The Lambda-Calculus is Nominal Algebraic

Murdoch J. Gabbay[1] and Aad Mathijssen[2,3]

1 Introduction

The λ-calculus is fundamental in the study of logic and computation. Partly this is because it is a tool to study functions and functions are an important object of study in this field. Partly this is because the λ-calculus seems to be, for *homo sapiens*, an ergonomic formal syntax.

DEFINITION 1. λ-terms g, h, k are inductively defined by

$$g \quad ::= \quad a \mid \lambda a.g \mid gg \mid \mathsf{c}.$$

In this paper we will write -$[a \mapsto$ -] as shorthand for $(\lambda a.$-$)$-. Thus $g[a \mapsto h]$ stands for $(\lambda a.g)h$ and *not* for the term resulting from 'substituting h for a in g' (we write that as $g[h/a]$, see Definition 44).

The λ-calculus represents functions in programming languages [25, 31], logic [7, 20], theorem-provers [3, 24], higher-order rewriting [5], and much more besides. However, the 'λ' in the λ-calculus has proved resistent to a treatment in universal algebra [8]. For example the property that

$$\text{"}(\lambda a.g)[b \mapsto h] = \lambda a.(g[b \mapsto h]) \text{ when } a \text{ does not occur free in } h\text{"}$$

cannot be represented in an algebraic framework, at least not obviously so, because of the *freshness condition* 'a does not occur free in h' which is necessary to avoid 'accidental capture' by λ. Similarly for the property "$\lambda a.(ga) = g$ when a does not occur free in g".

Nominal algebra is a form of universal algebra enriched with primitive constructs to handle names, binding, and freshness conditions — just like

[1]Homepage: http://www.gabbay.org.uk
[2]Email: a.h.j.mathijssen@tue.nl
[3]We are grateful to a dilligent anonymous referee of a previous paper for making a suggestion which put us on the path to write this paper, and to Pablo Nogueira and Chad Brown for useful advice on improving the exposition. Supported by grant RYC-2006-002131 at the Polytechnic University of Madrid.

$$
\begin{array}{rrrcl}
(\mathbf{var}{\mapsto}) & & a[a \mapsto X] & = & X \\
(\#{\mapsto}) & a\#Z \vdash & Z[a \mapsto X] & = & Z \\
(\mathbf{app}{\mapsto}) & & (Z'Z)[a \mapsto X] & = & (Z'[a \mapsto X])(Z[a \mapsto X]) \\
(\mathbf{abs}{\mapsto}) & b\#X \vdash & (\lambda b.Z)[a \mapsto X] & = & \lambda b.(Z[a \mapsto X]) \\
(\mathbf{id}{\mapsto}) & & Z[a \mapsto a] & = & Z
\end{array}
$$

Figure 1. Axioms of ULAM

those that appear in informal specifications of the λ-calculus and other languages with binders. Nominal algebra has the feature that, thanks to the enriched constructs, it allows fully formal algebraic reasoning which is pleasingly close to informal practice, including explicit reasoning on α-renaming and freshness side-conditions.

In this paper we introduce ULAM (Figure 1), a nominal algebra theory for the untyped λ-calculus. The axioms of ULAM make fundamental use of characteristic 'nominal' features of nominal algebra:

- We use *nominal unknowns* Z, Z', and X to represent unknown elements. Instantiation of nominal unknowns does not avoid capture; see Definition 11.
- *Freshness conditions* $a\#Z$, $b\#X$, and $b\#Z$ are a framework to prevent 'accidental capture' of names by binders.

We shall prove that ULAM is sound and complete with respect to a model constructed out of λ-terms quotiented by $\alpha\beta$-equivalence; the rest of the paper makes these observations formal.

Nominal techniques subscribe to a mathematical view according to which names are first-class entities in the denotation. This was used, for example, to develop the Gabbay-Pitts И quantifier and the Gabbay-Pitts model of α-abstraction [17]. A traditional view is that names arise as a syntax for talking about inputs to functions, and therefore they range over elements of the underlying domain.[1] The λ-calculus expresses this latter idea. With ULAM our nominal algebraic axiomatisation of the λ-calculus we make a novel connection between the two worlds; the axioms of ULAM express the properties that must be added to convert a nominal-style atom into a λ-calculus style variable, and a nominal-style abstraction into a λ-calculus

[1] In [17], names had no functional content at all; they were used just to build datatypes of abstract syntax trees with binding. Higher-order abstract syntax [26] is a way to do the same thing using the 'names as arguments to functions' philosophy.

binding.

Map of the paper. Section 2 introduces nominal algebra, giving basic definitions and results about syntax, freshness conditions, equality, and nominal algebra theories. Section 3 introduces the syntax and operational semantics of the untyped λ-calculus. Section 4 proves that ULAM is sound and complete (Subsections 4.1 and 4.2), and that it is conservative over the native nominal terms theory of α-equivalence (Subsection 4.3). The most technical material in this paper is concentrated in the proofs in Subsection 4.2. Finally, Section 5 discusses related and future work.

2 Nominal algebra

2.1 The syntax of nominal terms

We define a syntax of nominal terms. It is tailored to our application to the untyped λ-calculus; see elsewhere for general treatments [32, 13, 14, 23].

