 of *Lecture Notes in Computer Science*, pages 328–344. Springer, 2016.
37. M. Schmidt-Schauß, T. Kutsia, J. Levy, M. Villaret, and Y. Kutz. Nominal unification of higher order expressions with recursive let. Frank report 62, Institut für Informatik, Goethe-Universität Frankfurt am Main, October 2019.
38. M. Schmidt-Schauß and D. Sabel. Nominal unification with atom and context variables. In H. Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 28:1–28:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
39. C. Urban, A. M. Pitts, and M. Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004.