Undecidability of Nominal Unification with Atom Substitutions

Jesús Domínguez and Maribel Fernández

Department of Informatics, King's College London, UK
{jesus.dominguez_alvarez,maribel.fernandez}@kcl.ac.uk

Abstract

We consider a syntax for nominal terms extended with non-capturing atom substitution and show that unification over such terms is undecidable. The proof of undecidability is obtained by reducing Hilbert's tenth problem to unification of extended nominal terms, adapting Goldfarb's proof of undecidability of second-order unification.

1 Introduction

Nominal unification is a well-established generalisation of first-order unification that deals with variable binding using atom permutations and freshness constraints. Nominal terms are characterised by the distinction, at the syntactic level, between atoms, which can be abstracted but are not subject to substitution, and meta-variables, or simply variables, which behave like first-order variables but may be decorated with atom permutations. Urban, Pitts and Gabbay [UPG04] showed that unification of nominal terms, that is, unification modulo α-equivalence, is decidable and gave an algorithm to find the most general solution to any (unifiable) nominal unification problem. Efficient algorithms exist which solve the nominal unification problem in polynomial time [CF08b, LV10, CF10].

Nominal unification has applications in logic programming [CU04, BF07] and term rewriting [FGM04, FG07], among others. Matching is a form of unification where one of the terms is ground, that is, without variables, or its variables are regarded as constants. Nominal matching is efficient: linear time algorithms are available [CF09]. Various versions of nominal matching have been used in nominal rewriting [FG07] and functional programming languages involving names and binders (for instance, FreshML [PG00, SPG03, Pot07], Cαml [Pot06]). However, the capture-avoiding substitution used in many systems of interest has, thus far, needed to be encoded by explicit rewrite rules or axioms for the syntax in question. To address this issue, an extension of nominal syntax with a primitive capture-avoiding atom substitution was presented in [FFST15], along with a dependent type system defined over such extension. Here we adopt this extended syntax and show that unification is undecidable by providing an effective method to reduce *Hilbert's tenth problem* to nominal unification extended with atom substitutions. The proof is based on that given by Goldfarb [Gol81] to show the undecidability of second order unification. Nominal matching extended with atom substitution remains decidable (albeit no longer unitary); an algorithm is given in [Dom17], together with a characterisation of a wide class of terms for which matching is unitary. These results have applications in rewrite-based models of computation and open the way for the development of powerful reasoning frameworks based on nominal syntax.

Related Work Our theory of extended nominal terms is taken from [FFST15]. Capture-avoiding atom substitution was previously studied in the context of nominal algebra by Gabbay and Mathijssen [GM09, GM08]. Unification and rewriting of nominal terms were introduced

in [UPG04] and [FGM04, FG07] respectively. Cheney proved that a more general form than nominal unification, called equivariant unification, is NP-complete [Che05]. Efficient nominal unification algorithms were developed by Calvès and Fernández [CF08b, CF08a] and Levy and Villaret [LV10]. Both approaches were later unified by Calvès [Cal13]. Kumar and Norrish [KN10] also studied efficient forms of nominal unification, using triangular substitutions, which are not necessarily idempotent. In [SKLV16], a nominal unification algorithm extended with a recursive let (letrec) is defined, along with several nominal letrec matching algorithms for variants, which all run in nondeterministic polynomial time. Relations between nominal unification and higher-order pattern unification were studied in [Che05, LV10, LV12, DGM10]. An explicit substitution calculus for contextual type theory is described in [AP10], which distinguishes between variables, which can be bound and have a capture-avoiding substitution, and meta-variables, which cannot be bound and for which substitution is possibly capturing. They also provide an algorithm for definitional equality.

Goldfarb [Gol81] showed that second-order unification is undecidable, by reducing Hilbert's tenth problem to a unification problem for a second-order language. We have followed his methodology to provide the proof of undecidability for nominal unification extended with atom substitutions. In [Lev96], some cases of *linear* second-order unification are proved to be decidable. Particularly the case where each variable occurs at most twice. In [Lev98, LV00] it was proved that, under the same restriction, second-order unification is undecidable, along with other decidable and undecidable subclasses of second-order unification problems with variable occurrence restrictions. Such results are obtained by means of reducing *simultaneous rigid E-unification* [GRS87], proved to be undecidable in [DV96], to second-order unification. Additionally in [LV00], there is an undecidability proof of second-order unification by a direct encoding from the halting problem for Turing machines.

