Meta-variables as infinite lists in nominal terms unification and rewriting

Murdoch J. Gabbay
gabbay.org.uk

Abstract

We consider the theories of nominal unification and rewriting for a new and simplified presentation of nominal terms, based on modelling moderated nominal unknowns as infinite but decidable tuples of atoms. Nominal terms α-equivalence becomes a special case of ordinary α-equivalence, definitions and proofs come closer to those of traditional syntax, proofs are simplified, and some new properties are obtained.

Key words: Nominal terms, alpha-equivalence, binding, nominal unification, nominal rewriting, nominal algebra, permissive-nominal logic, logical frameworks, foundations

Contents

1 Introduction **2**

2 Nominal terms **3**
2.1 Atoms, permutations, permission sets . . . 3
2.2 Unknowns . . . 5
2.3 Terms . . . 7
2.4 Free unknowns of a term . . . 8
2.5 Substitutions . . . 8
2.6 *shift*-permutation . . . 10
2.7 Invertible substitutions . . . 12
2.8 Occurrences . . . 13

3 Permissive nominal unification **14**
3.1 The unification algorithm . . . 14
3.2 Examples of the algorithm . . . 16
3.3 Preservation of solutions . . . 18
3.4 Simplification rewrites calculate principal solutions . . . 20

4 Rewriting **21**
4.1 Rewriting, local confluence, confluence . . . 22
4.2 Peaks, critical pairs, joinability . . . 24
4.3 Uniform rewriting . . . 25
4.4 Terminating rewrite systems . . . 26

May 17, 2012

4.5 Orthogonal rewrite systems 26

5 Closed terms **28**
5.1 Closed terms, and the equivariant extension 28
5.2 Closed rewrite rules 30

6 Conclusions **31**

1. Introduction

In this paper we simplify nominal terms, by modelling *nominal unknowns* as infinite tuples of distinct atoms. We find that by doing this, α-equivalence becomes a special case of ordinary α-equivalence; working with the syntax becomes significantly easier; and some new mathematical properties appear.

The idea of modelling nominal unknowns as infinite tuples was introduced in [28] and explored further in [23].

The emphasis in this paper is on the practical and computational aspects of this model (and it is self-contained). We address questions like: Can we compute on these terms? Is it easier to express known proofs and properties of nominal terms? Do any extra properties become true? Our answer is: mostly, yes.

Nominal terms and their unification and rewriting were introduced by the author and others in [38] and [12] as a way of studying nominal sets (called equivariant Fraenkel-Mostowski sets in [30]).

Nominal terms are intuitively 'first-order syntax plus names and binding'. In nominal terms, term-formers can bind names in their arguments but (thanks to an intended semantics in nominal sets as described in [30]) they have the flavour of first-order rather than higher-order syntax. Nominal terms unification possesses the good computational properties of first-order unification: decidability, and most general unifiers can be computed [38]. For more on unification of nominal terms see [3, 32, 4, 33].

Nominal terms also have some specific technical properties, including two levels of variable (*atoms* and *unknowns*), a capturing substitution action, a notion of state called a *freshness context*, and a non-standard notion of α-equivalence. The interested reader is referred to an extensive literature [38, 12, 6, 24].

It matters to try making nominal terms as simple as possible. Developers of nominal terms' theory, including but not limited to this author, may benefit from a new presentation of the objects of interest—especially if this leads to simpler proofs. Also potential users of nominal terms may benefit from alternative presentations, especially if they look more like what we are used to seeing in 'ordinary' syntax.

We discuss in the body of the paper how complicated our infinite tuples are to work with. We will argue that what matters is not a infinity *per se* but whether it admits a compact representation: ours will.

Convenient properties of our model

Our new model of nominal terms has convenient mathematical properties:

- α-equivalence can be defined in one line as *we quotient by binding*. See Definition 2.19. This is less detailed and less formal than in [38], but this is acceptable

because the usual theory of α-equivalence on syntax is so well-known, and by the presentation of this paper we have been able to 'plug into' it.
- Substitutions have a (in this author's opinion) beautiful characterisation as equivariant functions; the 'freshness conditions' of substitutions on nominal terms emerge as a corollary; see Lemma 2.33.
- Permutations are generated by swappings as usual—and also a *shift* permutation δ, which is new (Definition 2.4).
  δ corresponds to a de Bruijn shift function $\uparrow$ and presheaf reindexing map up, but it is invertible and we use it in a nominal context. See Lemma 2.49 and rule (**IF**) of Figure 1.
- The simplification rules for problems (Figure 1) are new.
- The treatment of closed terms is also new and can be compared with that in [12]; see Section 5.

It is also possible to extend the syntax of this paper with quantification over unknowns. Thus the syntax of Permissive-Nominal Logic [22] which has a quantifier $\forall X$ for unknowns (which in that paper were modelled as a syntactic class of symbols) can fit into an extension of the model of unknowns explored in this paper.

In brief, in this paper we present nominal terms, but abstractly and as the reader has (probably) not seen them before. The reward is a presentation that more closely imitates traditional syntax, and perhaps has better maths, easier proofs, and more theorems.

2. Nominal terms

2.1. Atoms, permutations, permission sets

Following [11] we develop a theory of *permission sets*. A permission sets splits the set of atoms into two equally sized halves: it is used like a type, controlling the free symbols in a term. Intuitively, one half is 'the atoms that are permitted free', the other half is 'the atoms that must be bound'.

We also develop a theory of permutations, which we use later to handle α-equivalence (one distinctive feature of nominal techniques is that we take permutations as primitive instead of renamings or substitutions; permutations are invertible and this turns out to be useful). Unlike what the reader may have previously seen, our permutations will not be finite. Why this is the case will become clear later.

Definition 2.1. Fix two disjoint countably infinite sets $\mathbb{A}^<$ and $\mathbb{A}^>$ of **atoms** and write

$$\mathbb{A} = \mathbb{A}^< \cup \mathbb{A}^>.$$

$a, b, c, \ldots$ will range over *distinct* elements of $\mathbb{A}$; we call this the **permutative convention**.

Remark 2.2. The reader can think of $\mathbb{A}^<$ intuitively as atoms that are 'capturable' and atoms in $\mathbb{A}^>$ as atoms that are 'not capturable'. This is reminiscent of some treatments of syntax where a formal distinction is made between 'names that exist to be bound' and 'names that exist to be free'. See for instance the *freie* and *gebundene Gegenstansvariable* of Gentzen [31, Section 1], and the *individual variables* and *parameters* of Prawitz [34, Section 1], or Smullyan [36, Chapter IV, Section 1].

An important caveat is that permutations and permission sets, developed below, can and will move atoms between these worlds. However, no permutation can move

all the atoms in $\mathbb{A}^<$ into $\mathbb{A}^>$ *at once*. So it would be misleading to examine an individual atom and ask 'is this atom capturable?'. The interest of $\mathbb{A}^<$ and $\mathbb{A}^>$ is that they guarantee enough—countably infinitely many—that we cannot exhaust supplies of either type of atom.

These key points will become clearer as the maths is developed.

Definition 2.3. Fix a bijection from non-positive integers $\{\ldots, \text{-}3, \text{-}2, \text{-}1, 0\}$ to $\mathbb{A}^<$. Fix a bijection from positive integers $\{1, 2, \ldots\}$ to $\mathbb{A}^>$. Write their combination f; this is a bijection from integers to atoms.

Definition 2.4. Given $a, b \in \mathbb{A}$ write $(a\ b)$ for the **swapping** bijection on atoms specified by:

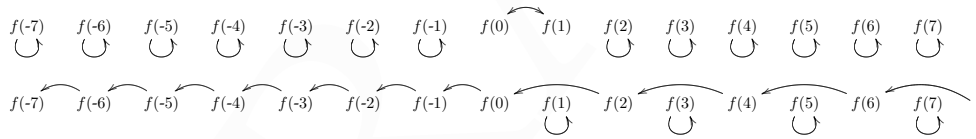
$$\begin{array}{c} a \mapsto b \\ b \mapsto a \\ c \mapsto c \end{array}$$

Recall that by our permutative convention a, b, and c are distinct. By convention $(a\ a)$ will denote the identity function on atoms id.

Write δ for the **shift** permutation which maps $f(i)$ to $f(i{-}1)$ for $i \leq 0$, and $f(2i)$ to $f(2(i{-}1))$ and $f(2i{+}1)$ to $f(2i{+}1)$ for $i \geq 1$. So:

$$\begin{array}{rl} f(i) \mapsto f(i{-}1) & i \leq 0 \\ f(2i) \mapsto f(2(i{-}1)) & i \geq 1 \\ f(2i{+}1) \mapsto f(2i{+}1) & i \geq 0 \end{array}$$

Example 2.5. We illustrate fragments of the actions of the swapping $(f(0)\ f(1))$ and δ:



Remark 2.6. Swappings are familiar from [30]. Shift permutations δ are new (though infinite permutations in general have been considered in a nominal setting in [20]).

δ has the following properties:

- There are infinitely many a such that $\delta(a) = a$, so we can always choose an atom 'fresh' for δ (we use this for instance in part 2 of Definition 2.44).
- δ bijects $\mathbb{A}^<$ with $\mathbb{A}^< \setminus \{f(0)\}$. In this sense it creates a 'fresh' atom.

If we think of $\mathbb{A}^<$ as a *namespace*—an environment of 'generated names'—and $\mathbb{A}^>$ as a space of names yet to be generated, then δ expresses that adding/deleting a name to/from the universe of countably infinitely many names, makes no difference up to bijection.

More on why δ is useful in Subsection 2.6.

Definition 2.7. Let **permutations** be generated as a group by swappings $(a\ b)$ and shift δ.[1] π and π' will range over permutations.

[1]Note that 'generated as a group' means that every element can be expressed as a *finite* compositions of the generating elements. Thus for example, δ is not generated by swappings.

Define $nontriv(\pi)$ by

$$nontriv(\pi) = \{a \mid \pi(a) \neq a\}.$$

Call π **finite** when it is in the subgroup generated by swappings. It is easy to see that π is finite if and only if $nontriv(\pi)$ is finite.

Write $\pi \circ \pi'$ for the **composition** of π and π' (so $(\pi \circ \pi')(a) = \pi(\pi'(a))$). Write id for the **identity** permutation (so $id(a) = a$ always).

Lemma 2.8. $(a\ b) \circ \delta = \delta \circ (\delta^{-1}(a)\ \delta^{-1}(b))$.

As a corollary, any π may be uniquely written as $\delta^i \circ \pi'$, and concretely represented as the pair (i, π'), for some $i \in \mathbb{Z}$ and some finite π'.

Proof. The first part is a fact of groups. The second part follows since permutations in Definition 2.7 are generated as a group by swapping permutation $(a\ b)$ and shift permutation δ. □

Definition 2.9. Give sets of atoms $A \subseteq \mathbb{A}$ the **pointwise** permutation action given by $\pi \cdot A = \{\pi(a) \mid a \in A\}$.

Definition 2.10. A **permission set** S is a set of the form $\pi \cdot \mathbb{A}^{<}$.

S, T will range over permission sets.

2.2. Unknowns

Remark 2.11. The slogan of this subsection is:

An unknown X is a well-ordering of a permission set.

Permission sets are infinite; the reader who does not care how we might represent a well-ordering inside a (finite) computing machine, or decide equality of two representations, can stop reading this subsection now, because that is all we will discuss.

The problem is that there are uncountably many such well-orderings and no way to represent them *all* in an implementation. However, some well-orderings do have good computational properties. In this subsection we exhibit a subset of the set of all well-orderings of permission sets such that:

- Each well-ordering has a compact and finite representation and equality is quickly decidable (Corollary 2.17).
- The subset is closed under the natural permutation action (this is fairly evident by construction but is pointed out explicitly after this action is defined, in Remark 2.24).
- For each permission set there are infinitely many well-orderings that are not related by any permutation (Proposition 2.18).

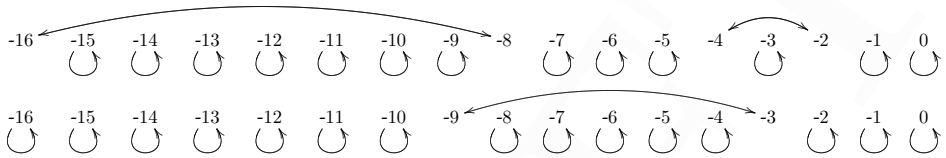
There are many such subsets; we just need to show concretely that one exists. Nothing in the rest of this paper will depend on the concrete calculations to follow.

Definition 2.12. For each prime number $p \geq 2$ define a bijection on non-positive integers ϖ_p which for every $i \geq 0$ maps $\text{-}p^{2*(i+1)}$ to $\text{-}p^{2*i+1}$ and maps $\text{-}p^{2*i+1}$ to $\text{-}p^{2*(i+1)}$, and is the identity elsewhere:

$$\begin{array}{rll} \text{-}p^{2*(i+1)} & \mapsto \text{-}p^{2*i+1} & i \geq 0 \\ \text{-}p^{2*i+1} & \mapsto \text{-}p^{2*(i+1)} & i \geq 0 \\ n & \mapsto n & \text{other } n \end{array}$$

Let ϖ range over the group generated by the ϖ_p.

Example 2.13. We illustrate initial fragments of the actions of ϖ_2 and ϖ_3:



Lemma 2.14. *Every ϖ may be represented canonically as a finite list of unique generating elements in numerical order.*

Proof. The generators biject with the prime numbers and form an involutive and commutative group. □

Definition 2.15. A **level 2 variable** or **unknown** X is a bijection from non-positive integers to a permission set $\pi{\cdot}\mathbb{A}^<$, of the form $\pi \circ f \circ \varpi$.

We may consider X as a tuple and write x_i for $X(i)$ and write $X = (x_i)_{i \leq 0}$ or even $X = (\ldots, x_{\text{-}5}, x_{\text{-}4}, x_{\text{-}3}, x_{\text{-}2}, x_{\text{-}1}, x_0)$.

Write $orb(X)$ for the unique ϖ such that $X = \pi \circ f \circ \varpi$ for some π.

X, Y, Z will range over unknowns. By convention, X and Y will range over unknowns such that $orb(X)$ and $orb(Y)$ are distinct; this is a form of **permutative convention**.

Write $pmss(X)$ for the **permission set** of X defined by $\{x_i \mid i \leq 0\}$.