DEFINITION 2. Fix the following:

- A countably infinite set of **atoms** $\mathbb{A}$. We let $a, b, c, \ldots$ range over atoms. These model λ-calculus variables.

  We use a **permutative convention** that a and b range *permutatively* over atoms unless stated otherwise. For example in (#**ab**) from Figure 3, and in (**perm**) from Figure 4, a and b range over any two *distinct* atoms.

- A countably infinite collection of **unknowns**. We let $X, Y, Z, T, U, \ldots$ range over unknowns. These represent unknown elements in nominal algebra axioms.

  We also use a permutative convention that X and Y range permutatively over unknowns. In Figure 1 we make a fixed but arbitrary choice of unknown. That is, ULAM contains five axioms — not infinitely many for every possible a, b, X, Z, and Z' — but the choice is immaterial for all practical purposes as we shall see in (**ax**) of Figure 4 and in Definition 25.

- A possibly infinite collection of **constant symbols** $\mathsf{c} \in \mathsf{C}$.

Unknowns, atoms, and other distinct syntactic classes, are assumed disjoint.

REMARK 3. For the reader's convenience we provide Figure 2: a 'cheat-sheet' linking in a single list concepts from informal practice to some of the main definitions and lemmas which will soon follow. This list, by its nature, contains forward references.

- a is an atom. It represents an object-level variable symbol.
- X is an unknown. It represents a meta-variable.
- $\pi \cdot t$ is a permutative renaming of the atoms in t. We use this to provide a 'naturally capture-avoiding' theory of α-equivalence. $\pi \cdot X$ has the intuition of 'permute π in whatever X is instantiated to'.
- $t\sigma$ is t with meta-variables substituted, we can think of this as *instantiation*. If $\pi \cdot X$ appears in t then π acts on $\sigma(X)$ and the result is included in $t\sigma$.
- $a\#t$ asserts a freshness. It has the intuition 'a cannot be free in t'. $a\#X$ has the intuition of 'a is fresh for whatever X is instantiated to'.

Figure 2. Cheat-sheet for intuitive reading of notation

We can now set about building the machinery of nominal algebra.

DEFINITION 4. A **permutation** π of atoms is a bijection on atoms with **finite support** meaning that for some finite set of atoms $\pi(a) \neq a$, and for all other atoms $\pi(a) = a$. In words: For 'most' atoms π is the identity.

DEFINITION 5. Let **terms** t, u, v be inductively defined by:

$$t \quad ::= \quad a \;\mid\; \pi \cdot X \;\mid\; \lambda a.t \;\mid\; tt \;\mid\; \mathsf{c}.$$

We write **syntactic identity** of terms t, u as $t \equiv u$ to distinguish it from '=' the derivable equality-in-freshness-context we construct in Subsection 2.3. Note that if $\pi = \pi'$ then $\pi \cdot X \equiv \pi' \cdot X$, since permutations are represented by themselves. Also note that we do *not* quotient terms in any way.

We give some intuition of terms:

- An atom a represents a λ-calculus variable symbol.
- We call $\pi \cdot X$ a **moderated unknown**. This represents an unknown term, on which a permutation of atoms will be performed when X is instantiated. The use of permutations provides primitive support for *α-equivalence*.
- $t't$ represents the usual λ-calculus application.
- $\lambda a.t$ represents the usual λ-abstraction. Recall from the Introduction that we write $t[a \mapsto u]$ as shorthand for $(\lambda a.t)u$, for example in Figure 1.

A typed version of this syntax is possible; the interaction between atoms, unknowns, permutations, and λ-abstraction raises subtle and unexpected issues which have been investigated independently in the general framework of nominal rewriting [12]. Types would cause no essential difficulties for the results which follow.

2.2 Permutation, substitution and freshness

We need some more notation to talk about permutations (Definition 4).

DEFINITION 6. As usual we write id for the **identity** permutation, π^{-1} for the **inverse** of π, and $\pi \circ \pi'$ for the **composition** of π and π', i.e. $(\pi \circ \pi')(a) = \pi(\pi'(a))$. id is also the identity of composition: $id \circ \pi = \pi$ and $\pi \circ id = \pi$. Importantly, we shall write $(a\ b)$ for the permutation that **swaps** a and b, i.e. the permutation that maps a to b and vice versa, and maps all other c to themselves; note that this is the same permutation as $(b\ a)$. We may drop $\circ$ between swappings, writing for example $(a\ b) \circ (b\ c)$ as $(a\ b)(b\ c)$; this is a standard notation in the theory of permutations. We may write X as shorthand for $id \cdot X$.

DEFINITION 7. We write $a \in \pi$ when $\pi(a) \neq a$. We extend this inductively to $a \in t$ as follows:

$$\frac{}{a \in a} \qquad \frac{a \in \pi}{a \in \pi \cdot X} \qquad \frac{a \in t'}{a \in t't} \qquad \frac{a \in t}{a \in t't} \qquad \frac{}{a \in \lambda a.t} \qquad \frac{a \in t}{a \in \lambda b.t}$$

If $a \in t$ is not derivable write $a \notin t$. We read '$a \in$' as 'a occurs in'.