2 Background

Fix countably infinite, pairwise disjoint, sets of **atoms**, $a, b, c, \ldots \in \mathcal{A}$; **variables**, $X, Y, Z, \ldots \in \mathcal{X}$; and **term-formers** $f, g, \ldots \in \mathcal{F}$, each with a fixed arity. A **permutation** is a bijection on $\mathcal{A}$ with finite domain, called the **support** of π, $\mathcal{S}upport(\pi)$. A **swapping** is a particular case, written $(a\ b)$, where a maps to b, b maps to a and all other atoms c map to themselves. **Atom substitutions** (or **a-substitutions** for short) ϑ, ϕ are mappings from atoms to terms with finite domain, that is, the set of atoms such that $\vartheta(a) \neq a$, written $\mathcal{D}om(\vartheta)$, is finite. Permutations, a-substitutions and **nominal terms with atom substitutions**, or just **terms** from now on, are generated by the grammar shown in Def. 2.1.

Definition 2.1. Syntax of terms.

$$\pi ::= \mathtt{Id} \mid \pi(a\ b) \qquad \vartheta, \phi ::= \mathtt{Id} \mid [a \mapsto s]\phi$$
$$s, t ::= a \mid \phi|\pi{\cdot}X \mid [a]s \mid fs \mid (s_1, \ldots, s_n).$$

Function applications, fs, must respect the arity of the term-former. A **moderated variable** $\phi|\pi{\cdot}X$ is $X \in \mathcal{X}$ along with a suspended permutation π and a suspended atom substitution ϕ. The final identity operator, $\mathtt{Id}$, is commonly omitted from the syntax of both permutations and a-substitutions. Write π^{-1} for the **inverse** of π. For example, if $\pi = (a\ b)(b\ c)$ then $\pi(c) = a$ and $c = \pi^{-1}(a)$. A-substitutions are interpreted as *simultaneous* bindings, abbreviated as $[a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]$. Write $\vartheta^{-a_1,\ldots,a_n}$ for the a-substitution ϑ with domain $\mathcal{D}om(\vartheta) \setminus \{a_1, \ldots, a_n\}$. The **image** of ϑ, written $\mathcal{I}mg(\vartheta)$, is the set of terms $\{\vartheta(a) \mid a \in \mathcal{D}om(\vartheta)\}$.

Write $V(t)$ for the set of variable symbols appearing in a term t; a **ground term** s is such that $V(s) = \varnothing$. Write $A(t)$ for the set of atoms appearing in a term t; this includes atoms in the domain and image of a-substitutions as well as atoms in the support of permutations. We omit the inductive definitions. Outer brackets are commonly omitted for $V(\cdot)$ and $A(\cdot)$ when applied to tuples. We also use the notation $V(\cdot)$ (resp. $A(\cdot)$) to denote the set of variables (resp. atoms) in the domain (resp. image) of an a-substitution.

Example 2.2 (Terms)**.** Let $\mathsf{map}, \mathsf{cons}$ be term-formers. We then have the terms $\mathsf{map}([a]F, \mathsf{cons}(H, T))$ and $\mathsf{cons}([a \mapsto H]\cdot F, \mathsf{map}([a]F, T))$.

For more examples, we refer the reader to [UPG04, FG07] for non-extended nominal terms and [FFST15] for the extended case.

Definition 2.3 (Permutation action)**.** Permutations apply to terms and a-substitutions. Write $\circ$ for **composition** of two permutations such that $(\pi' \circ \pi)(a) = \pi'(\pi(a))$.

$$\pi{\cdot}a \triangleq \pi(a) \qquad \pi{\cdot}[a]t \triangleq [\pi(a)]\pi{\cdot}t \qquad \pi{\cdot}ft \triangleq f\pi{\cdot}t \qquad \pi{\cdot}(t_1,\ldots,t_n) \triangleq (\pi{\cdot}t_1,\ldots,\pi{\cdot}t_n)$$
$$\pi{\cdot}(\phi|\pi'{\cdot}X) \triangleq (\pi{\cdot}\phi)\hat{}(\pi \circ \pi'){\cdot}X \text{ where} \quad \pi{\cdot}\mathtt{Id} \triangleq \mathtt{Id} \qquad \pi{\cdot}([a \mapsto t]\phi) \triangleq [\pi(a) \mapsto \pi{\cdot}t](\pi{\cdot}\phi).$$