Lemma 2.16. 1. *If $\pi \circ f \circ \varpi = \pi' \circ f \circ \varpi'$ then $\varpi = \varpi'$.*[2]

2. *If $\pi \circ f \circ \varpi = \pi' \circ f \circ \varpi$ then $\pi(a) = \pi'(a)$ for every $a \in \mathbb{A}^<$.*

As a corollary, $orb(X)$ is well-defined and $orb(\pi{\cdot}X) = orb(X)$.

Proof. A fact of the way we constructed the groups that π and ϖ inhabit. □

Corollary 2.17. *X may be finitely represented as a pair (π, P) where π is a permutation and P is a finite set of prime numbers. As a corollary, equality of unknowns is decidable and $orb(X)$ is quickly calculated from a reasonable finite representation using second projection.*

Proof. From Lemmas 2.8, 2.14 and 2.16. □

Proposition 2.18. *For every permission set S there are infinitely many unknowns X with distinct permutation orbits such that $pmss(X) = S$.*

[2] Having π and π' on the left is not a typo.

2.3. Terms

Definition 2.19. Fix a set of **term-formers**. $\mathsf{f}, \mathsf{g}, \mathsf{h}$ will range over distinct term-formers.

Define **(permissive nominal) terms** by:

$$r, s, t, \ldots ::= a \mid X \mid \mathsf{f}(r, \ldots, r) \mid [a]r$$

We quotient terms by α-equivalence, where $[a]r$ binds a in r and we treat X as a(n infinite) tuple. We may assume tupling term-formers and write (r_1, r_2) or $(r_1, \ldots, r_n)$.

Remark 2.20. The X in Definition 2.19 corresponds to a *moderated unknown* $\pi{\cdot}X$ from [38] or [11]. Whereas in [38] we 'pick' a set of unknowns and then attach permutations to them, in this paper we take the moderated unknowns as primitive. See [24] for a sorted permissive-nominal syntax.

Remark 2.21. *Terms can be represented using standard methods.*

We can use our favourite representation of syntax-with-binding in Definition 2.19: graphs to capture sharing; de Bruijn indexes; equivalence classes of α-equivalent terms; or nominal abstract syntax as introduced in the author's thesis [18] and subsequent work [30, 22].

This author is 'secretly' using nominal abstract syntax but the reader need not do the same.

Remark 2.22. *Terms are finite.*

Terms are conceptually infinite structures (because the conceptual model of unknowns is infinite tuples), but finite representations are easily constructed—see Subsection 2.2.

Infinities like this are familiar and routinely taken for granted: examples include '$1/3$' (whose decimal representation is infinite but which has a compact representation as $\ldots 1/3$), 'the number π', 'the α-equivalence class of the λ-term $\lambda x.x$', and 'the set of natural numbers'.

Definition 2.23. We define **free atoms** $fa(r)$ and a **permutation action** $\pi{\cdot}r$ by:

$$\begin{array}{rr} fa(a) = \{a\} & fa(\mathsf{f}(r_1, \ldots, r_n)) = \bigcup_{1 \leq i \leq n} fa(r_i) \\ fa([a]r) = fa(r) \backslash \{a\} & fa(X) = pmss(X) \\ \\ \pi{\cdot}a = \pi(a) & \pi{\cdot}\mathsf{f}(r_1, \ldots, r_n) = \mathsf{f}(\pi{\cdot}r_1, \ldots, \pi{\cdot}r_n) \\ \pi{\cdot}[a]r = [\pi(a)]\pi{\cdot}r & \pi{\cdot}X = (i \mapsto \pi(x_i))_{i \leq 0} \end{array}$$

Note the 'pointwise' nature of the clauses for $fa(X)$ and $\pi{\cdot}X$. An unknown X can be split up into 'a π and a ϖ' (see Lemma 2.16). This mirrors the moderated unknowns used in [38, 11].

Remark 2.24. If $X = \pi' \circ f \circ \varpi$ then $\pi{\cdot}X = (\pi \circ \pi') \circ f \circ \varpi$.

Lemma 2.25.
- $[a]r = [b]s$ *if and only if* $b \notin fa(r)$ *and* $(b\ a){\cdot}r = s$.
- $[a]r = [b]s$ *if and only if for fresh* c *(so* $c \notin fa(r) \cup fa(s)$*)* $(c\ a){\cdot}r = (c\ b){\cdot}s$.

Proof. By elementary calculations. (If we use nominal abstract syntax to build our nominal terms then these two characteristic equalities are, in essence, a *definition* of α-equivalence by construction. Otherwise, they are well-known lemmas [30, 22].) □

Lemma 2.26. $fa(\pi{\cdot}r) = \pi{\cdot}fa(r)$.

Proof. Routine induction on r. □

Lemma 2.27. *If* $\pi(a) = \pi'(a)$ *for all* $a \in fa(r)$ *then* $\pi{\cdot}r = \pi'{\cdot}r$.

Proof. First, we α-convert all bound atoms to be fresh for $nontriv(\pi) \cup nontriv(\pi')$; as noted in Remark 2.6 we can always do this. We then argue by induction on the size of r. We consider two cases:

- *The case of* X. It is a structural fact of the pointwise permutation acting X (Definition 2.23) that if $\pi(a) = \pi'(a)$ for all $a \in fa(X)$ then $\pi(a) = \pi'(a)$ for all a appearing in X considered as a tuple, and so $\pi{\cdot}X = \pi'{\cdot}X$.
- *The case of* $[a]r'$. From part 2 of Lemma 2.25, given that we α-converted all bound atoms to be fresh for the permutations.

□

2.4. Free unknowns of a term

Remark 2.28. Defining a notion of 'the free unknowns of r' is not entirely evident.

Consider for example $[a]X$ where $a \in pmss(X)$. If 'X appears in $[a]X$' is true then so is '$(b\ a){\cdot}X$ appears in $[a]X$' for any $b \notin pmss(X)$, since $[a]X = [b](b\ a){\cdot}X$. We deal with this in Definition 2.29 by quotienting up to all permutations. We take a more refined look at this later in Remark 2.58.

Definition 2.29. Define **free unknowns** $fv(r)$ by:

$$\begin{array}{cc} fv(a) = \varnothing & fv(\mathsf{f}(r_1, \ldots, r_n)) = fv(r_1) \cup \cdots \cup fv(r_n) \\ fv([a]r) = fv(r) & fv(X) = \{orb(X)\} \end{array}$$

By abuse of notation we write $X \in fv(r)$ for $orb(X) \in fv(r)$ and $X \notin fv(r)$ for $orb(X) \notin fv(r)$, and so forth.

Remark 2.30. $fv(r)$ may be computed as outlined in Corollary 2.17; intuitively, we just traverse the term and collect the ϖ ($orb(X)$ in the clause for $fv(X)$; see Definition 2.12).

Lemma 2.31. $fv(r)$ *is well-defined.*

Proof. From Lemma 2.25 it suffices to prove that $fv((b\ a){\cdot}r) = fv(r)$. This follows using Lemma 2.16. □

2.5. Substitutions

Definition 2.32. A **substitution** θ is a function from unknowns to terms such that

$$\forall \pi, X.\ \theta(\pi{\cdot}X) = \pi{\cdot}\theta(X).$$

We call θ **equivariant**. θ will range over substitutions.

Write id for the **identity** substitution mapping X to X always. It will always be clear whether id means the identity substitution or permutation.

The reader familiar with nominal terms will expect a 'freshness' condition on substitutions corresponding to '$\nabla' \vdash \nabla\theta$', as in for example Equation (11) or Lemma 2.14 of [38], or '$fa(\theta(X)) \subseteq pmss(X)$' as in Definition 3.1 of [11]. In the context of this paper, that condition follows as a corollary of equivariance:

Lemma 2.33. *If θ is a substitution then $fa(\theta(X)) \subseteq pmss(X)$ for all unknowns.*

Proof. Suppose there exists $a \in fa(\theta(X)) \setminus pmss(X)$. Choose fresh b (so $b \notin fa(\theta(X)) \cup pmss(X)$). Lemma 2.26 implies that $(b\ a)\cdot\theta(X) \neq \theta(X)$. Yet since $a, b \notin pmss(X)$ also $(b\ a)\cdot X = X$. Thus $\theta(X) \neq (b\ a)\cdot\theta(X) = \theta((b\ a)\cdot X) = \theta(X)$, a contradiction. □

Definition 2.34. Suppose $fa(t) \subseteq pmss(X)$. Write $[X{:=}t]$ for the substitution such that

$$[X{:=}t](\pi\cdot X) = \pi\cdot t \quad \text{and} \quad [X{:=}t](Y) = Y \quad \text{for all other } Y.$$

Definition 2.34 is a special case of Definition 5.10. However, this case is of particular interest, and simpler, so it seems seems useful (and perhaps kinder to the reader) to consider it separately.

Remark 2.35. *Unknowns from (permissive) nominal terms correspond to representatives of permutation equivalence classes.*

In this paper both X and $\pi\cdot X$ are ω-tuples of atoms with equal standing in an equivalence class of unknowns of the form $\{\pi'\cdot X \mid \pi' \text{ a permutation}\}$.[3]

When we specify $[X{:=}t]$ we have chosen a *particular* representative X. To specify the action of $[X{:=}t]$ on *all* unknowns we must consider unknowns in the permutation equivalence class of X, and unknowns not in this equivalence class. By part 1 of Lemma 2.16 the two cases of Definition 2.34 cover both possibilities and do not overlap.

The 'moderated unknown' $\pi\cdot X$ in Definition 2.34 is an artefact of our writing $[X{:=}t]$ instead of a mathematically equal $[\pi\cdot X{:=}\pi\cdot t]$ for some other π.

Since θ is constrained to be equivariant its behaviour on $\pi\cdot X$ is already determined by its behaviour on X and so we could unambiguously specify $[X{:=}t]$ succinctly just as $[X{:=}t](X) = t$ and $[X{:=}t](Y) = Y$, or even (being just a little lax) just as $[X{:=}t](X) = t$.

Definition 2.36. Define a **substitution action** on terms by:

$$a\theta = a \qquad \mathsf{f}(r_1,\ldots,r_n)\theta = \mathsf{f}(r_1\theta,\ldots,r_n\theta)$$
$$([a]r)\theta = [a](r\theta) \qquad X\theta = \theta(X)$$

Note that $X\theta$ refers to θ acting on X as a term whereas $\theta(X)$ refers the value of the function θ at X.

Lemma 2.37. $\pi\cdot(r\theta) = (\pi\cdot r)\theta$.

Proof. By a routine induction on r using equivariance. □

[3] Using Lemma 2.16 this equivalence class may also be written $\{x' \mid orb(x') = orb(X)\}$, where here x' ranges over all unknowns. (We write x' rather than X' here because by our permutative convention in Definition 2.15, X' ranges only over unknowns such that $orb(X)$ and $orb(X')$ are distinct.)

Lemma 2.38. $fa(r\theta) \subseteq fa(r)$.

Proof. By induction on r using Lemma 2.33. □

Lemma 2.39. *$r\theta$ is well-defined.*

Proof. The non-trivial part is to show that if $[a]r = [b]s$ then $([a]r)\theta = ([b]s)\theta$. Suppose $[a]r = [b]s$. By Lemma 2.25 $b \notin fa(r)$ and $(b\ a){\cdot}r = s$. By Lemma 2.38 $b \notin fa(r\theta)$. By Lemma 2.37 $(b\ a){\cdot}(r\theta) = ((b\ a){\cdot}r)\theta = s\theta$. We use Lemma 2.25 again. □

Lemma 2.40. *If $\theta(X) = \theta'(X)$ for all $X \in fv(r)$ then $r\theta = r\theta'$.*

2.6. shift-*permutation*

Remark 2.41. The reader may be familiar with the de Bruijn *shift* function $\uparrow$ [1, Subsection 2.2]. This maps $\mathbb{N}$ to $\mathbb{N} \setminus \{0\}$ by mapping $j \in \mathbb{N}$ to $j + 1 \in \mathbb{N}$, and in doing so it 'creates a fresh number' 0. The reader familiar with presheaf techniques may know of a functor δ and arrow up, which work the same way, as exemplified in [17, Section 1].

δ from Definition 2.4 follows the same general idea and does the same kind of job. It shifts 'down' instead of 'up', but $\delta^{\text{-}1}$ shifts 'up'. Note that unlike $\uparrow$ and up, δ is *invertible*, consistent with the general preference of nominal techniques for using permutations on atoms.

All permission sets can be bijected with one another by a permutation. In particular, for every permission set S and atom a there exists some π such that $S \setminus \{a\} = \pi{\cdot}S$. See Corollary 2.52. This would not be the case if we only admitted finite permutations. We use this in rule (**IF**) of Figure 1 to eliminate an atom from the support of an unknown.

Remark 2.42. The algorithm from [11, Section 6] used a notion of the 'known unknowns' $\mathcal{V}$ to do the job of δ. This introduced a notion of state and sequentiality into the algorithm. The original algorithm from [38] also avoided sequentiality, by using freshness constraints $a\#t$.

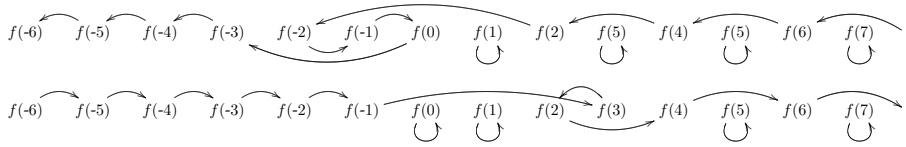
Very broadly speaking, the reader can translate '$a\#t$' in [38] to δ; the use of δ in the unification algorithm of Section 3, parallels the use of $a\#t$ in [38].

Notation 2.43. Recall the bijection f from integers to atoms from Definition 2.3. By abuse of notation write 0 for the atom $f(0)$.

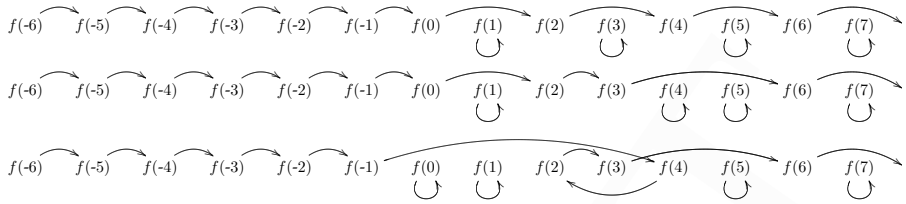
Definition 2.44.