We also write $X \in t$ when X occurs anywhere in t, and $X \notin t$ otherwise. Occurrence is literal, for example $a \in \lambda a.a$ and $a \in (a\ b) \cdot X$.

DEFINITION 8. Define a **permutation action** $\pi \cdot t$ inductively by:

$$\pi \cdot a \equiv \pi(a) \qquad \pi \cdot (\pi' \cdot X) \equiv (\pi \circ \pi') \cdot X \qquad \pi \cdot \lambda a.t \equiv \lambda(\pi(a)).(\pi \cdot t)$$
$$\pi \cdot (t't) \equiv (\pi \cdot t')(\pi \cdot t) \qquad \pi \cdot \mathsf{c} \equiv \mathsf{c}.$$

Intuitively π propagates through the structure of t until it reaches an atom or a moderated unknown. Note that in the clause for λ, π acts also on the 'a'. Following Cheney (verbal communication) we say that the permutation action is *inherently capture-avoiding*, meaning that it can act uniformly and will not require special behaviour for binders. For example:

$$(a\ b) \cdot \lambda a.X \equiv \lambda b.(a\ b) \cdot X$$

LEMMA 9. $\pi \cdot (\pi' \cdot t) \equiv (\pi \circ \pi') \cdot t$ *and* $id \cdot t \equiv t$.

Proof. By an easy induction on the structure of t using the fact that composition of permutations is just composition of functions and so is associative. ■

DEFINITION 10. We call a **substitution** σ a function from unknowns to terms. We write $[t_1/X_1, \ldots, t_n/X_n]$ for the substitution mapping X_i to t_i for $1 \leq i \leq n$, and mapping Y to Y for all other Y.

DEFINITION 11. Define a **substitution action** $t\sigma$ inductively by:

$$a\sigma \equiv a \qquad (\pi \cdot X)\sigma \equiv \pi \cdot \sigma(X) \qquad (\lambda a.t)\sigma \equiv \lambda a.(t\sigma)$$
$$(t't)\sigma \equiv (t'\sigma)(t\sigma) \qquad \mathsf{c}\sigma \equiv \mathsf{c}.$$

We may call $t\sigma$ an **instance** of t.

For example:

$$(\lambda a.X)[a/X] \equiv \lambda a.(X[a/X]) \equiv \lambda a.a$$

$$\begin{aligned}(\lambda b.(a\ b) \cdot X)[a/X] &\equiv \lambda b.(((a\ b) \cdot X)[a/X]) \\ &\equiv \lambda b.((a\ b) \cdot (X[a/X])) \equiv \lambda b.(a\ b) \cdot a \equiv \lambda b.b\end{aligned}$$

$$X\sigma \equiv \sigma(X)$$

The final example is direct from the definition and Lemma 9; $\sigma(X)$ is 'the function σ applied to X', whereas $X\sigma$ is shorthand for $(id \cdot X)\sigma$.

Intuitively, σ propagates through the structure of t until it reaches an atom or a moderated unknown. σ 'evaporates' on an atom a, and on $\pi \cdot X$ it instantiates X to $\sigma(X)$ — then the permutation acts on $\sigma(X)$.

Substitution does not avoid capture; $(\lambda a.X)[a/X] \equiv \lambda a.a$. There is an deliberate analogy here with context substitution, which is the substitution used when we write 'let - be a in $\lambda a.$-', or 'suppose t is a in $\lambda a.t$'; we obtain $\lambda a.a$. Moderated unknowns behave exactly like the 'hole' - in syntactic contexts — except that an unknown can occur multiple times in a nominal term; unknowns occur in terms with a moderating permutation because this is needed, along with freshness contexts, to manage α-equivalence.

Recall from Definition 11 that $(\pi \cdot X)\sigma \equiv \pi \cdot \sigma(X)$. This extends easily to all terms:

LEMMA 12. $(\pi \cdot t)\sigma \equiv \pi \cdot (t\sigma)$.

Proof. By induction on t. The only interesting case if when $t \equiv \pi' \cdot X$. Then unpacking definitions

$$(\pi \cdot (\pi' \cdot X))\sigma \equiv (\pi \circ \pi') \cdot \sigma(X) \quad \text{and} \quad \pi \cdot ((\pi' \cdot X)\sigma) \equiv \pi \cdot (\pi' \cdot \sigma(X)).$$

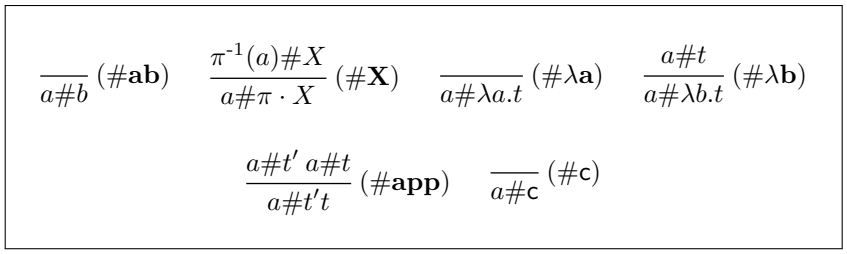

Figure 3. Freshness derivation rules for nominal terms

The result follows by Lemma 9. ■

DEFINITION 13. A **freshness (assertion)** is a pair $a\#t$ of an atom and a term. We call a freshness of the form $a\#X$ (so $t \equiv X$) **primitive**. We write Δ and ∇ for (finite, and possibly empty) sets of *primitive* freshnesses and call them **freshness contexts**.