Call $a\#t$ a **freshness constraint**. Let $\Delta, \nabla, \ldots$ range over sets of **primitive constraints** of the form $a\#X$; call such sets **freshness contexts**. Call $s \approx_\alpha t$ an **α-equivalence constraint**. Write $\nabla \vdash a\#t$ and $\nabla \vdash s \approx_\alpha t$, called **freshness** and **α-equivalence judgements** respectively, when a derivation exists using the syntax-directed rules from Def. 2.4 where, for a-substitutions ϕ, ϕ' and permutations π, π', we abbreviate $\mathcal{D}om(\phi) \cup \mathcal{D}om(\phi')$ as $\mathcal{D}omP(\phi, \phi')$ and $\mathcal{S}upport(\pi) \cup \mathcal{S}upport(\pi')$ as $\mathcal{S}upportP(\pi, \pi')$. We may drop set brackets in freshness contexts, for example, $a\#X, b\#Y$ for $\{a\#X, b\#Y\}$ and also write $a, b\#t$ (resp. $a\#s, t$) instead of $a\#t, b\#t$ (resp. $a\#s, a\#t$).

Definition 2.4 (Freshness and α-equivalence judgements)**.**

$$\frac{}{\nabla \vdash a\#b}\,(\#\mathbf{ab}) \qquad \frac{}{\nabla \vdash a\#[a]s}\,(\#[\mathbf{a}]) \qquad \frac{\nabla \vdash a\#s}{\nabla \vdash a\#[b]s}\,(\#[\mathbf{b}]) \qquad \frac{\nabla \vdash a\#s}{\nabla \vdash a\#fs}\,(\#\mathbf{f})$$

$$\frac{\bigwedge_{b \in (\mathcal{D}om(\phi) \cup \{a\})} (\nabla \vdash a\#\phi(b) \vee (\pi^{-1}(b)\#X \in \nabla))}{\nabla \vdash a\#\phi|\pi{\cdot}X}\,(\#\mathbf{X}) \qquad \frac{\nabla \vdash a\#s_1 \;\cdots\; \nabla \vdash a\#s_n}{\nabla \vdash a\#(s_1,\ldots,s_n)}\,(\#\mathbf{tupl})$$

$$\frac{}{\nabla \vdash a \approx_\alpha a}\,(\approx_\alpha\mathbf{a}) \qquad \frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash [a]s \approx_\alpha [a]t}\,(\approx_\alpha[\mathbf{a}]) \qquad \frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash fs \approx_\alpha ft}\,(\approx_\alpha\mathbf{f})$$

$$\frac{\nabla \vdash (b\ a){\cdot}s \approx_\alpha t \quad \nabla \vdash b\#s}{\nabla \vdash [a]s \approx_\alpha [b]t}\,(\approx_\alpha[\mathbf{b}]) \qquad \frac{\nabla \vdash s_1 \approx_\alpha t_1 \;\cdots\; \nabla \vdash s_n \approx_\alpha t_n}{\nabla \vdash (s_1,\ldots,s_n) \approx_\alpha (t_1,\ldots,t_n)}\,(\approx_\alpha\mathbf{tupl})$$

$$\frac{\bigwedge_{a \in (\mathcal{D}omP(\phi,\phi') \cup \mathcal{S}upportP(\pi,\pi'))} (\nabla \vdash \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)) \vee (a\#X \in \nabla))}{\nabla \vdash \phi|\pi{\cdot}X \approx_\alpha \phi'|\pi'{\cdot}X}\,(\approx_\alpha\mathbf{X}).$$