1. If $a \in \mathbb{A}^{<}$ then define $\delta^{\text{-}a}$ by:
$$\delta^{\text{-}a} = (a\,0) \circ \delta \circ (a\,0)$$
2. If $b \in \mathbb{A}^{>}$ then for some fixed but arbitrary choice of $c \in \mathbb{A}^{>}$ such that $\delta(c) = c$ (and so also $c \notin \mathbb{A}^{<}$), define δ^{+b} by:
$$\delta^{+b} = (b\,0) \circ (c\,b) \circ \delta^{\text{-}1} \circ (c\,b) \circ (b\,0)$$

Example 2.45. We illustrate $\delta^{\text{-}a}$ and δ^{+b} where $a = f(\text{-}2)$ and $b = f(3)$ and where we take $c = b$:



We also consider the slightly more complex example of δ^{+d} where $d = f(4)$, and again we take $c = f(3)$. We do this in three steps, where we illustrate δ^{-1}, then $(c\ d) \circ \delta^{-1} \circ (c\ d)$, and finally δ^{+d}:



Lemma 2.46. 1. *If $a \in \mathbb{A}^<$ then δ^{-a} bijects $\mathbb{A}^<$ with $\mathbb{A}^< \setminus \{a\}$.*
2. *If $b \in \mathbb{A}^>$ then δ^{+b} bijects $\mathbb{A}^<$ with $\mathbb{A}^< \cup \{b\}$.*

Proof. For the first part, suppose $a \in \mathbb{A}^<$. Then $\mathbb{A}^< = (a\ 0){\cdot}\mathbb{A}^<$. We reason as follows:

$$\delta^{-a}{\cdot}((a\ 0){\cdot}\mathbb{A}^<) = ((a\ 0) \circ \delta \circ (a\ 0) \circ (a\ 0)){\cdot}\mathbb{A}^< = ((a\ 0) \circ \delta){\cdot}\mathbb{A}^< = (a\ 0){\cdot}(\mathbb{A}^<\setminus\{0\}) = \mathbb{A}^<\setminus\{a\}$$

Now suppose $b \in \mathbb{A}^>$. It is easier to work with $(\delta^{+b})^{-1}$, to keep the parallel with the previous case. So $\mathbb{A}^< \cup \{b\} = ((b\ 0){\cdot}\mathbb{A}^<) \cup \{0\}$. We reason as follows:

$$\begin{array}{lll}
(\delta^{+b})^{-1}{\cdot}(((b\ 0){\cdot}\mathbb{A}^<)\cup\{0\}) = ((b\ 0) \circ (c\ b) \circ \delta \circ (c\ b) \circ (b\ 0)){\cdot}(((b\ 0){\cdot}\mathbb{A}^<) \cup \{0\}) & \text{(Def. 2.44)} \\
= \big(((b\ 0) \circ (c\ b) \circ \delta \circ (c\ b) \circ (b\ 0)){\cdot}((b\ 0){\cdot}\mathbb{A}^<)\big) \cup \{0\} & (\delta(c){=}c) \\
= \big(((b\ 0) \circ (c\ b) \circ \delta \circ (c\ b)){\cdot}\mathbb{A}^<\big) \cup \{0\} & \text{(Fact)} \\
= \big(((b\ 0) \circ (c\ b) \circ \delta){\cdot}\mathbb{A}^<\big) \cup \{0\} & (b, c \notin \mathbb{A}^<) \\
= \big(((b\ 0) \circ (c\ b)){\cdot}(\mathbb{A}^<\setminus\{0\})\big) \cup \{0\} & (\delta{\cdot}\mathbb{A}^<{=}\mathbb{A}^<\setminus\{0\}) \\
= (\mathbb{A}^<\setminus\{0\}) \cup \{0\} & (b, c \notin \mathbb{A}^<\setminus\{0\}) \\
= \mathbb{A}^<
\end{array}$$

□

Recall from Definition 2.10 that each permission set S has the form $\pi{\cdot}\mathbb{A}^<$ for some permutation π.

Definition 2.47. For each S make some choice of permutation π_S such that $S = \pi_S^{-1}{\cdot}\mathbb{A}^<$.[4]

Definition 2.48. Suppose S is a permission set and $a \in S$ and $b \notin S$. Then we define:

$$\delta_{S-a} = \pi_S^{-1} \circ \delta^{-\pi_S(a)} \circ \pi_S \qquad \delta_{S+b} = \pi_S^{-1} \circ \delta^{+\pi_S(b)} \circ \pi_S$$

The concrete details of the construction are only interesting insofar as they give us Lemma 2.49. Other permutations are possible, but we only need that one exists.

Lemma 2.49. 1. *δ_{S-a} bijects S with $S\setminus\{a\}$.*
2. *δ_{S+b} bijects S with $S\cup\{b\}$.*

Proof. From Lemma 2.46. □

[4]Taking the inverse here saves writing $^{-1}$ quite so many times in Definition 2.48, and is harmless since permutations are invertible.

Definition 2.50. Suppose S is a permission set and $pmss(X) = S$. Suppose D is a finite list of atoms $d_1, \ldots, d_n$ and E is a finite list of atoms $e_1, \ldots, e_n$. Suppose $\{d_1, \ldots, d_n\} \subseteq S$ and $\{e_1, \ldots, e_n\} \cap S = \varnothing$. Then define $\delta_{S\text{-}D}$ and δ_{S+E}, and $X\text{-}D$ and $X{+}E$ by:

$$\begin{array}{rlrl} \delta_{S\text{-}[]} &= id & \delta_{S+[]} &= id \\ \delta_{S\text{-}[d]} &= \delta_{S\text{-}d} & \delta_{S+[e]} &= \delta_{S+e} \\ \delta_{S\text{-}d,D} &= \delta_{(S\setminus\{d\})\text{-}D} \circ \delta_{S\text{-}d} & \delta_{S+e,E} &= \delta_{(S\cup\{e\})+E} \circ \delta_{S+e} \\ X\text{-}D &= \delta_{pmss(X)\text{-}D} \cdot X & X{+}E &= \delta_{pmss(X)+E} \cdot X \end{array}$$

Lemma 2.51. *Suppose S is a permission set. Suppose D and E are finite lists of atoms $d_1, \ldots, d_n$ and $e_1, \ldots, e_n$. Suppose $\{d_1, \ldots, d_n\} \subseteq S$ and $\{e_1, \ldots, e_n\} \cap S = \varnothing$.*

Then $\delta_{S\text{-}D}$ bijects S with $S \setminus \{d_1, \ldots, d_n\}$ and δ_{S+E} bijects S with $S \cup \{e_1, \ldots, e_n\}$.

Proof. Using Lemma 2.49. □

Corollary 2.52. *S is a permission set if and only if $S = (\mathbb{A}^< \setminus A) \cup B$ for some finite $A \subseteq \mathbb{A}^<$ and $B \subseteq \mathbb{A}^>$.*

Proof. If S is a permission set then by Definition 2.10 $S = \pi \cdot \mathbb{A}^<$ for some π and the result follows by a routine induction on the generators of π (swapping and δ; see Definition 2.7).

Conversely consider $S = (\mathbb{A}^< \setminus A) \cup B$. Let D be the atoms in A in some order, and E be the atoms in B in some order. Then we apply $\delta_{S\text{-}D}$ and then $\delta_{(S\setminus A)+E}$ and use Lemma 2.51. □

2.7. Invertible substitutions

In this brief subsection we discuss some simple constructions which will be useful later.

Definition 2.53. Define **composition** of substitutions $\theta_1 \circ \theta_2$ by

$$(\theta_1 \circ \theta_2)(X) = (\theta_1(X))\theta_2.$$

Lemma 2.54. $(r\theta)\theta' = r(\theta \circ \theta')$.

Proof. By a routine induction on r, we consider one case.[5]

- *The case X.* We reason as follows:

$$\begin{array}{rll} X(\theta \circ \theta') &= (\theta \circ \theta')(X) & \text{Definition 2.36} \\ &= (\theta(X))\theta' & \text{Definition 2.53} \\ &= (X\theta)\theta' & \text{Definition 2.36} \end{array}$$

□

Definition 2.55. Call θ **invertible** when there exists $\theta^{\text{-}1}$ such that $\theta \circ \theta^{\text{-}1} = \theta^{\text{-}1} \circ \theta = id$.

[5] The interested reader might compare this with the corresponding case in the proof of [11, Lemma 3.8], also using a permissive-nominal syntax but without taking meta-variables as infinite lists.

Lemma 2.56. *θ is invertible if and only if θ is a bijection on the set of all unknowns. Furthermore, if θ is invertible then $pmss(\theta(X)) = pmss(X)$ always.*

Proof. Substitution cannot make syntax smaller, or (by Lemma 2.38) make free atoms larger. □

Proposition 2.57. *Given any finite set of unknowns $\mathcal{X}$ there exists an invertible substitution θ such that for every $X \in \mathcal{X}$ it is the case that $orb(\theta(X)) \notin \{orb(X') \mid X' \in \mathcal{X}\}$.*

Proof. Using Proposition 2.18. □

So—just as for atoms—we can always rename unknowns to 'be fresh'. The reader might note that we do not actually need θ to be invertible (it suffices for it to be injective on $\mathcal{X}$). However, the invertible construction is no harder and is also a special case of a more abstract framework explored elsewhere [23].

2.8. Occurrences

Remark 2.58. As discussed in Remark 2.28 we have to be careful if we wish to say 'X appears in r'; this might not quite mean what we think it does.

For example if 'X appears in $[a]X$' where $a \in pmss(X)$ then also '$(b\ a)\cdot X$ appears in $[a]X$' for any $b \notin pmss(X)$, since $[a]X = [b](b\ a)\cdot X$.

We dealt with this in Definition 2.29 by quotienting out all permutations. But this is a little drastic.

For instance, '$(b\ a)\cdot X$ appears in $[a]X$' is not true for $b \in pmss(X)$; it is not the case that if 'X appears in r' then '$\pi\cdot X$ appears in r' for any π.

We did not need to quotient out *all* permutations—only some of them—and so returning $orb(X)$ in Definition 2.29 throws out more information than necessary.

Definitions 2.59 and 2.60 develop a more refined notion of occurrence, based on an intuition of 'X appears in r under a list of abstractions D'. This will be useful later.

Definition 2.59. D will range over finite lists of distinct atoms. A **(level 2) occurrence** is a term of the form $[D]X$ where $[]X$ is X and $[a, D]X$ is $[a][D]X$.

Definition 2.60. Define the **occurrences in** r inductively by:

$$\begin{array}{cc} occ(a) = \varnothing & occ(X) = X \\ occ(\mathsf{f}(r_1, \ldots, r_n)) = \bigcup occ(r_i) & occ([a]r) = \{[a, D]X \mid [D]X \in occ(r),\ a \notin D\} \end{array}$$

Example 2.61.

- X occurs in X.
- $[a]X$ occurs in $[a]X$ and also in $[a](X, Y)$; so does $[a]Y$. X does not occur in $[a]X$ or $[a](X, Y)$.
- Suppose c is fresh (so $c \notin \mathbb{A}^<$). Then $[a][b]X$ and $[a][c](c\ a)\cdot X$ occur in $[a]([b]X, [a]X)$. We quotient terms by α-equivalence, so $[a][c](c\ a)\cdot X$ is equal to $[a][a]X$ and thus $[a][a]X$ occurs in $[a]([b]X, [a]X)$.

3. Permissive nominal unification

We now revisit material from nominal unification [38] and permissive-nominal unification [11]. The algorithm here is in the same spirit as this previous work, but there are significant differences:

- Solutions are equality constraints.

  Solutions to nominal unification problems from [38] include also freshness constraints $a\#r$. Their job is done here by permission sets; this is inherited from the permissive-nominal algorithm in [11].

- Our algorithm does not keep track of a context of 'known' unknowns.

  The permissive-nominal unification algorithm in [11] needed to keep track of a context of 'known' unknowns written $\mathcal{V}$; see for instance Figure 5 of [11]. The job of $\mathcal{V}$ is done here by δ. We find this gives easier proofs.

The main definition of this section is Definition 3.8. The main result is Theorem 3.25.

3.1. The unification algorithm

Definition 3.1. An **equality** is a unordered pair $r \stackrel{?}{=} s$ (so $r \stackrel{?}{=} s$ is identical to $s \stackrel{?}{=} r$) such that:

1. $sort(r) = sort(s)$.
2. If $[D]X$ and $[D']\pi{\cdot}X$ are both in $occ(r) \cup occ(s)$ then π is finite.
   So we exclude an equality like $X \stackrel{?}{=} \delta{\cdot}X$, where δ is a shift permutation and $nontriv(\delta) \cap pmss(X)$ is not finite.

A **freshness** is an ordered pair $a\#_?r$.

Let ef range over equalities or freshnesses and define $ef\theta$ by:

$$(r \stackrel{?}{=} s)\theta = (r\theta \stackrel{?}{=} s\theta)$$
$$(a\#_?r)\theta = (a\#_?(r\theta))$$

A **unification problem** Pr is a finite list $ef_1, \ldots, ef_n$.

We (ab)use standard sets notation and write $ef \in Pr$ as shorthand for 'ef appears in the list Pr'.

Remark 3.2. Condition 2 in Definition 3.1 protects ($\stackrel{?}{=}$**X**) in Figure 1 from an 'infinite freshness explosion', if $nontriv(\pi) \cap pmss(X)$ is not finite. This condition exists implicitly in [38], in the sense that all permutations there are finite. We discuss the implications of this condition to nominal rewriting, at the end of Section 5.

Definition 3.3. If $Pr = ef_1, \ldots, ef_n$ is a problem then define $Pr\theta$ by:

$$Pr\theta = ef_1\theta, \ldots, ef_n\theta$$

Say θ **solves** Pr when

$r\theta = s\theta$	for every	$r \stackrel{?}{=} s \in Pr$,	and
$a \notin fa(r\theta)$	for every	$a\#_?r \in Pr$.	