We may drop set brackets in freshness contexts, e.g. writing $a\#X, b\#Y$ for $\{a\#X, b\#Y\}$. Also, we may write $a, b\#X$ for $a\#X, b\#X$. We write $a \in \Delta$ when $a\#X \in \Delta$ for some X, and $X \in \Delta$ when $a\#X \in \Delta$ for some a.

DEFINITION 14. We define **derivability on freshnesses** in natural deduction style [19] by the rules in Figure 3. In accordance with our permutative convention, a and b range over distinct atoms.

REMARK 15. A sequent style presentation of Figure 3 is also possible; for example $(\#\mathbf{X})$ becomes $\frac{\Delta \vdash \pi^{-1}(a)\#X}{\Delta \vdash a\#\pi\cdot X}$ and we need a new rule $\frac{[a\#X \in \Delta]}{\Delta \vdash a\#X}$ where the 'X' in the bottom line is shorthand for $id\cdot X$ and square brackets indicate a side-condition.

DEFINITION 16. We write $\Delta \vdash a\#t$ when a derivation of $a\#t$ exists using the rules of Figure 3, such that the assumptions are elements of Δ. In words, we say "$a\#t$ is **derivable** from Δ". We usually write $\emptyset \vdash a\#t$ as $\vdash a\#t$.

For example $\vdash a\#\lambda b.b$, $\vdash a\#\lambda a.a$, and $a\#X \vdash a\#X(\lambda a.Y)$:

$$\frac{\frac{}{a\#b}\,(\#\mathbf{ab})}{a\#\lambda b.b}\,(\#\lambda\mathbf{b}) \qquad \frac{}{a\#\lambda a.a}\,(\#\lambda\mathbf{a}) \qquad \frac{a\#X \quad \frac{}{a\#\lambda a.Y}\,(\#\lambda\mathbf{a})}{a\#X(\lambda a.Y)}\,(\#\mathbf{app})$$

The example of $\vdash a\#\lambda a.a$ demonstrates that $a\#t$, whose intuitive reading is 'is not free in', differs from $a \notin t$, whose intuitive reading is 'does not occur in'.

The derivation rules are completely syntax-directed. Therefore:

LEMMA 17.

1. $\Delta \vdash a\#b$ *always.*
2. *If* $\Delta \vdash a\#X$ *then* $a\#X \in \Delta$.
3. *If* $\Delta \vdash a\#\pi \cdot X$ *then* $\Delta \vdash \pi^{-1}(a)\#X$.
4. $\Delta \vdash a\#\lambda a.t$ *always.*
5. *If* $\Delta \vdash a\#\lambda b.t$ *then* $\Delta \vdash a\#t$.
6. *If* $\Delta \vdash a\#t't$ *then* $\Delta \vdash a\#t'$ *and* $\Delta \vdash a\#t$.
7. $\Delta \vdash a\#\mathsf{c}$ *always.*

Proof. By an easy induction on the structure of the derivation rules in Figure 3. ■

We can *strengthen* the freshness context Δ in freshness derivations:

THEOREM 18. *If* $c\#Z, \Delta \vdash a\#t$ *and* $c \notin t$ *then* $\Delta \vdash a\#t$.

Proof. We transform a derivation of $c\#Z, \Delta \vdash a\#t$ into a derivation of $\Delta \vdash a\#t$:

- If $c\#Z, \Delta \vdash a\#X$ by assumption then $a\#X \in c\#Z, \Delta$. Since $a \neq c$, we know $a\#X \in \Delta$, and we conclude $\Delta \vdash a\#X$ by assumption.

  The case of $a\#Z$ is similar.
- ($\#\mathbf{X}$): Suppose $c\#Z, \Delta \vdash a\#\pi \cdot X$ is derived using ($\#\mathbf{X}$). Then $c\#Z, \Delta \vdash \pi^{-1}(a)\#X$. By assumption $c \notin \pi \cdot X$, so $\pi(c) = c$. Since also $a \neq c$, we know $\pi^{-1}(a) \neq c$. By the inductive hypothesis and the simple fact that $c \notin X$ we obtain $\Delta \vdash \pi^{-1}(a)\#X$. We conclude $\Delta \vdash a\#\pi \cdot X$ using ($\#\mathbf{X}$).

  The case of $a\#\pi \cdot Z$ is similar.
- ($\#\mathbf{ab}$), ($\#\lambda\mathbf{a}$) and ($\#\mathsf{c}$) carry over directly.
- ($\#\lambda\mathbf{b}$) and ($\#\mathbf{app}$) are straightforward using the inductive hypothesis and the following facts: if $a \notin \lambda b.t$ then $a \notin t$, and if $a \notin t't$ then $a \notin t'$ and $a \notin t$.

■

Freshness context *weakening* also holds and is part of a more general result; see Theorem 21.