The derivation rules are similar to the ones given in [FFST15], except rules $(\#\mathbf{X})$ and $(\approx_\alpha\mathbf{X})$. In [FFST15], $(\#\mathbf{X})$ is divided into a pair of derivation rules, whereas in our version both rules are combined by the conjunction normal form (CNF), generated over the set $(\mathcal{D}om(\phi) \cup \{a\})$. Similarly, the premise of rule $(\approx_\alpha\mathbf{X})$ is here formed as a CNF instead of using the **disagreement set** $\mathtt{ds}(\nabla, \phi|\pi, \phi'|\pi') = \{a \mid a \in (\mathcal{D}omP(\phi, \phi') \cup \mathcal{S}upportP(\pi, \pi')) \wedge \nabla \not\vdash \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))\}$

as in [FFST15]. Def. 2.4 offers a more succinct approach by having fewer inference rules and avoiding the use of auxiliary functions in premises, opting instead to show the expanded form of such auxiliary functions. Recall that, unlike for non-extended nominal terms [UPG04], there may exists more than one least set of primitive constraints logically entailing $\phi|\pi{\cdot}X \approx_\alpha \phi'|\pi'{\cdot}X$ (see [FFST15]). In the case where a-substitutions are $\mathtt{Id}$, both rules $(\#\mathbf{X})$ and $(\approx_\alpha \mathbf{X})$ reduce to their non-extended analogue as given in [FG07, Def. 6].

The relation $\approx_\alpha$ is indeed an equivalence relation [FFST15].

The action of an a-substitution ϕ on a term t, written $t\phi$, relies on a freshness context ∇ and therefore define over **terms-in-context**, written $\nabla \vdash t$, or simply $\vdash t$ whenever $\nabla = \varnothing$.

Definition 2.5 (A-substitution action)**.** Write $\bullet$ for a-substitution **composition** so that $t(\phi \bullet \phi') = (t\phi)\phi'$.

$$\nabla \vdash a\phi \triangleq \phi(a) \quad \nabla \vdash (ft)\phi \triangleq ft\phi \quad \nabla \vdash (t_1,\ldots,t_n)\phi \triangleq (t_1\phi,\ldots,t_n\phi)$$
$$\nabla \vdash (\phi'|\pi{\cdot}X)\phi \triangleq (\phi' \bullet \phi)|\pi{\cdot}X \qquad \nabla \vdash ([a]t)\phi \triangleq [b]((a\ b){\cdot}t)\phi^{-b} \text{ where } \nabla \vdash b\#t, \mathcal{I}mg(\phi).$$

A-substitutions work uniformly on α-equivalence classes of terms [FFST15]. Then, following the notation above, capture-avoidance is guaranteed by selecting a distinct α-equivalent representative of $\nabla \vdash [a]t$ (i.e., $\nabla \vdash [b](a\ b){\cdot}t$ since $\nabla \vdash b\#t, \mathcal{I}mg(\phi)$) prior to ϕ acting on such term-in-context. Note that, there exists an infinite number of atoms which do not appear in either $[a]t$ or ϕ and, since variables have finite support [Pit03], one can augment the freshness context with the primitive constraints $b\#X$ for all X in $(V(t) \cup V(\mathcal{I}mg(\phi)))$ and some $b \in (\mathcal{A} \setminus (A(t) \cup A(\mathcal{I}mg(\phi))))$ whenever required. In practice, one could start with a large enough set of *new* atoms not appearing in the system under consideration in order to generate a set of primitive constraints built from such atoms with respect to each variable in the system; such a set would then be joined with the given freshness context before interacting with the system. This approach is similar to the approach taken in [FG07, FFST15] and tacitly assumed in the rest of the paper.

Variable substitutions, or just **v-substitutions**, are mappings from variables to terms with finite domain, i.e., the set of variables such that $\sigma(X) \neq X$, written $\mathcal{D}om(\sigma)$, is finite. Write $\mathcal{I}mg(\sigma)$ for the **image** of σ, that is, the set of terms $\{\sigma(X) \mid X \in \mathcal{D}om(\sigma)\}$. Variable substitutions are generated by the grammar: $\sigma, \theta ::= \mathtt{Id} \mid [X \mapsto s]\sigma$ where $\mathtt{Id}$ is commonly omitted; $[X_1 \mapsto s_1] \cdots [X_n \mapsto s_n]$ is interpreted as *simultaneous* bindings, abbreviated to $[X_1 \mapsto s_1; \ldots; X_n \mapsto s_n]$ where the variables X_i are pairwise distinct. The **action of v-substitutions** σ **on a freshness context** ∇ is defined as $\nabla\sigma = \{a\#\sigma(X) \mid X \in \mathcal{D}om(\sigma), a\#X \in \nabla\}$.