Write $Sol(Pr)$ for the set of solutions to Pr and call Pr **solvable** when $Sol(Pr)$ is non-empty.

$$
\begin{array}{lll}
(\stackrel{?}{=}\mathbf{a}) & a \stackrel{?}{=} a,\ Pr & \Longrightarrow \quad Pr \\
(\stackrel{?}{=}\mathsf{f}) & \mathsf{f}(r_1,\ldots) \stackrel{?}{=} \mathsf{f}(s_1,\ldots),\ Pr & \Longrightarrow \quad r_1 \stackrel{?}{=} s_1,\ \ldots,\ Pr \\
(\stackrel{?}{=}[]) & [a]r \stackrel{?}{=} [a]s,\ Pr & \Longrightarrow \quad r \stackrel{?}{=} s,\ Pr \\
(\stackrel{?}{=}\mathbf{X}) & X \stackrel{?}{=} \pi{\cdot}X,\ Pr & \Longrightarrow \quad a_1\#_? X, \ldots, a_n\#_? X, Pr \\
 & & \qquad (\{a_1,\ldots,a_n\} = \mathit{nontriv}(\pi) \cap \mathrm{supp}(X)) \\
(\mathbf{F}) & r \stackrel{?}{=} X,\ Pr & \Longrightarrow \quad a\#_? r,\ r \stackrel{?}{=} X,\ Pr \qquad (a \in \mathit{fa}(r) \backslash \mathit{pmss}(X)) \\
(\mathbf{F}\#) & a\#_? r,\ Pr & \Longrightarrow \quad Pr \qquad\qquad\qquad\qquad\quad (a \notin \mathit{fa}(r)) \\
(\mathbf{F}\mathsf{f}) & a\#_? \mathsf{f}(r_1,\ldots,r_n),\ Pr & \Longrightarrow \quad a\#_? r_1, \ldots, a\#_? r_n, Pr \\
(\mathbf{F}[]) & a\#_? [b]r, Pr & \Longrightarrow \quad a\#_? r,\ Pr \\
(\mathbf{IE}) & r \stackrel{?}{=} X,\ Pr & \stackrel{[X:=r]}{\Longrightarrow} \quad Pr[X:=r] \qquad (X \notin \mathit{fv}(r),\ \mathit{fa}(r) \subseteq \mathit{pmss}(X)) \\
(\mathbf{IF}) & a\#_? X, Pr & \stackrel{[X:=\delta_{\mathit{pmss}(X)\text{-}a}\cdot X]}{\Longrightarrow} \quad Pr[X:=\delta_{\mathit{pmss}(X)\text{-}a}\cdot X]
\end{array}
$$

Figure 1: Simplification rules for problems

Lemma 3.4. *$\theta \circ \theta' \in \mathit{Sol}(Pr)$ if and only if $\theta' \in \mathit{Sol}(Pr\theta)$.*

Proof. By unpacking Definition 3.3 and using Lemma 2.54. □

Definition 3.5. Define a **simplification** rewrite relation $Pr \Longrightarrow Pr'$ on unification problems by the rules in Figure 1.

We call rules (**IF**) and (**IE**) **instantiating rules**. We call all the other rules **non-instantiating rules**.

Write $\Longrightarrow^*$ for the transitive and reflexive closure of $\Longrightarrow$.

Remark 3.6. The instantiating rule (**IF**) preserves condition 2 of Definition 3.1 because it applies $[X{:=}\delta_{\mathit{pmss}(X)\text{-}a}{\cdot}X]$ uniformly to all occurrences involving X. Condition 2 does not outlaw δ, it just outlaws attaching it to one occurrence of X *and not another*.

Remark 3.7. Unlike was the case in [38], there is no separate simplification rule for $[a]r \stackrel{?}{=} [b]s$. This is because we have quotiented syntax by α-equivalence, so a and b can both be renamed to some fresh c. This was not possible in [38] because there, α-equivalence takes place in a freshness context (a set of freshness assumptions on unknowns) whereas here it does not (instead of freshness contexts we have permission sets, which are static and fixed).

A reader might be now be tempted to comment "so the syntax of [38] is lower-level than the syntax of this paper". Not necessarily so. An implementation of nominal syntax might well seek a representation in which bound atoms really are nameless.

Definition 3.8. If Pr is a problem, define a **unification algorithm** by:

1. Rewrite Pr using the rules of Definition 3.5 where possible, with top-down precedence (so apply $(\stackrel{?}{=}\mathbf{a})$ before $(\stackrel{?}{=}\mathsf{f})$, and so on).
2. If we reduce to $\varnothing$ then we succeed and return θ where θ is the composition of all the substitutions labelling rewrites (we take $\theta = \mathit{id}$ if there are none). Otherwise, we fail.

Remark 3.9. In Definition 3.8, we apply each rule to the head of the list Pr. This is to prevent 'unfair' looping, e.g. repeatedly applying $(\mathbf{F})$ to some equality $r \stackrel{?}{=} X$ wherever it appears in Pr.

We note in passing that the rule $(\mathbf{F}\#)$ is equivalent—in the presence of the other rules—to a pair of rules $a\#_? b, Pr \Longrightarrow Pr$ and $a\#_? X, Pr \Longrightarrow Pr$ if $a \notin pmss(X)$.

Proposition 3.10. *The algorithm of Definition 3.8 always terminates.*

Proof. It is not hard to generate an inductive quantity which is reduced by the reductions in Figure 1. □

Remark 3.11. In [33] it is suggested to implement freshness conditions in nominal unification using equality conditions of the form $[a]r \stackrel{?}{=} [b]s$. Freshness *can* be approached using equality alone, as was also noted in Theorem 5.5 from [26] and Lemma 4.51 from [27]. In the context of nominal unification the observation of [33] is that doing so avoids having to 'write algorithms twice'.

This technique is not so directly applicable here, because $[a]r \stackrel{?}{=} [b]s$ is identical to $[c](c\ a)\cdot r \stackrel{?}{=} [c](c\ b)\cdot s$ for fresh c, because we already quotiented by α-equivalence. This equality in turn has the same value as $(c\ a)\cdot r \stackrel{?}{=} (c\ b)\cdot s$—technically, this happens via $([]\stackrel{?}{=})$.

This suggests that we might do something slightly different from [33] but in the same spirit, and also in the same spirit as Theorem 5.5 from [26]—which is no coincidence since both are related to the definition of freshness using the И-quantifier from [30]: we could encode $a\#_? r$ as $r \stackrel{?}{=} (b\ a)\cdot r$ for $b \notin pmss(r)$.

However, this would not really simplify our algorithm here. The design of $(\mathbf{F})$ relies on generating freshness conditions which are then solved; if we lost freshness conditions then to avoid looping we would have to 'hard-wire' recognition of equality-constraints-that-are-really-freshness-conditions into the algorithm instead. The gain from doing this is not clear.

Freshness conditions on syntax really do seem to be independently useful. This is not absolute—we could eliminate them—but they 'want' to be there, and are natural in the algorithms.

3.2. *Examples of the algorithm*

Recall the definition of X-D from Definition 2.48.

Example one (succeeds).

Suppose $a, c \in \mathbb{A}^<$ and $d \notin \mathbb{A}^<$. Take $pmss(X) = \mathbb{A}^<$ and suppose a term-former g. We apply the algorithm to $\{\mathsf{g}([a]X, [a]a) \stackrel{?}{=} \mathsf{g}([d]c, [d]d)\}$:

$$
\begin{array}{lcl}
\mathsf{g}([a]X, [a]a) \stackrel{?}{=} \mathsf{g}([d]c, [d]d) & \Longrightarrow & (\stackrel{?}{=}\mathsf{g}), (\stackrel{?}{=}()) \\
[a]X \stackrel{?}{=} [d]c\ , [a]a \stackrel{?}{=} [d]d & \Longrightarrow & (\stackrel{?}{=}[]), [a]X = [d](d\ a)\cdot X \\
(d\ a)\cdot X \stackrel{?}{=} c\ , [a]a \stackrel{?}{=} [d]d & \stackrel{[X:=c]}{\Longrightarrow} & (\mathbf{IE}) \\
[a]a \stackrel{?}{=} [d]d & \Longrightarrow & (\stackrel{?}{=}[]), [a]a = [d]d \\
d \stackrel{?}{=} d & \Longrightarrow & (\stackrel{?}{=}\mathbf{a}) \\
\varnothing \quad \text{Success, with } [X{:=}c] & &
\end{array}
$$

Example two (succeeds).

Suppose $a, c \in \mathbb{A}^<$ and $b, d \notin \mathbb{A}^<$. Take $pmss(X) = \mathbb{A}^< \cup \{b, d\}$, $pmss(Y) = \mathbb{A}^< \cup \{b'\}$, and $pmss(Z) = \mathbb{A}^<$. Suppose a term-former f.

We apply the algorithm to $\{\mathsf{f}([a]b, Z, X) \stackrel{?}{=} \mathsf{f}([d]b, [a]a, Y)\}$:

$$\begin{array}{lll}
\mathsf{f}([a]b, Z, X) \stackrel{?}{=} \mathsf{f}([d]b, [a]a, Y) & \Longrightarrow & (\stackrel{?}{=}\mathsf{f}), (\stackrel{?}{=}()) \\
[a]b \stackrel{?}{=} [d]b\ ,\ Z \stackrel{?}{=} [a]a,\ X \stackrel{?}{=} Y & \Longrightarrow & (\stackrel{?}{=}[]), [a]b = [d]b \\
b \stackrel{?}{=} b\ ,\ Z \stackrel{?}{=} [a]a,\ X \stackrel{?}{=} Y & \Longrightarrow & (\stackrel{?}{=}\mathbf{a}) \\
Z \stackrel{?}{=} [a]a\ ,\ X \stackrel{?}{=} Y & \stackrel{[Z:=[a]a]}{\Longrightarrow} & (\mathbf{IE}) \\
X \stackrel{?}{=} Y & \Longrightarrow & (\mathbf{F}) \\
b\#_?X\ ,\ X \stackrel{?}{=} Y & \stackrel{[X:=X\text{-}b]}{\Longrightarrow} & (\mathbf{IF}) \\
X\text{-}b \stackrel{?}{=} Y & \Longrightarrow & (\mathbf{F}) \\
d\#_?X\text{-}b\ ,\ X\text{-}b \stackrel{?}{=} Y & \stackrel{[X\text{-}b:=X\text{-}b,d]}{\Longrightarrow} & (\mathbf{IF}) \\
X\text{-}b, d \stackrel{?}{=} Y & \Longrightarrow & (\mathbf{F}) \\
b'\#_?Y\ ,\ X\text{-}b, d \stackrel{?}{=} Y & \stackrel{[Y:=Y\text{-}b']}{\Longrightarrow} & (\mathbf{IF}) \\
X\text{-}b, d \stackrel{?}{=} Y\text{-}b' & \stackrel{[Y\text{-}b':=X\text{-}b,d]}{\Longrightarrow} & (\mathbf{IE}) \\
\varnothing & \text{Success, with } [X{:=}X\text{-}b, d,\ Y{:=}X\text{-}b, d,\ Z{:=}[a]a] &
\end{array}$$

Example three (fails).

Take $pmss(X) = \mathbb{A}^<$. We run the algorithm on $\{[a][b]X \stackrel{?}{=} [a]X\}$:

$$\begin{array}{lll}
[a][b]X \stackrel{?}{=} [a]X & \Longrightarrow & (\stackrel{?}{=}[]) \\
[b]X \stackrel{?}{=} X & \text{Failure} &
\end{array}$$

The algorithm fails because the precondition of rule (**IE**), $X \notin fv([b]X)$ is not satisfied.

Example four (succeeds).

Take $pmss(X) = \mathbb{A}^<$ and take $a, b \in \mathbb{A}^<$. We run the algorithm on $\{X \stackrel{?}{=} (a\ b)\cdot X\}$:

$$\begin{array}{lll}
X \stackrel{?}{=} (a\ b)\cdot X & \Longrightarrow & (\stackrel{?}{=}\mathbf{X}) \\
a\#_?X\ ,\ b\#_?X & \stackrel{[X:=X\text{-}a]}{\Longrightarrow} & (\mathbf{IF}) \\
b\#_?X\text{-}a & \stackrel{[X\text{-}a:=(X\text{-}a)\text{-}b]}{\Longrightarrow} & \\
\varnothing & \text{Success, with } [X{:=}(X\text{-}a)\text{-}b] &
\end{array}$$

Later we will prove Theorem 3.25, which tells us that failure here implies that no solution to the unification problem exists.

3.3. Preservation of solutions

3.3.1. ... under non-instantiating rules

Lemma 3.12. *If $Pr \implies Pr'$ by a non-instantiating rule (Definition 3.5) then $Sol(Pr) = Sol(Pr')$.*

Proof. The empty set cannot be simplified, so suppose $Pr = r \stackrel{?}{=} s, Pr'$ where the simplification rule acts on $r \stackrel{?}{=} s$. We consider two cases:

- *The case* ($\stackrel{?}{=}[]$). Suppose $Pr = [a]r \stackrel{?}{=} [a]s, Pr'$ and $[a]r \stackrel{?}{=} [a]s, Pr' \implies r \stackrel{?}{=} s, Pr'$ by ($\stackrel{?}{=}[]$). By Definition 2.36 and properties of equality, $[a](r\theta) = [a](s\theta)$ if and only if $r\theta = s\theta$.
- *The case* ($\mathbf{F}\mathsf{f}$). Suppose $Pr = a\#_?\mathsf{f}(r_1, \ldots, r_n), Pr'$ and $a\#_?\mathsf{f}(r_1, \ldots, r_n), Pr' \implies a\#_?r_1, \ldots, a\#_?r_n, Pr'$ by ($\mathbf{F}\mathsf{f}$). By Definitions 2.36 and 2.23, $a \notin fa(\mathsf{f}(r_1, \ldots, r_n)\theta)$ if and only if $a \notin fa(r_1\theta), \ldots, a \notin fa(r_n\theta)$.
- *The case* ($\mathbf{F}$). Suppose $Pr = r \stackrel{?}{=} X, Pr'$ and $a \in pmss(X) \setminus fa(r)$, and $r \stackrel{?}{=} X, Pr' \implies a\#_?r, r \stackrel{?}{=} X, Pr'$ by ($\mathbf{F}$). Now if θ solves $r \stackrel{?}{=} X$ then $\theta(X) = r\theta$. By Lemma 2.38 $fa(r\theta) \subseteq pmss(X)$ and so in particular θ also solves $a\#_?r$. The result follows.