Equivariance is a characteristic property of nominal techniques. Equivariance arises from the use nominal techniques make of permutations as

opposed to renamings (possibly non-bijective functions on atoms). Intuitively, equivariance states that if something is true, then it should remain true if we permute atoms. Formally we write:

LEMMA 19. *If* $\Delta \vdash a\#t$ *then* $\Delta \vdash \pi(a)\#\pi \cdot t$.

Proof. By induction on the structure of derivations of $a\#t$ from Δ. The only non-trivial case is $(\#\mathbf{X})$. Suppose $a\#\pi' \cdot X$ is derived from $\pi'^{-1}(a)\#X$ using $(\#\mathbf{X})$. We must show that $\pi(a)\#(\pi \circ \pi') \cdot X$. This follows from $(\pi \circ \pi')^{-1}(\pi(a))\#X$. It is a fact that $(\pi \circ \pi')^{-1}(\pi(a)) = \pi'^{-1}(a)$. The result follows. ■

Lemma 19 is part of a collection of equivariance properties and these are responsible for much of the technical convenience of the nominal treatment of names. See for example [32, Lemma 2.7], [13, Lemma 20], [16, Appendix A], and [17, Lemma 4.7].

We can extend the substitution action to freshness contexts:

DEFINITION 20. We write

$$\Delta\sigma \quad \text{for} \quad \{a\#\sigma(X) \mid a\#X \in \Delta\}.$$

Intuitively read this as 'apply σ to every X in Δ'.

We write $\Delta' \vdash \Delta\sigma$ when $\Delta' \vdash a\#\sigma(X)$ for every $a\#X \in \Delta$.

Note that $\Delta\sigma$ need not be a freshness context, because it might contain $a\#t$ for t not an unknown.

THEOREM 21. *For any* Δ', Δ, σ, *if* $\Delta \vdash a\#t$ *and* $\Delta' \vdash \Delta\sigma$ *then* $\Delta' \vdash a\#t\sigma$.
As a corollary, if $\Delta \vdash a\#t$ *and* $\Delta \subseteq \Delta'$ *then* $\Delta' \vdash a\#t$.

Proof. The structure of natural deduction derivations is such that the conclusion of one derivation may be 'plugged in' to an assumption in another derivation, if assumption and conclusion are syntactically identical. The structure of all the rules except for $(\#\mathbf{X})$ is such that if unknowns are instantiated by σ nothing need change.

For the case of $(\#\mathbf{X})$ we use Lemma 19.

The corollary follows taking $\sigma(X) \equiv id \cdot X$ for all X. ■

2.3 Equality, axioms and theories

DEFINITION 22. We call a pair $t = u$ an **equality (assertion)**. We call the pair $\nabla \vdash t = u$ of a freshness context ∇ and an equality $t = u$ an **axiom**. We may write $\emptyset \vdash t = u$ as $\vdash t = u$.

We call a set of axioms T a **theory**. The theories needed in this paper are:

- CORE: the empty set of axioms.
- ULAM: the axioms from Figure 1.

REMARK 23. Theory ULAM axiomatises a non-extensional λ-calculus. An extensional version is obtained if we add the following axiom to Figure 1:

$$(\eta) \qquad a\#Z \vdash \lambda a.(Za) = Z$$

We see no difficulties with extending the results of this paper to the extensional case.

REMARK 24. A word on the history of the ideas behind ULAM: ULAM grew out of SUB [15], which was based on a nominal rewrite system for the λ-calculus [13, Example 43], which was itself based on an example signature used in nominal unification [32, Example 2.2]. The axioms of ULAM directly and deliberately identify nominal abstraction $[a]t$ (notation in the style of [32, 15]) with λ-abstraction $\lambda a.t$, and nominal substitution $\mathsf{sub}(t,u)$ with λ-calculus application tu. In [32, 13, 15] term-formers such as λ-abstraction and application exist separately and a sort-system ensures for example that the t in $\mathsf{sub}(t,u)$ is an abstraction:

- λ is a unary term-former. It takes a nominal abstraction as an argument and returns a term of base sort. In the style of [15, 32] we write that it has arity $([\mathbb{A}]\mathbb{T})\mathbb{T}$.
- Substitution is a binary term-former sub with arity $([\mathbb{A}]\mathbb{T},\mathbb{T})\mathbb{T}$.

The direct identification of nominal abstraction with λ-abstraction, and substitution with application, is possible because terms of ULAM are unsorted (i.e. all terms have base sort, including abstractions) and because nominal algebra allows us to assert the relevant equalities in a logical framework.

DEFINITION 25. Define **derivability on equalities** in natural deduction style by the rules in Figure 4. We write $\Delta \vdash_{\mathsf{T}} t = u$ when a derivation of $t = u$ exists using these rules such that:

- for each instance of $(\mathbf{ax}_{\nabla\vdash\mathbf{t}=\mathbf{u}})$, $\nabla \vdash t = u$ is an axiom from T.
- in the derivations of freshnesses (introduced by instances of $(\mathbf{ax}_{\nabla\vdash\mathbf{t}=\mathbf{u}})$ and (**perm**)) the freshness assumptions used are from Δ only.

We write $\emptyset \vdash_{\mathsf{T}} t = u$ as $\vdash_{\mathsf{T}} t = u$.