Definition 2.6 (V-substitution action)**.** Write $\bullet$ for the **composition** of two v-substitutions such that $t(\sigma \bullet \sigma') = (t\sigma)\sigma'$.

$$a\sigma \triangleq a \qquad ([a]t)\sigma \triangleq [a]t\sigma \quad (ft)\sigma \triangleq ft\sigma \quad (t_1,\ldots,t_n)\sigma \triangleq (t_1\sigma,\ldots,t_n\sigma)$$
$$(\phi|\pi{\cdot}X)\sigma \triangleq (\pi{\cdot}\sigma(X))(\phi\sigma) \text{ where } \quad \mathtt{Id}\sigma \triangleq \mathtt{Id} \quad ([a \mapsto s]\phi)\sigma \triangleq [a \mapsto s\sigma](\phi\sigma).$$

Lemma 2.7. *The action of v-substitutions σ commutes with both permutation action, $\nabla \vdash \pi{\cdot}(s\sigma) \approx_\alpha (\pi{\cdot}s)\sigma$, and a-substitution action, $\nabla \vdash (s\sigma)\vartheta\sigma \approx_\alpha (s\vartheta)\sigma$. Furthermore, v-substitution action preserves $\#$ and $\approx_\alpha$ in the following sense:*

1. *if $\Delta \vdash \nabla\sigma$ and $\nabla \vdash a\#s$, then $\Delta \vdash a\#s\sigma$;*
2. *if $\Delta \vdash \nabla\sigma$ and $\nabla \vdash s \approx_\alpha t$, then $\Delta \vdash s\sigma \approx_\alpha t\sigma$.*

Let C range over freshness and α-equivalence constraints. A **constraint problem** Cr is a finite arbitrary set of such constraints. We abbreviate $\Delta \vdash C_1, \ldots, \Delta \vdash C_n$ as $\Delta \vdash \{C_1, \ldots, C_n\}$. The actions of permutations and substitutions and the functions $V(\cdot)$ and $A(\cdot)$ extend naturally to constraints and constraint problems. The following is a corollary of Lem. 2.7.

Corollary 2.8. *For any pair of freshness contexts ∇, Δ, constraint problem Cr and v-substitution σ, such that $\Delta \vdash \nabla\sigma$ we have that, if $\nabla \vdash Cr$, then $\Delta \vdash Cr\sigma$. Similarly, $\nabla \vdash Cr$ if and only if $\nabla \vdash \pi{\cdot}Cr$.*

Definition 2.9. A **unification problem** Pr is a constraint problem Cr where α-equivalence constraints are now written as **unification constraints** $s \;{}_?{\approx}_?\; t$. A **solution** to Pr, if one exists, is a pair (Δ, σ) of a freshness context Δ and a v-substitution σ such that $\Delta \vdash Cr\sigma$. Then, call σ a **unifier** of Pr and say Pr is **unifiable**.

Write $\mathcal{U}(Pr)$ for the **set of all solutions** to a unification problem Pr. $(\Delta, \sigma) \in \mathcal{U}(Pr)$ is **more general** than $(\Delta', \sigma') \in \mathcal{U}(Pr)$, written $(\Delta, \sigma) \leq (\Delta', \sigma')$, if there exists some v-substitution θ such that $\Delta' \vdash (\sigma \circ \theta) \approx_\alpha \sigma'$and $\Delta' \vdash \Delta\theta$. Then, (Δ, σ) is a **most general solution** if there is no $(\Delta', \sigma') \in \mathcal{U}(Pr)$ such that $(\Delta', \sigma') < (\Delta, \sigma)$.