□

Lemma 3.13. *$r\theta = r\theta'$ if and only if $\forall X \in fv(r).\theta(X) = \theta'(X)$.*

Proof. By a routine induction on r. We consider two cases:

- *The case* $[a]r$. Suppose $\theta(X) = \theta'(X)$ for every $X \in fv([a]r)$. $fv([a]r) = fv(r)$ so by inductive hypothesis $r\theta = r\theta'$. The result follows from the definitions.
  The reverse implication is similar.
- *The case* X. Suppose $\theta(\pi{\cdot}X) = \theta'(\pi{\cdot}X)$ for all π. Then taking $\pi = id$ we have $X\theta = \theta(X) = \theta'(X) = X\theta'$.
  Conversely if $X\theta = X\theta'$ then by equivariance (Definition 2.32) $\theta(\pi{\cdot}X) = \theta'(\pi{\cdot}X)$ for all π.

□

Remark 3.14. Recall from Definition 2.29 that we write $X \in fv(r)$ for $orb(X) \in fv(r)$. It may seem in Lemma 3.13 that the condition $\forall X \in fv(r).\theta(X) = \theta'(X)$ would require checking $\theta(X) = \theta'(X)$ for infinitely many X provided that $fv(r) \neq \varnothing$. In fact, this is not the case: by equivariance of θ, we only need to check equality for finitely many representative X.

Lemma 3.15. *Suppose $\theta(X) = \theta'(X)$ for all $X \in fv(Pr)$. Then $\theta \in Sol(Pr)$ if and only if $\theta' \in Sol(Pr)$.*

Proof. From Definition 3.3 it suffices to show that $r\theta = s\theta$ if and only if $r\theta' = s\theta'$, for every $(r \stackrel{?}{=} s) \in Pr$, and $a \notin fa(r\theta)$ if and only if $a \notin fa(r\theta')$, for every $(a\#_?r) \in Pr$. This is immediate using Lemma 3.13. □

3.3.2. . . . under (**IE**)

Recall from Remark 2.35 the discussion of why we write $\pi{\cdot}X$ when we have chosen a representative element X of an equivalence class of unknowns under permutations.

Definition 3.16. Write $\theta{-}X$ for the substitution such that

$$\begin{array}{rl} (\theta{-}X)(\pi{\cdot}X) = \pi{\cdot}X & \text{and} \\ (\theta{-}X)(Y) = \theta(Y) & \text{for all other } Y. \end{array}$$

In the right circumstances, a substitution θ can be factored as 'a part of θ that does not touch X' and 'a single substitution for X':

Theorem 3.17. *If $X\theta = s\theta$ and $X \notin fv(s)$ then $\theta = [X{:=}s]{\circ}(\theta{-}X)$. That is,*

$$\theta(X) = X([X{:=}s]{\circ}(\theta{-}X)) \quad \textit{and} \quad \theta(Y) = Y([X{:=}s]{\circ}(\theta{-}X)).$$

Proof. We reason as follows:

$$\begin{array}{rll} (\pi{\cdot}X)([X{:=}s]{\circ}(\theta{-}X)) = (\pi{\cdot}s)(\theta{-}X) & & \text{Definition 2.36, Lemma 2.54} \\ = (\pi{\cdot}s)\theta & & X{\notin}fv(s),\ \text{Lemma 3.13} \\ = (\pi{\cdot}X)\theta & & \text{Assumption} \\ Y([X{:=}s]{\circ}(\theta{-}X)) = Y(\theta{-}X) & & \text{Definition 2.36, Lemma 2.54} \\ = Y\theta & & \text{Definition 3.16} \end{array}$$

□

3.3.3. . . . under (**IF**)

Definition 3.18. Suppose θ is a substitution. Suppose $a \in pmss(X)$ and $a \notin fa(\theta(X))$. Define a substitution $\theta_{[X\text{-}a:=X]}(X)$ by:

- $(\theta_{[X\text{-}a:=X]})(\pi{\cdot}X) = (\pi \circ \delta^{-1}_{X\text{-}a}){\cdot}\theta(X)$.
- $(\theta_{[X\text{-}a:=X]})(Y) = \theta(Y)$ for all other Y.

It is routine to verify that Definition 3.18 is well-defined and a substitution.

Theorem 3.19. *Suppose $a \in pmss(X)$ and $a \notin fa(\theta(X))$. Then*

$$\theta(\pi{\cdot}X) = ([X{:=}X\text{-}a]{\circ}(\theta_{[X\text{-}a:=X]}))(\pi{\cdot}X).$$

Proof. We unpack definitions:

$$\begin{array}{rll} ([X{:=}X\text{-}a]{\circ}(\theta_{[X\text{-}a:=X]}))(\pi{\cdot}X) = (\pi{\cdot}(X\text{-}a))\theta_{[X\text{-}a:=X]} & & \text{Definition 2.53} \\ = ((\pi{\circ}\delta_{X\text{-}a}){\cdot}X)\theta_{[X\text{-}a:=X]} & & \text{Definition 2.50} \\ = (\pi \circ \delta_{X\text{-}a} \circ \delta^{-1}_{X\text{-}a}){\cdot}X & & \text{Definition 3.18} \\ = \pi{\cdot}X & & \text{Fact of the group action} \end{array}$$

□

3.4. Simplification rewrites calculate principal solutions

Definition 3.20. Write $\theta_1 \leq \theta_2$ when there exists some θ' such that $X\theta_2 = X(\theta_1 \circ \theta')$ always. Call $\leq$ the **instantiation ordering**.

Definition 3.21. A **principal** (or **most general**) solution to a problem Pr is a solution $\theta \in Sol(Pr)$ such that $\theta \leq \theta'$ for all other $\theta' \in Sol(Pr)$.

Our main result is Theorem 3.24: the unification algorithm from Definition 3.8 calculates a principal solution.

Lemma 3.22. *If $\theta_1 \leq \theta_2$ then $\theta \circ \theta_1 \leq \theta \circ \theta_2$.*

Proof. By Definition 3.20, θ' exists such that $X\theta_2 = X(\theta_1 \circ \theta')$ always. Then:

$$\begin{aligned} X(\theta \circ \theta_2) &= (X\theta)\theta_2 && \text{Lemma 2.54} \\ &= (X\theta)(\theta_1 \circ \theta') && \text{Lemma 3.13} \\ &= X((\theta \circ \theta_1) \circ \theta') && \text{Lemma 2.54} \end{aligned}$$

□

Lemma 3.23. 1. *Suppose $fa(s) \subseteq pmss(X)$ and $X \notin fv(s)$. Write $\chi = [X := s]$. If $Pr \overset{\chi}{\Longrightarrow} Pr'$ with* (**IE**) *then $\theta \in Sol(Pr)$ implies $\theta - X \in Sol(Pr')$.*

2. *Suppose $a \in pmss(X)$. Write $\rho = [X := X\text{-}a]$. If $Pr \overset{\rho}{\Longrightarrow} Pr'$ with* (**IF**) *then $\theta \in Sol(Pr)$ implies $\theta_{[X\text{-}a:=X]} \in Sol(Pr')$.*

Proof. 1. Suppose $Pr = X \overset{?}{=} s,\ Pr''$ so that $X \overset{?}{=} s,\ Pr'' \overset{\chi}{\Longrightarrow} Pr''\chi$. Now suppose $\theta \in Sol(Pr)$. By Theorem 3.17 $\chi \circ (\theta - X) \in Sol(Pr)$. By Lemma 3.4, $\theta - X \in Sol(Pr\chi)$. It follows that $\theta - X \in Sol(Pr''\chi)$ as required.

2. Suppose $Pr = a \#_? X,\ Pr''$ and $a \in pmss(X)$ so that $Pr \overset{\rho}{\Longrightarrow} Pr\rho$. Now suppose $\theta \in Sol(Pr)$. By Theorem 3.19 $\rho \circ \theta_{[X\text{-}a:=X]} \in Sol(Pr)$. By Lemma 3.4, $\theta_{[X\text{-}a:=X]} \in Sol(Pr\rho)$ as required.

□

Theorem 3.24. *If $Pr \overset{\theta}{\Longrightarrow}{}^* \varnothing$ then θ is a principal solution to Pr (Definition 3.21).*

Proof. By induction on the path of $Pr \overset{\theta}{\Longrightarrow}{}^* \varnothing$.

- *The empty path.* So $Pr = \varnothing$ and $\theta = id$. By Definition 3.20, $id \leq \theta'$.
- *The non-instantiating case.* Suppose

$$Pr \Longrightarrow Pr' \overset{\theta}{\Longrightarrow}{}^* \varnothing$$

where $Pr \Longrightarrow Pr'$ by a non-instantiating rule. By inductive hypothesis θ is a principal solution of Pr'. It follows from Lemma 3.12 that θ is also a principal solution of Pr.

- *The case* (**IE**). Suppose $fa(r) \subseteq pmss(X)$ and $X \notin fv(r)$. Write $\chi = [X{:=}r]$. Suppose $Pr = r \stackrel{?}{=} X, Pr''$ so that

$$r \stackrel{?}{=} X,\ Pr'' \stackrel{\chi}{\Longrightarrow} Pr''\chi \stackrel{\theta''}{\Longrightarrow}{}^{*}\ \varnothing.$$

  Further, consider any other $\theta' \in Sol(Pr)$.
  By Lemma 3.23 $(\theta' {-} X) \in Sol(Pr''\chi)$ and by inductive hypothesis $\theta'' \in Sol(Pr''\chi)$ and $\theta'' \leq \theta' {-} X$. By Lemma 3.22, $\chi \circ \theta'' \leq \chi \circ (\theta' {-} X)$. By Theorem 3.17 $\chi \circ (\theta' {-} X) = \theta'$.
- *The case* (**IF**). Suppose $a \in pmss(X)$. Write $\rho = [X{:=}X\text{-}a]$, so that

$$Pr \stackrel{\rho}{\Longrightarrow} Pr\rho \stackrel{\theta''}{\Longrightarrow}{}^{*}\ \varnothing,$$

  Further, consider any other $\theta' \in Sol(Pr)$.
  By Lemma 3.23, $\theta'_{[X\text{-}a:=X]} \in Sol(Pr\rho)$ and by inductive hypothesis $\theta'' \in Sol(Pr\rho)$ and $\theta'' \leq \theta'_{[X\text{-}a:=X]}$. By Lemma 3.22, $\rho \circ \theta'' \leq \rho \circ \theta'_{[X\text{-}a:=X]}$. By Theorem 3.19 $\rho \circ \theta'_{[X\text{-}a:=X]} = \theta'$.

□

Theorem 3.25. *Given a problem Pr, if the algorithm of Definition 3.8 succeeds then it returns a principal solution; if it fails then there is no solution.*

Proof. If the algorithm succeeds we use Theorem 3.24. Otherwise, the algorithm generates an element of the form $\mathsf{f}(r_1, \ldots, r_n) \stackrel{?}{=} \mathsf{f}(r'_1, \ldots, r'_{n'})$ where $n \neq n'$, $\mathsf{f}(\ldots) \stackrel{?}{=} \mathsf{g}(\ldots)$, $\mathsf{f}(\ldots) \stackrel{?}{=} [a]s$, $\mathsf{f}(\ldots) \stackrel{?}{=} a$, $[a]r = a$, $[a]r = b$, $a \stackrel{?}{=} b$, $a \#_? a$, or $X \stackrel{?}{=} s$ where $X \in fv(s)$. By arguments on syntax and size of syntax, no solution to the reduced problem exists. It follows by Lemma 3.23 that no solution to Pr exists. □

Definition 3.26. Fix terms r and s.

- Call **nominal unification** the problem of finding a θ to make $r\theta = s\theta$.
- Call **nominal matching** the problem of finding a θ to make $r\theta = s$.

Corollary 3.27. *Nominal unification and nominal matching are decidable.*

Proof. An algorithm for unification is sketched in Definition 3.8; furthermore by Theorem 3.25 it calculates a most general θ which represents all other solutions.

For matching, by Proposition 2.57 we can invertibly substitute unknowns in r so they are disjoint from unknowns in s and run the unification algorithm with the modification that it should not trigger (**IE**) or (**IF**) for $X \in fv(r)$. It is not hard to see that this calculates a most general matching solution. □

4. Rewriting

Nominal rewriting was introduced in [14, 12]. The use of nominal terms allows us to write rewrite rules for systems with binding, such as the λ-calculus (Example 4.2). The material from [12] simplifies (for completeness and to enable meaningful comparison, we sketch full proofs).

Definition 4.1. A **rewrite rule** is a pair $l \to m$ such that $fv(m) \subseteq fv(l)$. R will range over rewrite rules.
A **rewrite theory** R is a (possibly infinite) set of rewrite rules.

The notion of rewrite rule and rewrite theory in Definition 4.1 is much like the first-order case, but because of the 'nominal' aspects of our syntax we can handle names and binding.

Example 4.2. We write down a signature and some rewrite rules for the untyped λ-calculus. Assume a name-sort ν and a base sort ι and term-formers $\mathsf{lam} : ([\nu]\iota)\iota$, $\mathsf{app} : (\iota,\iota)\iota$, and $\mathsf{var} : (\nu)\iota$. Sugar $\mathsf{lam}([a]r)$ to $\lambda a.r$ and $\mathsf{app}(r',r)$ to $r'r$ and $\mathsf{var}(a)$ to a.

An axiom for η-reduction is:

$$\lambda a.(Za) \to Z \quad (a \notin pmss(Z)) \tag{η}$$

Here is that same η-equivalence axiom, written out as it would be informally:

$$\lambda x.(tx) \to t \quad \text{if } x \notin fv(t)$$

The reader can see how similar they look written out on the page, but one is written in a formal syntax, and the other is not. Note how permission sets are used to avoid or permit capture.

If we sugar $(\lambda a.r)r'$ to $r[a{\mapsto}r']$ then rewrites for β-reduction are:

$$\begin{array}{rll} a[a{\mapsto}Y] & \to Y & \\ Z[a{\mapsto}X] & \to Z & (a \notin pmss(Z)) \\ (X'X)[a{\mapsto}Y] & \to (X'[a{\mapsto}Y])(X[a{\mapsto}Y]) & \\ (\lambda a.X)[b{\mapsto}Z] & \to \lambda a.(X[b{\mapsto}Z]) & (a \notin pmss(Z)) \\ X[a{\mapsto}a] & \to X & \end{array}$$

There is no rule for the general case of $Z[a{\mapsto}X]$ where $a \in pmss(Z)$—this does not reduce. Similarly in informal practice we cannot say anything about the reductions of the schema of terms represented by $(\lambda a.t)r$ without knowing any further particulars of what t and r range over. For more on this example and others like it, see Examples 5.1.3 and 7.1.3 of [24].[6]

We now set about defining the rewrites generated by a nominal rewrite theory.