We briefly discuss the most interesting rules of Figure 4:

- (**perm**). This rule provides us with a concise way to express α-equivalence. See Lemma 28 and Theorem 49 for formal expressions of this intuition.

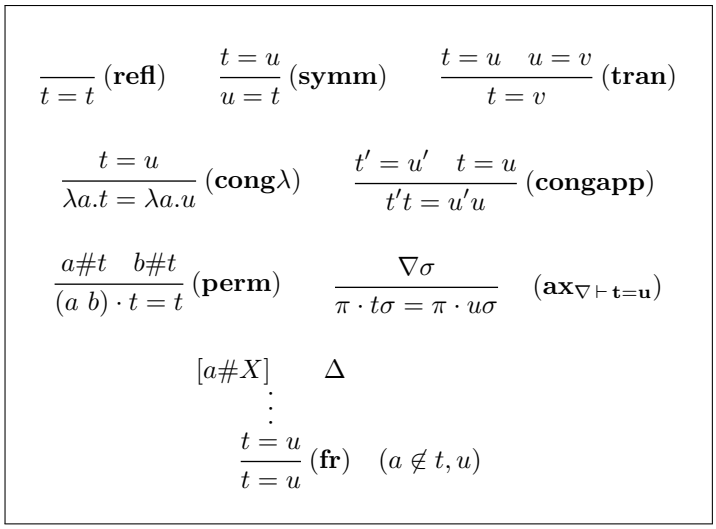

Figure 4. Derivation rules for nominal equality

- $(\mathbf{ax}_{\nabla\vdash\mathbf{t=u}})$. This rule formally expresses how we obtain instances of axioms: instantiate unknowns by terms (using substitutions) and rename atoms (using permutations) such that the hypotheses are corresponding instances of freshness conditions of the axiom. In this rule σ ranges over substitutions, π ranges over permutations, and $\nabla\sigma$ stands for the hypotheses $\{a\#\sigma(X) \mid a\#X \in \nabla\}$.

  The reader might expect the premise of the axiom rule to be $\pi\cdot\nabla\sigma$ instead of $\nabla\sigma$. It turns out that both versions are correct, because of Lemma 19: $\Delta \vdash \nabla\sigma$ if and only if $\Delta \vdash \pi\cdot\nabla\sigma$ for any Δ.

- (**fr**). This rule allows us to introduce a fresh atom into the derivation. In (**fr**) the square brackets denote *discharge* in the sense of natural deduction, for example as in implication introduction [19]; Δ denotes the other assumptions of the derivation of $t = u$. In sequent style (**fr**) would be $\frac{a\#X, \Delta \vdash t = u}{\Delta \vdash t = u}\ (a \notin t, u)$.

  We will *always* be able to find a fresh atom, no matter how unknowns are instantiated, since all our syntax is finite and can never mention more than finitely many atoms.

To provide some intuition for these rules, we give a number of examples.

EXAMPLE 26. The (**perm**) rule allows us to show standard α-equivalence

properties such as $\vdash_{\mathsf{CORE}} \lambda a.a = \lambda b.b$ and $\vdash_{\mathsf{CORE}} \lambda a.\lambda b.(ab) = \lambda b.\lambda a.(ba)$:

$$\dfrac{\dfrac{\dfrac{}{a\#b}\,(\#\mathbf{ab})}{a\#\lambda b.b}\,(\#\lambda\mathbf{b}) \qquad \dfrac{}{b\#\lambda b.b}\,(\#\lambda\mathbf{a})}{\lambda a.a = \lambda b.b}\,(\mathbf{perm}) \qquad \dfrac{\dfrac{\dfrac{}{a\#\lambda a.(ba)}\,(\#\lambda\mathbf{a})}{a\#\lambda b.\lambda a.(ba)}\,(\#\lambda\mathbf{b}) \qquad \dfrac{}{b\#\lambda b.\lambda a.(ba)}\,(\#\lambda\mathbf{a})}{\lambda a.\lambda b.(ab) = \lambda b.\lambda a.(ba)}\,(\mathbf{perm})$$

Note that we use $\lambda a.a \equiv (a\ b) \cdot \lambda b.b$ and $\lambda a.\lambda b.(ab) \equiv (a\ b) \cdot \lambda b.\lambda a(ba)$ in the conclusions of the derivations.

EXAMPLE 27. We give a full derivation of $\vdash_{\mathsf{ULAM}} (\lambda b.(\lambda a.b))a = \lambda c.a$. We use the shorthand from the Introduction to write for example $(\lambda b.(\lambda a.b))a$ as $(\lambda a.b)[b \mapsto a]$.