Similarly to nominal unification of non-extended terms [UPG04], solving a unification problem Pr involves finding a pair comprising of a v-substitution σ and a freshness context Δ such that $\Delta \vdash Pr\sigma$. However, unlike non-extended nominal unification, there may be more than one least set of primitive constraints Δ_i such that $\Delta_i \vdash Pr\sigma$. Moreover, in [UPG04], the *unicity* of most general solutions was shown for non-extended nominal terms. This does not hold in the presence of a-substitutions. For instance, the unification problem $\{[a \mapsto c]{\cdot}X \;{}_?{\approx}_?\; c\}$ has solutions $(\varnothing, [X \mapsto a])$ and $(\varnothing, [X \mapsto c])$ which are more general than any other solution (∇, σ) to the problem; there is no v-substitution θ such that $(\nabla, \sigma) \leq (\varnothing, [X \mapsto a])$ or $(\nabla, \sigma) \leq (\varnothing, [X \mapsto c])$. Also, neither $(\varnothing, [X \mapsto a]) \leq (\varnothing, [X \mapsto c])$ nor $(\varnothing, [X \mapsto c]) < (\varnothing, [X \mapsto a])$. Accordingly, the role of principality must be taken on, in general, by a **complete set of solutions** [BS01].

3 Undecidability of Extended Nominal Unification

To prove undecidability of extended nominal unification, we follow closely [Gol81] where *Hilbert's tenth problem*, proved undecidable in [Mat70], is used to show the undecidability of second-order unification. The main idea is to build unification problems for which a *ground unifier* simulates addition or multiplication. Then, one can use such term language to generate *Diophantine equations*.

Goldfarb numbers are inspired by Church numerals, $\lambda x.\lambda f.f(f(\ldots f(x)))$, dropping the abstraction on f so that the representation is of second-order type. In nominal terms, we denote Goldfarb numbers as tuples of atoms: $(a, (a, \ldots (a, c)))$ with n unabstracted occurrences of atom a and one occurrence of atom c represents the natural number n. Then, we abbreviate $(a, (a, \ldots (a, c)))$ as $[n, a, c]$; more generally, $[n, a, t]$ represents the expression $(a, (a, \ldots (a, t)))$ where $n \geq 0$, t is a term belonging to the language defined in Def. 3.1. Goldfarb numbers are exactly those that solve the extended nominal unification problem

$$\{(a, [c \mapsto a]{\cdot}F) \;{}_?{\approx}_?\; [c \mapsto (a, a)]{\cdot}F\}. \tag{1}$$

Definition 3.1 (Term language L)**.** Let $\mathcal{X}, \mathcal{A}$ be countable sets of variables and atoms respectively (see Section 2), and consider an empty set of function symbols. The language L contains all the nominal terms generated by the grammar given in Def. 2.1 with the exception of abstraction terms, which are not part of L. We refer to such terms as **L-terms**.

Informally, language L contains a restricted class of extended nominal terms with neither function symbols nor abstraction terms. Such constraint on the grammar of extended terms and the restriction to ground unifiers allows us to state that any unifier to Lem. 3.2 and Lem. 3.3 simulates addition and multiplication respectively. Intuitively, if unification of L-terms is proven undecidable, it is easy to see that undecidability also holds for the general case.

Following a notation closer to that of Goldfarb's term language, L-terms of form $[n, a, t]$ are now denoted as $\overline{n}_a t$ for any atom $a \in \mathcal{A}$, L-term t and integer n such that $n \geq 0$. Then, $[n, a, [m, a, t]]$ is thus denoted as $\overline{n + m}_a t$.

Observe that, to simulate addition, Goldfarb uses Church's notation for the λ-term $add = \lambda n.\lambda m.\lambda x.n(m(x))$. Below, we adapt such notation for our language L.

Lemma 3.2 (Simulating addition)**.** *Let* $Pr^+ = \{[c \mapsto F_2]{\cdot}F_1 \ {}_?{\approx}_?\ F_3\}$. *For all* $m, n, p \geq 0$, *there exists a ground unifier* θ *for* Pr^+ *such that* $\{[F_1 \mapsto \overline{n}_a c; F_2 \mapsto \overline{m}_a c; F_3 \mapsto \overline{p}_a c]\} \subseteq \theta$ *if and only if* $p = m + n$.

Next, a unification problem to simulate multiplication is defined. Once again, the lemma and its proof are closely based on their counterpart in [Gol81].