[6] In Remark 2.11 our slogan was that a nominal unknown is an infinite list of distinct atoms. An extension of this slogan is to allow unknowns to be an infinite list of distinct atoms, except for finitely many entries which are allowed to be terms. We do not do that in this paper because we are interested in modelling α-equivalence without atoms-substitution as primitive. If we did allow this, then our model of syntax would be appropriate to permit an atoms-substitution action $[a{:=}t]$ as primitive (the disadvantage of this is that it would greatly complicate the theories of rewriting, unification, and algebra—but sometimes we just want substitution around). In any case, if we did this then the atoms-substitution would be able to act on Z where $a \in pmss(Z)$, by actually replacing a in Z by t. We return to this briefly in the Conclusions.

Definition 4.3. Define the terms s in which X occurs **only once** by:

$$s ::= \pi{\cdot}X \mid [a]s \mid \mathsf{f}(r_1,\ldots,r_{i-1},s,r_{i+1},\ldots,r_n) \qquad (X \notin fv(r_1),\ldots,fv(r_{i-1}),fv(r_{i+1}),\ldots,fv(r_n))$$

A **position** C is a pair (s,X) of a nominal term and an unknown X which occurs only once in s.

Remark 4.4. In Definition 4.3, $\pi{\cdot}X$ denotes an unknown in the same permutation equivalence class as X. For example:

- X occurs only once in $\pi{\cdot}X$.
- $\pi{\cdot}X$ occurs only once in X (recall that $\pi{\cdot}X$ is a well-ordering of $\pi{\cdot}pmss(X)$).
- X and $\pi{\cdot}X$ do not occur in a Y such that $orb(Y) \neq orb(X)$.

Notation 4.5. If $C=(s,X)$ is a position write $pmss(C)$ for $pmss(X)$.
If $fa(r) \subseteq pmss(C)$ (so that $[X{:=}r]$ is a substitution) write $C[r]$ for $s[X{:=}r]$.

Definition 4.6. The **one-step rewrite relation** $r \xrightarrow{\mathsf{R}} s$ is the least relation such that for every $(l \to m) \in \mathsf{R}$, position C, and substitution θ, if $fa(l\theta) \cup fa(m\theta) \subseteq pmss(C)$ (so that $C[l\theta]$ and $C[m\theta]$ are well-defined) then

$$C[l\theta] \xrightarrow{R} C[m\theta].$$

The **multi-step rewrite relation** $r \xrightarrow{\mathsf{R}}{}^{*} s$ is the reflexive transitive closure of the one-step rewrite relation.

Remark 4.7. Note that $r \xrightarrow{R} s$ implies $\pi{\cdot}r \xrightarrow{R} \pi{\cdot}s$ holds by construction because of the $\pi{\cdot}X$ in Definition 4.3.

Example 4.8. Recall the rule (η) from Example 4.2: $\lambda a.(Za) \to Z$ where $a \notin pmss(Z)$.

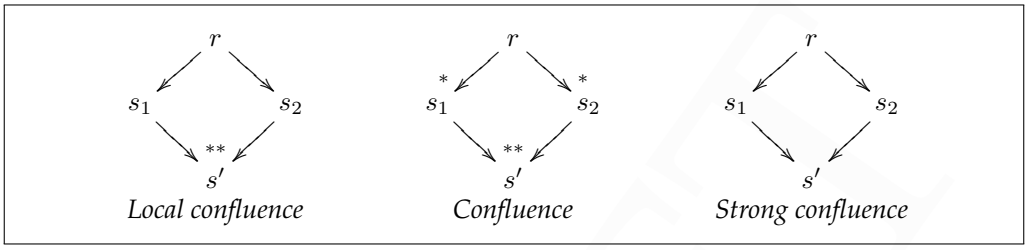
- To deduce $\lambda a.(ba) \to b$ where $b \notin pmss(Z)$ we take $C = ((b\ c){\cdot}Z, Z)$ for some $c \in pmss(Z)$ and we take $\theta = [Z{:=}c]$.
- To deduce $\lambda a'.(ba') \to b$ we *also* take $C = ((b\ c){\cdot}Z, Z)$ and $\theta = [Z{:=}c]$. This is because $\lambda a'.(ba')$ and $\lambda a.(ba)$ are α-equivalent and we (use nominal abstract syntax or) quotient syntax by α-equivalence.
- To deduce $\lambda a.(Za) \to Z$ we take $C=(Z,Z)$ and $\theta = id$.
- Suppose $pmss(Z') = pmss(Z) \cup \{a\}$. To deduce $\lambda a.(Z'a) \to Z'$ we take $C = ((\delta_{Z'a})^{-1}{\cdot}Z, Z)$ and $\theta = [Z{:=}\delta_{Z'a}{\cdot}Z']$.
- We cannot deduce $\lambda a.(aa) \to a$, because $[Z{:=}a]$ is not a substitution: no function mapping Z to a can be equivariant, since $(b\ a){\cdot}Z = Z$ but $(b\ a){\cdot}a = b \neq a$ (also $a \notin pmss(Z)$: see Lemma 2.33).

Some terminology will be useful later, so we introduce it now in one place:

Definition 4.9.
- Call R **locally confluent** when $r \xrightarrow{\mathsf{R}} s_1$ and $r \xrightarrow{\mathsf{R}} s_2$ implies there exists some s' such that $s_1 \xrightarrow{\mathsf{R}}{}^{*} s'$ and $s_2 \xrightarrow{\mathsf{R}}{}^{*} s'$.

- Call R **confluent** when $r \overset{\mathsf{R}}{\to}{}^* s_1$ and $r \overset{\mathsf{R}}{\to}{}^* s_2$ implies there exists some s' such that $s_1 \overset{\mathsf{R}}{\to}{}^* s'$ and $s_2 \overset{\mathsf{R}}{\to}{}^* s'$.

We illustrate this below.



4.2. Peaks, critical pairs, joinability

We now investigate criteria for deducing confluence of nominal rewrite systems.

Definition 4.10. Consider two rewrite rules $R_1 = (l_1 \to m_1)$ and $R_2 = (l_2 \to m_2)$. Call R_1 a **copy** of R_2 when there exists an invertible substitution θ (Definition 2.55) such that $(l_2\theta \to m_2\theta) = R_1$.

Clearly, if R_1 is a copy of R_2 then R_2 is also a copy of R_1. Furthermore:

Lemma 4.11. *If R_1 and R_2 are copies of the same rule then $l \xrightarrow{R_1} m$ if and only if $l \xrightarrow{R_2} m$.*

Proof. Unpacking Definition 4.6 and exploiting the existence of an inverse θ^{-1}. □

Definition 4.12. Write $r \to s_1, s_2$ when $r \to s_1$ and $r \to s_2$ and call this a **peak**. Call this peak **joinable** when there exists a t such that $s_1 \to^* t$ and $s_2 \to^* t$.

So R is locally confluent when every peak is joinable.

Definition 4.13. Suppose that $R_i = (l_i \to m_i)$ for $i = 1, 2$ and $fv(R_1) \cap fv(R_2) = \varnothing$. Suppose $l_1 = L[l_1']$ for some l_1' and $l_1' \stackrel{?}{=} l_2$ has a principal solution θ. Call the pair $(m_1\theta, L[m_2]\theta)$ a **critical pair**.

Call $(m_1\theta, L[m_2]\theta)$ **trivial** when at least one of the following hold:

1. $L = (\pi{\cdot}X, X)$ and R_1 and R_2 are copies of the same rule.
2. $l_1' = X$ for some unknown X.

Lemma 4.14. *Peaks that are instances of trivial critical pairs, are not always joinable.*

Proof. It suffices to provide a counterexample. Fix term-formers $\mathsf{0}$ and f and take $R_1 = (\mathsf{0} \to a)$ and $R_2 = (X \to \mathsf{f}(a))$ where $a \notin pmss(X)$.

There is a critical pair $(a, \mathsf{f}(a))$ between R_1 and R_2.

Also, $\mathsf{0} \xrightarrow{R_1} a$ and $\mathsf{0} \xrightarrow{R_2} \mathsf{f}(a)$ and it is a fact that this peak cannot be joined—we 'want' to close this peak by rewriting a to $\mathsf{f}(a)$ using R_2, but the fact that $a \notin pmss(X)$ blocks this. □

Remark 4.15. In nominal rewriting from [12] it was not in general the case that if $\Delta \vdash r \approx_\alpha r'$ and $\Delta \vdash r \xrightarrow{R} s$ then $\Delta \vdash r' \xrightarrow{R} s$ (see the end of Subsection 5.2 in [12]).

That has become irrelevant in this paper, because of our different treatment of α-equivalence.

The subtle point here is not that we quotient by α-equivalence, though we do; but that we *can* quotient by α-equivalence. Conversely, the point about [12] is not that we did not quotient; it was that we *could not* quotient, as reflected by the fact that rewriting did not respect α-equivalence.

The notion of rewriting of this paper corresponds more closely to rewriting in an arbitrarily extended freshness context, as developed in [13]; see for example Theorem 4.4 of [13].

4.3. Uniform rewriting

Definition 4.16. Call a rule $R = (l \to m)$ **uniform** when $fa(m) \subseteq fa(l)$. Call a rewrite theory R uniform when every $R \in \mathsf{R}$ is uniform.

Definition 4.16 mirrors the condition in Definition 4.1 that $fv(m) \subseteq fv(l)$, but for atoms instead of unknowns.

Lemma 4.17. *If $fa(m) \subseteq fa(l)$ then $fa(C[m]) \subseteq fa(C[l])$.*

Proof. Routine induction using Lemma 2.26 and Definition 2.23. □

Corollary 4.18. *$R = (l \to m)$ is uniform if and only if $\forall r,s.r \xrightarrow{R} s \Rightarrow fa(s) \subseteq fa(r)$.*

Proof. From Lemmas 2.26 and 2.38. □

Lemma 4.19. *Suppose $R = (l \to m)$ is uniform and $X \notin fv(R)$. Suppose $\theta(X) = l\theta$. Specify θ' by $\theta'(\pi{\cdot}X) = \pi{\cdot}(m\theta)$ and $\theta'(Y) = \theta(Y)$. Then $r\theta \to^* r\theta'$ for any r.*

Proof. θ' is a substitution by Lemmas 2.38 and 2.26. The result follows by a routine induction on r. □

Theorem 4.20. *If a rewrite theory R (Definition 4.1) is uniform then peaks that are instances of trivial critical pairs, are joinable.*

Proof. Consider two rules $R_i = (l_i \to m_i) \in \mathsf{R}$ for $i = 1, 2$. Without loss of generality we suppose the unknowns of R_1 are disjoint from those of R_2. Suppose they have a critical pair $(m_1\theta, L[m_2]\theta)$. That is, there exists l_1' such that $l_1 = L[l_1']$ and θ is a principal solution to $l_1' \stackrel{?}{=} l_2$.

There are two cases:

- *The case $L = (\pi{\cdot}X, X)$ and R_1 and R_2 are copies of the same rule $l \to m$.* The peak we want to join is $l_1\theta = \pi{\cdot}l_2\theta \to m_1\theta,\ \pi{\cdot}m_2\theta$, where the rules $l_1 \to m_1$ and $l_2 \to m_2$ are identical aside from their free variables which are renamed disjoint. We use Lemma 3.13 and the assumption in Definition 4.1 that $fv(m) \subseteq fv(l)$.

- *The case $L = (m_1\theta, L[m_2]\theta)$ where $l_1 = L[X]$ and $\theta(X) = l_2$.* Specify θ' by $\theta'(\pi{\cdot}X) = \pi{\cdot}m_2$ and $\theta'(Y) = \theta(Y)$ for all other Y; note that θ' *is* a substitution since $fa(m_2) \subseteq fa(l_2)$ by uniformity and $fa(l_2) \subseteq pmss(X)$ by our assumption that θ is a substitution.
  By Lemma 4.19 $m_1\theta \to m_1\theta'$. By definition $L[m_2]\theta = l_1\theta' \xrightarrow{R_1} m_1\theta'$, so we have joined the peak. □

Theorem 4.21. *Suppose* R *is uniform and all non-trivial critical pairs of* R *are joinable. Then* R *is locally confluent.*

Proof. Suppose $r \xrightarrow{R_1} s_1$ and $r \xrightarrow{R_2} s_2$. Write L_1 and L_2 for the positions at which the two rewrites occur.

If L_1 and L_2 identify distinct subterms of r then local confluence holds by a standard diagrammatic argument (see for instance [2]).

Otherwise it must be that $L_2 = (L_1[L], X)$; that is, L_2 identifies a point in r beneath the point identified by L_1 (or the symmetric case that $L_1 = (L_2[L], X)$, which is similar and we elide). There are now three possibilities:

1. X in L_2 replaces an unknown in r. This is an instance of a trivial critical pair; we use Theorem 4.20.
2. $L = (\pi{\cdot}X, X)$ and R_1 and R_2 are copies of the same rule. Then again this is an instance of a trivial critical pair and we use Theorem 4.20.
3. Otherwise, this is an instance of a non-trivial critical pair at it may be joined using our assumption that non-trivial critical pairs are joinable.

□

4.4. Terminating rewrite systems

Definition 4.22. Call a rewrite system R **terminating** when all rewrite sequences are finite. Call a term r a **normal form** (with respect to a rewrite system R) when $\forall s.\neg(r \xrightarrow{\mathsf{R}} s)$, that is, when r does not R-rewrite to anything.

Corollary 4.23. *Suppose* R *is terminating, uniform, and suppose non-trivial critical pairs in* R *are joinable. Then:*

1. R *is confluent.*
2. *If $r \to^* s$ and $r \to^* s'$ and s and s' are normal forms, then $s = s'$.*

4.5. Orthogonal rewrite systems

Definition 4.24. Call $R = (l \to m)$ **left-linear** when each unknown occurring in l occurs only once (Definition 4.3).

For example $\mathsf{f}(X) \to \mathsf{g}(X, X)$ is left-linear but $\mathsf{g}(X, X) \to \mathsf{f}(X)$ and $\mathsf{g}(\pi{\cdot}X, X) \to \mathsf{f}(X)$ are not. Note that $(a, a) \to a$ is left-linear.