$$\Pi \;=\; \dfrac{\dfrac{\dfrac{\dfrac{}{a\#b}\,(\#\mathbf{ab})}{a\#\lambda c.b}\,(\#\lambda\mathbf{b}) \qquad \dfrac{}{c\#\lambda c.b}\,(\#\lambda\mathbf{a})}{\lambda a.b = \lambda c.b}\,(\mathbf{perm})}{\lambda b.\lambda a.b = \lambda b.\lambda c.b}\,(\mathbf{cong}\lambda) \qquad \dfrac{}{a = a}\,(\mathbf{refl})}{(\lambda a.b)[b \mapsto a] = (\lambda c.b)[b \mapsto a]}\,(\mathbf{congapp})$$

$$\Pi' \;=\; \dfrac{\dfrac{}{c\#a}\,(\#\mathbf{ab})}{(\lambda c.b)[b \mapsto a] = \lambda c.(b[b \mapsto a])}\,(\mathbf{ax}_{\mathbf{abs}\mapsto}) \qquad \dfrac{\dfrac{}{b[b \mapsto a] = a}\,(\mathbf{ax}_{\mathbf{var}\mapsto})}{\lambda c.(b[b \mapsto a]) = \lambda c.a}\,(\mathbf{cong}\lambda)}{(\lambda c.b)[b \mapsto a] = \lambda c.a}\,(\mathbf{tran})$$

$$\dfrac{\Pi \qquad \Pi'}{(\lambda a.b)[b \mapsto a] = \lambda c.a}\,(\mathbf{tran})$$

The examples above do not showcase the full power of nominal terms, because we are not reasoning on terms containing *unknowns*. The raison d'étre of nominal terms is their unknowns and how their interaction with permutations and freshness allow us to manage α-conversion. We now consider examples of derivations with α-renaming and freshness in the presence of unknowns:

LEMMA 28. $b\#X \vdash_{\mathsf{CORE}} X[a \mapsto T] = ((b\ a) \cdot X)[b \mapsto T]$

Proof. De-sugaring we must derive $(\lambda a.X)T = (\lambda b.(b\ a) \cdot X)T$ from $b\#X$:

$$\dfrac{\dfrac{\dfrac{\dfrac{b\#X}{a\#(b\ a)\cdot X}(\#\mathbf{X})}{a\#\lambda b.(b\ a)\cdot X}(\#\lambda\mathbf{b}) \qquad \dfrac{}{b\#\lambda b.(b\ a)\cdot X}(\#\lambda\mathbf{a})}{\lambda a.X = \lambda b.(b\ a)\cdot X}(\mathbf{perm}) \qquad \dfrac{}{T=T}(\mathbf{refl})}{(\lambda a.X)T = (\lambda b.(b\ a)\cdot X)T}(\mathbf{congapp})$$

The instance of (**perm**) relies on the fact that $(b\ a)\cdot\lambda a.X \equiv \lambda b.(b\ a)\cdot X$. ■

LEMMA 29 (Substitution Lemma).

$$a\#Y \vdash_{\mathsf{ULAM}} Z[a\mapsto X][b\mapsto Y] = Z[b\mapsto Y][a\mapsto X[b\mapsto Y]].$$

The usual proof of the substitution lemma is by induction on Z but now Z is just a formal symbol and part of the syntax — but instead, the axioms of ULAM capture this behaviour.

Proof. By (**tran**) it suffices to derive

$$((\lambda a.Z)X)[b\mapsto Y] = ((\lambda a.Z)[b\mapsto Y])(X[b\mapsto Y])$$

and

$$((\lambda a.Z)[b\mapsto Y])(X[b\mapsto Y]) = (\lambda a.(Z[b\mapsto Y]))(X[b\mapsto Y])$$

from $a\#Y$. The first part follows by axiom (**app**$\mapsto$). The second part can be derived as follows:

$$\dfrac{\dfrac{a\#Y}{(\lambda a.Z)[b\mapsto Y] = \lambda a.(Z[b\mapsto Y])}(\mathbf{ax_{abs\mapsto}}) \qquad \dfrac{}{X[b\mapsto Y] = X[b\mapsto Y]}(\mathbf{refl})}{((\lambda a.Z)[b\mapsto Y])(X[b\mapsto Y]) = (\lambda a.(Z[b\mapsto Y]))(X[b\mapsto Y])}(\mathbf{congapp})$$

■

The following lemma formally connects the usual use of substitution to handle α-renaming, with the unusual treatment of α-renaming which is primitive to nominal terms, based on permutations.

LEMMA 30. $b\#Z \vdash_{\mathsf{ULAM}} Z[a\mapsto b] = (b\ a)\cdot Z$.

Proof. We sketch the derivation:

$$\dfrac{\dfrac{\begin{array}{c} b\#Z \\ \vdots \\ \text{As for Lemma 28} \end{array}}{Z[a\mapsto b] = ((b\ a)\cdot Z)[b\mapsto b]} \qquad \dfrac{}{((b\ a)\cdot Z)[b\mapsto b] = (b\ a)\cdot Z}(\mathbf{ax_{id\mapsto}})}{Z[a\mapsto b] = (b\ a)\cdot Z}(\mathbf{tran})$$

■

REMARK 31. Axiom (**id**$\mapsto$) is *equivalent* to Lemma 30. That is, we can derive $Z[a \mapsto a] = Z$ using Lemma 30:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}[b\#Z]^1\\ \vdots\\ \text{As for Lemma 28}\end{array}}{Z[a \mapsto a] = ((b\ a) \cdot Z)[b \mapsto a]} \qquad \dfrac{\dfrac{[b\#Z]^1}{a\#(b\ a) \cdot Z}\,(\#\mathbf{X})}{((b\ a) \cdot Z)[b \mapsto a] = Z}\,(\text{Lemma 30})}{Z[a \mapsto a] = Z}\,(\mathbf{tran})}{Z[a \mapsto a] = Z}\,(\mathbf{fr})^1$$

In this derivation the superscript number one ^{1} is an annotation associating the instance of the rule (**fr**) with the assumption it discharges in the derivation. Note how we use (**fr**) to *generate* a 'fresh atom'.