Lemma 3.3. *Let* $Pr^\times = \{s_1 \ {}_?{\approx}_?\ s_2\}$ *where*
$s_1 = [c_1 \mapsto a; c_2 \mapsto b; c_3 \mapsto ([c \mapsto a]{\cdot}F_3, ([c \mapsto b]{\cdot}F_2, c))]{\cdot}G$ *and*
$s_2 = ((a, b), [c_1 \mapsto [c \mapsto a]{\cdot}F_1; c_2 \mapsto \overline{1}_b b; c_3 \mapsto c]{\cdot}G)$.

For all $m, n, p \geq 0$, *there is a ground unifier* θ *for* $Pr^\times$ *such that* $\sigma = [F_1 \mapsto \overline{m}_a c; F_2 \mapsto \overline{n}_b c; F_3 \mapsto \overline{p}_a c]$ *and* $\sigma \subset \theta$ *if and only if* $p = m \times n$.

Finally, we prove the undecidability for extended nominal unification using the term language from Def. 3.1, Lem. 3.2 & Lem. 3.3, as follows.

Notice that every Diophantine equation of form $P(X_1, \ldots, X_n) = Q(X_1, \ldots, X_n)$ can be decomposed into a system of equations of the form indicated below (m denotes a natural number)

$$X_i + X_j = X_k, \qquad X_i \times X_j = X_k, \qquad X_i = m.$$

Now, we associate a unification problem with each such system, containing

- for each X_i, a unification problem as given in Eq. 1;
- for each $X_i + X_j = X_k$, the unification problem used to define addtion in Lem. 3.2;
- for each $X_i \times X_j = X_k$, the unification problem used to define multiplication in Lem. 3.3;
- for each $X_i = m$, the equation $X_i = \overline{m}_a c$.

We have thus an encoding of Hilbert's tenth problem.

Theorem 3.4. *There is an effective method that reduces Hilbert's tenth problem to the nominal unification problem for* L*-terms. Therefore unification of nominal terms extended with atom substitution is undecidable.*

4 Conclusion

We have shown that nominal unification is undecidable if we extend nominal syntax with a primitive, capture-avoiding atom substitution operation. Matching on the other hand is decidable: we refer to [Dom17] for a matching algorithm for extended nominal terms, which is unitary for a class of 'simple' matching problems. In future work, we will explore their use in nominal rewriting and nominal equational reasoning, and also their application to type systems and programming languages.

Acknowledgements: We thank James Cheney and Jordi Levy for pointing out Goldfarb's results and for many useful suggestions.

References

[AP10] Andreas Abel and Brigitte Pientka. Explicit substitutions for contextual type theory. In *Proceedings 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010, Edinburgh, UK, 14th July 2010.*, pages 5–20, 2010.

[BF07] William E. Byrd and Daniel Friedman. αKanren: A fresh name in nominal logic programming. In *Prooceedings of the 2007 Workshop on Scheme and Functional Programming*, pages 79–90, 2007.

[BS01] F. Baader and W. Snyder. Unification theory. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 447–533. Elsevier Science Publishers, 2001.

[Cal13] Christophe Calvès. Unifying Nominal Unification. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications (RTA 2013)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 143–157, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[CF08a] Christophe Calvès and Maribel Fernández. Nominal matching and alpha-equivalence. In *Logic, Language, Information and Computation, 15th International Workshop, WoLLIC 2008, Edinburgh, UK, July 1-4, 2008, Proceedings*, pages 111–122, 2008.

[CF08b] Christophe Calvès and Maribel Fernández. A polynomial nominal unification algorithm. *Theor. Comput. Sci.*, 403(2-3):285–306, 2008.

[CF09] Christophe Calvès and Maribel Fernández. Matching and alpha-equivalence check for nominal terms. *Journal of Computer and System Sciences*, 2009. Special issue: Selected papers from WOLLIC 2008.

[CF10] Christophe Calvès and Maribel Fernández. The first-order nominal link. In *Logic-Based Program Synthesis and Transformation - 20th International Symposium, LOPSTR 2010, Hagenberg, Austria, July 23-25, 2010, Revised Selected Papers*, pages 234–248, 2010.

[Che05] James Cheney. Relating nominal and higher-order pattern unification. In *Proceedings of UNIF 2005*, pages 104–119, 2005.