Definition 4.25. Call R **orthogonal** when every $R \in \mathsf{R}$ is uniform and left-linear, and all critical pairs are trivial.

Definition 4.26. Suppose $R = (l \to m)$. Write $r \xrightarrow{R}_{\epsilon} s$ when $r \xrightarrow{R} s$ and the rewrite occurs at a position $C = (\pi{\cdot}X, X)$. We say that the rewrite with R occurs at **root position**.

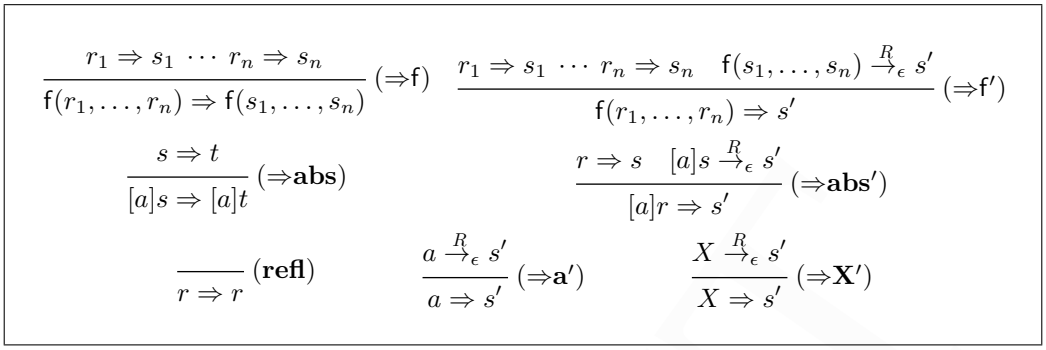


Figure 2: Parallel reduction relation

Expanding Definition 4.26, $r \overset{R}{\to}_\epsilon s$ when there exists θ and π such that $r = \pi{\cdot}(l\theta)$ and $s = \pi{\cdot}(m\theta)$. For example: if $R = (a \to a)$ then $a \overset{R}{\to}_\epsilon a$ but not $[a]a \overset{R}{\to}_\epsilon [a]a$.

We now set about proving Theorem 4.29, that $\to$ is confluent. Our proof-method, which is standard and goes back to [35], uses a *parallel reduction* relation $\Rightarrow$ (Definition 4.27). We prove $\Rightarrow$ confluent (Lemma 4.28) and then note that $\to^* = \Rightarrow^*$. The reason we use $\Rightarrow$ is because substitutions can turn 'one rewrite' into 'many parallel rewrites'; see the diagram in Lemma 4.28.

Definition 4.27. We define a **parallel reduction** relation $\Rightarrow$ by the rules in Figure 2.

Lemma 4.28. *If* R *is orthogonal then* $\Rightarrow$ *is strongly confluent (Definition 4.9), and therefore confluent.*

Proof. We prove by induction on the derivation of $r \Rightarrow s$ that for all s' if $r \Rightarrow s'$ then there exists some s'' such that $s \Rightarrow s''$ and $s' \Rightarrow s''$. We consider a selection of cases:

- *The derivations of* $r \Rightarrow s$ *and* $r \Rightarrow s'$ *both end in* $(\Rightarrow\mathsf{f})$. We use the inductive hypotheses and $(\Rightarrow\mathsf{f})$.
- *The derivation of* $r \Rightarrow s$ *ends in* $(\Rightarrow\mathsf{f})$ *and that of* $r \Rightarrow s'$ *ends in* $(\Rightarrow\mathsf{f}')$. So $r_i \Rightarrow s_i$ and $r_i \Rightarrow s'_i$ for $1 \le i \le n$, and $\mathsf{f}(s'_1,\ldots,s'_n) = \pi{\cdot}(l\theta) \overset{R}{\to}_\epsilon \pi{\cdot}(m\theta)$ for some π and $R = (l \to m) \in \mathsf{R}$. By inductive hypothesis there exist s''_i such that $s_i \Rightarrow s''_i$ and $s'_i \Rightarrow s''_i$. We now proceed as illustrated and explained below:

$$\begin{array}{lcl} \mathsf{f}(r_1,\ldots,r_n) \Longrightarrow \mathsf{f}(s'_1,\ldots,s'_n) & = & \pi{\cdot}(l\theta) \xrightarrow{R_\epsilon} \pi{\cdot}(m\theta) \\ \Downarrow \qquad\qquad \Downarrow & & \Downarrow \text{ Many parallel rewrites} \\ \mathsf{f}(s_1,\ldots,s_n) \Longrightarrow \mathsf{f}(s''_1,\ldots,s''_n) & = & \pi{\cdot}(l\theta') \xrightarrow{R_\epsilon} \pi{\cdot}(m\theta') \end{array}$$

  Either l is an unknown X or the rewrite $\mathsf{f}(s'_1,\ldots,s'_n) \Rightarrow \mathsf{f}(s''_1,\ldots,s''_n)$ takes place in the substitution θ.
  If l is an unknown then by uniformity we may rewrite $f(s''_1,\ldots,s''_n)$ using R and close the diagram by rewriting corresponding instances of $\theta(X)$ in $\pi{\cdot}(m\theta)$.
  Otherwise, by uniformity there is a substitution θ' such that $\theta(X) \Rightarrow \theta'(X)$ for every X and $\mathsf{f}(s''_1,\ldots,s''_n) = \pi{\cdot}(l\theta')$. Rules are also left-linear so R still applies to $\pi{\cdot}(l\theta)$: $\mathsf{f}(s''_1,\ldots,s''_n) \overset{R}{\to}_\epsilon \pi{\cdot}(m\theta')$ and therefore $\mathsf{f}(s_1,\ldots,s_n) \Rightarrow s\theta'$ by $(\Rightarrow\mathsf{f}')$ for R.

The other cases are no harder. □

Theorem 4.29. *If a theory* R *is orthogonal (Definition 4.25) then* R *is confluent (Definition 4.9).*

Proof. If the uniform rewrite system has only left-linear rules and only trivial critical pairs, then $\Rightarrow$ is confluent by Lemma 4.28. It is not hard to verify that $\rightarrow^* = \Rightarrow^*$, and the result follows. □

5. Closed terms

Equivariant unification—the problem of finding θ *and* π such that $\pi{\cdot}(r\theta) = s\theta$—is NP complete [5]. The same applies to corresponding matching problems. This matters to us because the rewrite relation in Definition 4.6 is equivariant; to determine whether r rewrites with a rule $(l \to r)$, we must solve an equivariant matching problem.

Fernández and the author introduced a notion of *closed term* such that for closed terms, equivariant matching/unification coincides with 'ordinary' matching/unification [12]. That is, for closed terms we can throw away the π.

We now develop corresponding definitions and results. The definitions and proofs in this paper are significantly different from previous work.[7]

5.1. Closed terms, and the equivariant extension

Definition 5.1. Define **explicit atoms** $ea(r)$ inductively by:

$$ea(a) = \{a\} \quad ea(X) = \varnothing \quad ea(\mathsf{f}(r_1, \ldots, r_n)) = \bigcup ea(r_i) \quad ea([a]r) = ea(r) \backslash \{a\}$$

Remark 5.2. The explicit atoms of r are the atoms that actually appear (unbound) in r. Contrast this with $fa(r)$ which is intuitively the atoms that *might* appear in r (perhaps after a substitution). For instance, $ea(X) = \varnothing \neq pmss(X) = fa(X)$.

Recall the notion of the *occurrences* in r from Subsection 2.8.

Notation 5.3. Write $\pi{\cdot}occ(r) = \{\pi{\cdot}[D]X \mid [D]X \in occ(r)\}$. Also if $D = [d_1, \ldots, d_n]$ and S is a permission set define $S \setminus D = S \setminus \{d_1, \ldots, d_n\}$.

Lemma 5.4. $ea(\pi{\cdot}r) = \pi{\cdot}ea(r)$ *and* $occ(\pi{\cdot}r) = \pi{\cdot}occ(r)$. *In addition,* $ea(r) \subseteq ea(r\theta)$.

Proof. By routine inductions on r. □

Lemma 5.5. $fa(r) = ea(r) \cup \bigcup\{fa([D]X) \mid [D]X \in occ(r)\}$.

Unpacking Definition 2.23, $fa(r) = ea(r) \cup \bigcup\{pmss(X) \setminus D \mid [D]X \in occ(r)\}$.

Definition 5.6. Call $occ(r)$ fa**-functional** when if $[D_1]X \in occ(r)$ and $[D_2]X \in occ(r)$ then $fa([D_1]X) = fa([D_2]X)$ (equivalently, when D_1 and D_2 contain the same atoms but not necessarily in the same order).

[7] The interested reader can begin by comparing our notion of closed terms in Definition 5.7, based on two simpler inductive definitions, with that used in [12, Definition 68], based on a renamed variant of a term and an equality derivable in an extended freshness context.

Definition 5.7. Call r **closed** when r is fa-functional and $ea(r) = \varnothing$.

Example 5.8.
- a is not closed (ea is non-empty).
- X is closed.
- $([a]X, X)$ is not closed (occ is not fa-functional).
- $[a](X, a)$ is closed.

Lemma 5.9. *Suppose $ea(r) = \varnothing$. Then $\pi{\cdot}(r\theta) = r\theta'$ if and only if $\pi{\cdot}(([D]X)\theta) = ([D]X)\theta'$ for every $[D]X \in occ(r)$.*

Proof. By a routine induction on r (actually a nominal induction, since we α-convert). We consider some cases:

- *The case X.* By definition $occ(X) = \{X\}$. The result is immediate.
- *The case a.* $ea(a) \neq \varnothing$ so there is nothing to prove.
- *The case $[a]r$.* α-renaming if necessary, suppose $\pi(a) = a$. By definition $occ([a]r) = \{[a, D]X \mid [D]X \in occ(r)\}$. Suppose $\pi{\cdot}(([a]r)\theta) = ([a]r)\theta'$, so that from Definitions 2.23 and 2.36 and by Lemma 2.25, $\pi{\cdot}(r\theta) = r\theta'$. By inductive hypothesis $\pi{\cdot}(([D]X)\theta) = ([D]X)\theta'$. It follows that $\pi{\cdot}(([a, D]X)\theta) = ([a, D]X)\theta'$. The reverse implication is similar. □

We need Definition 5.10 and Proposition 5.11 for Theorem 5.12.

Definition 5.10. Suppose we are given the following data:

- For each ϖ (Definition 2.15) a fixed but arbitrary choice of representative X_ϖ such that $orb(X) = \varpi$.
- For each X_ϖ a choice of term r_ϖ such that $fa(r_\varpi) \subseteq \mathrm{supp}(X_\varpi)$.

Define the **equivariant extension** F of this data by:

$$F(\pi{\cdot}X_\varpi) = \pi{\cdot}r_\varpi$$

Proposition 5.11.
1. *The equivariant extension from Definition 5.10 is well-defined and is a substitution.*
2. *Every substitution θ is an equivariant extension.*

Proof. For the first part, suppose $\pi{\cdot}X_\varpi = \pi'{\cdot}X_\varpi$. By construction that $\pi(a) = \pi'(a)$ for every $a \in \mathrm{supp}(X_\varpi)$. By assumption $fa(r_\varpi) \subseteq pmss(X_\varpi)$. The result follows by Lemma 2.27.

The second part is easy, noting that $\mathrm{supp}(\theta(X)) \subseteq pmss(X)$ by Lemma 2.33. □

Theorem 5.12. *r is closed if and only if*

$$\exists S.fa(r) \subseteq S \wedge \forall \pi, \theta.\pi{\cdot}fa(r\theta) \subseteq S \Rightarrow \exists \theta'.\pi{\cdot}(r\theta) = r\theta'.$$

Proof. Suppose there is a permission set $S \supseteq fa(r)$ such that if $\pi{\cdot}fa(r\theta) \subseteq S$ then there exists θ' such that $\pi{\cdot}(r\theta) = r\theta'$. There are two things to prove:

- *$ea(r)$ is empty.* Suppose there exists $a \in ea(r)$. Pick $b \in S \setminus ea(r)$. By assumption taking $\theta = id$ there exists θ' such that $(b\ a){\cdot}(r\theta) = r\theta'$. By Lemma 5.4 $ea((b\ a){\cdot}r) = (b\ a){\cdot}ea(r) \not\ni a$ and $a \in ea(r) \subseteq ea(r\theta')$, a contradiction.
- *$occ(r)$ is fa-functional.* Consider $[D_1]X$ and $[D_2]X$ in $occ(r)$; choose D_i such that $D_i \cap fa(r) = \varnothing$ for $i = 1, 2$. Suppose there exists $a \in fa([D_2]X) \setminus fa([D_1]X)$, and choose any $b \in fa([D_1]X)$ (since $pmss(X)$ is infinite and D_1 is finite, such a b exists).
  By Lemma 5.5 $a, b \in fa(r)$ so by assumption taking $\theta = id$ there exists θ' such that $(b\ a){\cdot}r = r\theta'$. By Lemma 5.9 $(b\ a){\cdot}[D_1]X = ([D_1]X)\theta$. By Lemma 2.26 a is free in the left-hand side, and by Lemma 2.38 a is not free in the right-hand side; a contradiction.

Suppose $occ(r)$ is fa-functional and $ea(r) = \varnothing$ and choose some permutation π and substitution θ.

If $occ(r) = \varnothing$ then by Lemma 5.5 $fa(r) = \varnothing$ so by Lemmas 2.27 and 2.40 $\pi{\cdot}(r\theta) = r$ and $r\theta' = r$, so there is nothing to prove.

Otherwise take $S = fa(r)$. For every element of $occ(r)$ make a fixed but arbitrary choice of representation as $[D]X$ where the atoms in D are disjoint from the atoms in $nontriv(\pi)$. We take θ' to equivariantly extend this choice (Definition 5.10), so we map $\pi'{\cdot}X$ to $(\pi' \circ \pi){\cdot}\theta(X)$ for the choice of representing X above, and otherwise to map Y to Y. Using Proposition 5.11 this is a substitution and $\pi{\cdot}(([D]X)\theta) = ([D]X)\theta'$ for every $[D]X \in occ(r)$. We use Lemma 5.9. □

5.2. Closed rewrite rules

Definition 5.13. Call a rewrite rule $l \to m$ **closed** when (l, m) is closed.

Recall that uniform rules satisfy Theorems 4.29 and 4.21.

Theorem 5.14. *If $R = (l \to m)$ is closed then it is uniform.*

Proof. By assumption $fv(m) \subseteq fv(l)$. Also (l, m) is fa-functional; it follows that $occ(m) \subseteq occ(l)$. The result follows from Lemma 5.5. □

Lemma 5.15. *If $fa(r)$ is infinite then $fa(r)$ is a permission set.*

Proof. By a routine induction on the definition of $fa(r)$ in Definition 2.23, using Corollary 2.52. □

Lemma 5.16. *For any terms r and l, if there exists a π such that $fa(r) \subseteq \pi{\cdot}fa(l)$ then one such π can be computed.*

Proof. If $fa(r)$ is finite then then problem is easy. If $fa(r)$ is infinite and $fa(l)$ is finite then no such π can exist. Suppose $fa(r)$ and $fa(l)$ are both infinite. By Lemma 5.15 and the construction of permission sets in Definition 2.10, they differ in finitely many atoms. The proof of Corollary 2.52 completes the algorithm.

(A direct proof using Lemma 2.8 is also possible.) □

Lemma 5.17. *Suppose r and l are terms and l is closed. Then*

1. $\exists\pi, \theta.r = \pi{\cdot}(l\theta)$ *implies*
2. $\forall\pi.fa(r) \subseteq \pi{\cdot}fa(l) \Rightarrow \exists\theta.r = \pi{\cdot}(l\theta)$.

Proof. Suppose $fa(r) \subseteq \pi{\cdot}fa(l)$ and $fa(r) \subseteq \pi'{\cdot}fa(l)$ and $r = \pi{\cdot}(l\theta)$.

We need a θ' such that $r = \pi'{\cdot}(l\theta')$. It follows from the above that $(\pi'^{-1} \circ \pi){\cdot}fa(l\theta) \subseteq fa(l)$. We use Theorem 5.12. □

Theorem 5.18. *If* R *is closed then* $\xrightarrow{\mathsf{R}}$ *can be calculated as follows, where for simplicity we suppose* $\mathsf{R} = \{(l \to m)\}$*:*

- *We try to match r against $\pi{\cdot}l$ for some π such that $fa(r) \subseteq \pi{\cdot}fa(l)$, if such a π exists (if such a π does exist then using Lemma 5.16, it can be computed).*
- *If we fail then, taking the contrapositive of Lemma 5.17, we must fail for instantiating for any $\pi{\cdot}l$. We descend into subterms of r and repeat the previous step.*

To use the matching algorithm of Section 3 to decide rewrites, it suffices that (l, m) satisfy condition 2 of Definition 3.1. So for example, this excludes a rewrite of the form $X \to \delta{\cdot}X$.

6. Conclusions

Many of the proofs above have appeared for the case of nominal terms, e.g. in nominal unification [38], rewriting [12], or permissive-nominal terms [11]. Yet, the approach to unknowns and α-equivalence taken here is different. Definitions and proofs simplify, shorten, and become better-behaved, and new properties emerge. Notably the use of δ simplifies the algorithms.

The infinite permutation δ opens the door to a new set of questions, since our unification algorithm fails if given a problem like $X \stackrel{?}{=} \delta{\cdot}X$ (this would generate an infinite freshness condition that $a\#X$ for every $a \in nontriv(\delta) \cap pmss(X)$). But this unification problem goes beyond what can be expressed in nominal unification from [38]. It should be possible to strengthen the algorithm: we would just require infinite (but still computable) freshness conditions. This has already been mentioned in Subsection 9.2 of [12] (closure conditions $\bullet t$).

We would argue that nominal terms as presented in [38] are not optimal: terms have a freshness context which introduces sequentiality and state into proofs and algorithms; terms cannot be quotiented by α-equivalence; it is unclear how to quantify over unknowns; α-equivalence has an exotic definition and cannot be mapped directly to atoms-abstraction in nominal abstract syntax.

For comparison the syntax in this paper has no freshness contexts; terms can be and are quotiented by binding; universal quantification can be added (see [8, 9, 24]); the definition of α-equivalence is ordinary; characteristic 'freshness' conditions on substitutions of nominal unknowns are revealed as corollaries of equivariance (Lemma 2.33); and nominal terms and nominal abstract syntax are made compatible and unified.

There are also mathematical reasons to like the syntax of this paper; it gives us some new theorems. For example, nominal rewriting from [12] and nominal algebra from [27] do not match up. A detailed discussion of this mismatch is in [13], where we go on

to make them match up by considering a notion of 'nominal rewriting in an arbitrarily extended freshness context'. In this paper the mismatch disappears (we have not used the syntax of this paper to build an algebraic logic; for that see [24]). The arbitrarily extended freshness context is already present, in this paper, as the complement of a permission set. The treatment of closed terms is simple and attractive. Syntax can be extended with binders for unknowns, as described in [8, 9, 23]. The nominal HSPA theorem of [21] simplifies to a nominal HSP theorem [24, Section 8].

The price we pay is the complexity in unknowns. In this paper, this involves adding to the syntax injections from natural numbers to atoms. Some readers may object to adding an infinity to syntax. But this is a paper that works from first principles, aimed at a readership of theoreticians who are expected to understand and care about such things. Consider that a presentation of 'ordinary' syntax from first principles (using Kuratowski pairs, strings, or initial objects) would look scary—and would include an infinity if there is a quotient by α-equivalence.

In a tutorial exposition we would sweep all this under the carpet and present things more simply, albeit in less detail. We might also wish to generalise the two-level structure of nominal terms, using the model of this paper as a guide: see *'generalising the treatment of dependencies'* below.

This paper builds on previous 'nominal' work as follows: permissive-nominal terms [10, 11] introduced the idea of permission sets for unknowns. An extended abstract proposed to model unknowns as lists [28], taking up ideas about names and well-orderability from an earlier paper [19]; this was followed by a more detailed and extensive journal paper [23]. Of course, this paper also uses [38] and [12] which introduced nominal unification and rewriting. This paper is a sequel to that work.

This paper is also a stepping-stone to further developments. Our suggestion that the approach in this paper has many advantages should not be read as a claim that it has all the answers. Notably in a survey chapter [24]—written after this paper—we generalise the treatment of unknowns to be a *strongly-supported nominal set*. The difference is that here, the development is concrete and committed to a specific representation of unknowns; whereas the development in [24] is more abstract, pays less attention to computability, and includes material from several other papers.

For future work we have in mind the following:

- It is natural to extend this paper to include nominal algebra [27] and first-order (permissive-nominal) logic [8, 9]. For more on this, see [24].
- We can extend this paper to include λ-abstraction over atoms and over unknowns. This would mirror the development of the Lambda-Context Calculus or Two-level lambda calculus [25, 29].
- We can extend the syntax so that unknowns are not lists of atoms but lists of atoms and finitely many terms; this will allow us to easily define a substitution action for atoms, as is considered (though using a very different presentation) in [16]—if we did this then the theory of unification would become more complex since unification up to a theory of substitution is known to be hard [7]; however, the logical aspects of the syntax should extend smoothly.
  Here, we would also be moving towards, amongst other things, the enquiry of Kit Fine into 'arbitrary objects' [15].
- We can generalise the treatment of dependencies to lose the two-level structure

evident in this paper (atoms depending on nothing, and unknowns depending on infinite lists of atoms) and have instead a general notion of *variable and variable-dependency*. This would subsume the ideas of nominal techniques into a more general framework.

In developing such future work we expect the concrete model of this paper to be useful as a prototype. It suggests how to generalise nominal permutations and abstractions to variables and variable-dependencies: the generalisation should be consistent with, but not necessarily committed to, mapping a variable to an ordered list of its dependencies. In other words, even if the reader does not care to model meta-variables as lists of 'lower-level' variables, we speculate that any sensible model of meta-variables should be *consistent* with that concrete model.

References

[1] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, Great Britain, 1998.

[3] Christophe Calvès and Maribel Fernández. A polynomial nominal unification algorithm. *Theoretical Computer Science*, 403:285–306, August 2008.

[4] Christophe Calvès and Maribel Fernández. The first-order nominal link. In *Proceedings of Logic-Based Program Synthesis and Transformation (LOPSTR'2010)*, volume 6564 of *Lecture Notes in Computer Science*, pages 234–248, 2011.

[5] James Cheney. The complexity of equivariant unification. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *Lecture Notes in Computer Science*, pages 332–344. Springer, 2004.

[6] James Cheney and Christian Urban. Alpha-prolog: A logic programming language with names, binding and alpha-equivalence. In Bart Demoen and Vladimir 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, 2004.

[7] Gilles Dowek. The undecidability of unification modulo sigma alone. Manuscript https://who.rocq.inria.fr/Gilles.Dowek/Publi/undecidability.pdf, accessed 18 April 2012.

[8] Gilles Dowek and Murdoch J. Gabbay. Permissive Nominal Logic. In *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP 2010)*, pages 165–176. ACM Press, 2010.

[9] Gilles Dowek and Murdoch J. Gabbay. Permissive Nominal Logic (journal version). *Transactions on Computational Logic*, 2012. In press.

[10] Gilles Dowek, Murdoch J. Gabbay, and Dominic P. Mulligan. Permissive Nominal Terms and their Unification. In *Proceedings of the 24th Italian Conference on Computational Logic (CILC'09)*, 2009.

[11] Gilles Dowek, Murdoch J. Gabbay, and Dominic P. Mulligan. Permissive Nominal Terms and their Unification: an infinite, co-infinite approach to nominal techniques (journal version). *Logic Journal of the IGPL*, 18(6):769–822, 2010.

[12] Maribel Fernández and Murdoch J. Gabbay. Nominal rewriting (journal version). *Information and Computation*, 205(6):917–965, June 2007.

[13] Maribel Fernández and Murdoch J. Gabbay. Closed nominal rewriting and efficiently computable nominal algebra equality. In *Electronic Proceedings in Theoretical Computer Science*, volume 34, pages 37–51, 2010.

[14] Maribel Fernández, Murdoch J. Gabbay, and Ian Mackie. Nominal Rewriting Systems. In *Proceedings of the 6th ACM SIGPLAN symposium on Principles and Practice of Declarative Programming (PPDP 2004)*, pages 108–119. ACM Press, August 2004.
[15] Kit Fine. *Reasoning with Arbitrary Objects*. Blackwell, 1985.
[16] Marcelo Fiore and Chung-Kil Hur. Second-order equational logic. In *Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL 2010)*, Lecture Notes in Computer Science. Springer, 2010.
[17] Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science (LICS 1999)*, pages 193–202. IEEE Computer Society Press, 1999.
[18] Murdoch J. Gabbay. *A Theory of Inductive Definitions with alpha-Equivalence*. PhD thesis, University of Cambridge, UK, March 2001.
[19] Murdoch J. Gabbay. FM-HOL, a higher-order theory of names. In F. Kamareddine, editor, *35 Years of Automath*, April 2002.
[20] Murdoch J. Gabbay. A General Mathematics of Names. *Information and Computation*, 205(7):982–1011, July 2007.
[21] Murdoch J. Gabbay. Nominal Algebra and the HSP Theorem. *Journal of Logic and Computation*, 19(2):341–367, April 2009.
[22] Murdoch J. Gabbay. Foundations of nominal techniques: logic and semantics of variables in abstract syntax. *Bulletin of Symbolic Logic*, 17(2):161–229, 2011.
[23] Murdoch J. Gabbay. Two-level nominal sets and semantic nominal terms: an extension of nominal set theory for handling meta-variables. *Mathematical Structures in Computer Science*, 21:997–1033, 2011.
[24] Murdoch J. Gabbay. Nominal terms and nominal logics: from foundations to meta-mathematics. In *Handbook of Philosophical Logic*, volume 17. Kluwer, 2012.
[25] Murdoch J. Gabbay and Stéphane Lengrand. The lambda-context calculus (extended version). *Information and computation*, 207:1369–1400, December 2009.
[26] Murdoch J. Gabbay and Aad Mathijssen. A Formal Calculus for Informal Equality with Binding. In *WoLLIC'07: 14th Workshop on Logic, Language, Information and Computation*, volume 4576 of *Lecture Notes in Computer Science*, pages 162–176. Springer, July 2007.
[27] Murdoch J. Gabbay and Aad Mathijssen. Nominal universal algebra: equational logic with names and binding. *Journal of Logic and Computation*, 19(6):1455–1508, December 2009.
[28] Murdoch J. Gabbay and Dominic P. Mulligan. Semantic nominal terms. In *Proceedings of the 2nd International Workshop on the Theory and Applications of Abstraction, Substitution and Naming (TAASN 2009)*, March 2009.
[29] Murdoch J. Gabbay and Dominic P. Mulligan. Two-level lambda-calculus. In *Proceedings of the 17th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2008)*, volume 246, pages 107–129, August 2009.
[30] Murdoch J. Gabbay and Andrew M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3–5):341–363, July 2001.
[31] Gerhard Gentzen. Untersuchungen über das logische Schließen [Investigations into logical deduction]. *Mathematische Zeitschrift 39*, pages 176–210,405–431, 1935. Translated in [37], pages 68–131.
[32] Jordi Levy and Mateu Villaret. An efficient nominal unification algorithm. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA 2010)*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 209–226. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.
[33] Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. *Transactions on Computational Logic*, 2012. to appear.
[34] Dag Prawitz. *Natural deduction: a prooof-theoretical study*. Almquist and Wiksell, 1965.

Reprinted by Dover, 2006.
[35] Barry K. Rosen. Tree-manipulating systems and church-rosser theorems. *Journal of the ACM (JACM)*, 20(1):160–187, 1973.
[36] Raymond Smullyan. *First-order logic*. Springer, 1968. Reprinted by Dover, 1995.
[37] M. E. Szabo, editor. *Collected Papers of Gerhard Gentzen*. North Holland, 1969.
[38] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal Unification. *Theoretical Computer Science*, 323(1–3):473–497, September 2004.

Acknowledgements

We are grateful to Jane Spurr and to two anonymous referees for their expertise and support. We also acknowledge the support of the Leverhulme trust, of grant DESAFIOS10 (TIN2009-14599-C03-03) from the Spanish Ministry of Science and Innovation, and of grant PROMETIDOS (P2009/TIC-1465) from the Madrid Regional Government.

DRAFT