[CU04] James Cheney and Christian Urban. αprolog: A logic programming language with names, binding and α-equivalence. In *Logic Programming*, pages 269–283. Springer Berlin Heidelberg, 2004.

[DGM10] Gilles Dowek, Murdoch James Gabbay, and Dominic P. Mulligan. Permissive nominal terms and their unification: an infinite, co-infinite approach to nominal techniques. *Logic Journal of the IGPL*, 18(6):769–822, 2010.

[Dom17] Jesús Domínguez. *Rewriting formalisms with binding support: Comparing Combinatory Reduction Systems and Nominal Rewrite Systems with atom substitution.* PhD thesis, King's College London, 2017.

[DV96] Anatoli Degtyarev and Andrei Voronkov. The undecidability of simultaneous rigid e-unification. *Theor. Comput. Sci.*, 166(1&2):291–300, 1996.

[FFST15] Elliot Fairweather, Maribel Fernández, Nora Szasz, and Alvaro Tasistro. Dependent Types for Nominal Terms with Atom Substitutions. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 180–195, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[FG07] Maribel Fernández and Murdoch Gabbay. Nominal rewriting. *Inf. Comput.*, 205(6):917–965, 2007.

[FGM04] Maribel Fernández, Murdoch J. Gabbay, and Ian Mackie. Nominal rewriting systems. PPDP '04, pages 108–119, New York, NY, USA, 2004. ACM.

[GM08] Murdoch James Gabbay and Aad Mathijssen. Capture-avoiding substitution as a nominal algebra. *Formal Aspects of Computation*, 20(4-5):451–479, 2008.

[GM09] Murdoch J. Gabbay and Aad Mathijssen. Nominal universal algebra: Equational logic with names and binding. *Journal of Logic and Computation*, 2009.

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

[GRS87] Jean H. Gallier, Stan Raatz, and Wayne Snyder. Theorem proving using rigid e-unification equational matings. In *Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, June 22-25, 1987*, pages 338–346, 1987.

[KN10] Ramana Kumar and Michael Norrish. (nominal) unification by recursive descent with triangular substitutions. In *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, pages 51–66, 2010.

[Lev96] Jordi Levy. Linear second-order unification. In *Rewriting Techniques and Applications, 7th International Conference, RTA-96, New Brunswick, NJ, USA, July 27-30, 1996, Proceedings*, pages 332–346, 1996.

[Lev98] Jordi Levy. Decidable and undecidable second-order unification problems. In *Rewriting Techniques and Applications, 9th International Conference, RTA-98, Tsukuba, Japan, March 30 - April 1, 1998, Proceedings*, pages 47–60, 1998.

[LV00] Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Inf. Comput.*, 159(1-2):125–150, 2000.

[LV10] Jordi Levy and Mateu Villaret. An efficient nominal unification algorithm. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications, RTA 2010, July 11-13, 2010, Edinburgh, Scottland, UK*, pages 209–226, 2010.

[LV12] Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. *ACM Trans. Comput. Log.*, 13(2):10:1–10:31, 2012.

[Mat70] Yu. V. Matiyasevich. Enumerable sets are diophantine (in russian). *Soviet Mathematical Doklady*, 191(2):279–282, 1970.

[PG00] Andrew M. Pitts and Murdoch Gabbay. A metalanguage for programming with bound names modulo renaming. In *Mathematics of Program Construction, 5th International Conference, MPC 2000, Ponte de Lima, Portugal, July 3-5, 2000, Proceedings*, pages 230–255, 2000.

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

[Pot06] François Pottier. An overview of Cαml. *Electr. Notes Theor. Comput. Sci.*, 148(2):27–52, 2006.

[Pot07] François Pottier. Static name control for freshml. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 356–365, 2007.

[SKLV16] Manfred Schmidt-Schauß, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal unification of higher order expressions with recursive let. In *Logic-Based Program Synthesis and Transformation - 26th International Symposium, LOPSTR 2016, Edinburgh, UK, September 6-8, 2016, Revised Selected Papers*, pages 328–344, 2016.

[SPG03] Mark R. Shinwell, Andrew M. Pitts, and Murdoch James Gabbay. Freshml: programming with binders made simple. *SIGPLAN Notices*, 38(9):263–274, 2003.

[UPG04] Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004.