LEMMA 32. *If $a\#X$ for every unknown in t, and $a \notin t$, then $\vdash a\#t$.*

Proof. By an easy induction on t using the rules in Figure 3. ■

DEFINITION 33. We write $\mathrm{ds}(\pi, \pi')$ for the set $\{a \mid \pi(a) \neq \pi'(a)\}$, the **difference set** of permutations π and π'. We write $\Delta \vdash \mathrm{ds}(\pi, \pi')\#X$ for a set of proof-obligations $\Delta \vdash a\#X$, one for each $a \in \mathrm{ds}(\pi, \pi')$.

REMARK 34. Figure 4 contains some redundancy; (**refl**) may be emulated using (**fr**) and (**tran**) as follows:

$$\dfrac{\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots\\ a\#(a\ b) \cdot t & b\#(a\ b) \cdot t\end{array}}{t = (a\ b) \cdot t}\,(\mathbf{perm}) \qquad \dfrac{\begin{array}{cc}\vdots & \vdots\\ a\#t & b\#t\end{array}}{(a\ b) \cdot t = t}\,(\mathbf{perm})}{t = t}\,(\mathbf{tran})}{t = t}\,(\mathbf{fr}) \text{ for all unknowns in } t$$

Here a, b are chosen fresh (so $a \notin t$ and $b \notin t$). The vertical dots elide derivations described in Lemma 32.

It is convenient in a logic of equality to be able to derive that $t = t$ without 'going round the houses' with (**tran**) and (**fr**), so we include both (**perm**) and (**refl**).

(**perm**) and (**refl**) are both instances of the following rule:

$$\dfrac{\mathrm{ds}(\pi, \pi')\#t}{\pi \cdot t = \pi' \cdot t}\,(\mathrm{ds}\mathbf{refl})$$

Here $\mathrm{ds}(\pi, \pi')\#t$ is shorthand for the set of $a\#t$ for all $a \in \mathrm{ds}(\pi, \pi')$, if any. This rule looks complicated, so we do not use it in Figure 4.

We can derive syntactic criteria for determining equality in CORE. These will be useful later:

THEOREM 35. *$\Delta \vdash_{\mathsf{CORE}} t = u$ precisely when one of the following hold:*

1. *$t \equiv a$ and $u \equiv a$.*
2. *$t \equiv \pi \cdot X$ and $u \equiv \pi' \cdot X$ and $\Delta \vdash \mathrm{ds}(\pi, \pi')\#X$.*
3. *$t \equiv \lambda a.t'$ and $u \equiv \lambda a.u'$ and $\Delta \vdash_{\mathsf{CORE}} t' = u'$.*
4. *$t \equiv \lambda a.t'$ and $u \equiv \lambda b.u'$ and $\Delta \vdash b\#t'$ and $\Delta \vdash_{\mathsf{CORE}} (b\ a) \cdot t' = u'$.*
5. *$t \equiv t't$ and $u \equiv u'u$ and $\Delta \vdash_{\mathsf{CORE}} t' = u'$ and $\Delta \vdash_{\mathsf{CORE}} t = u$.*
6. *$t \equiv \mathsf{c}$ and $u \equiv \mathsf{c}$.*

Proof. See [15, Corollary 2.32] or [23, Corollary 2.5.4]. ■

COROLLARY 36 (Consistency). *For all Δ there are t and u such that $\Delta \not\vdash_{\mathsf{CORE}} t = u$.*

Proof. By Theorem 35, $\Delta \vdash_{\mathsf{CORE}} a = b$ is never derivable. ■

COROLLARY 37 (Decidability). *It is decidable whether $\Delta \vdash_{\mathsf{CORE}} t = u$ is true or not.*

Proof. By an easy induction on t, using the syntactic criteria of Theorem 35. ■

A number of properties on freshnesses also hold for equalities of any theory T. For instance we can strengthen the freshnesses context Δ in equational derivations:

LEMMA 38. *If $c\#Z, \Delta \vdash_{\mathsf{T}} t = u$ and $c \notin t$ and $c \notin u$, then $\Delta \vdash_{\mathsf{T}} t = u$.*

Proof. We extend the derivation with (**fr**). ■

Also we may permute atoms and instantiate unknowns in equational derivations:

LEMMA 39. *If $\Delta \vdash_{\mathsf{T}} t = u$ then $\Delta \vdash_{\mathsf{T}} \pi \cdot t = \pi \cdot u$.*

Proof. By induction on the structure of the rules of Figure 4. ■

THEOREM 40. *For any T, Δ', Δ, σ, if $\Delta \vdash_{\mathsf{T}} t = u$ and $\Delta' \vdash \Delta\sigma$ then $\Delta' \vdash_{\mathsf{T}} t\sigma = u\sigma$.*

As a corollary, if $\Delta \vdash_{\mathsf{T}} t = u$ and $\Delta \subseteq \Delta'$ then $\Delta' \vdash_{\mathsf{T}} t = u$.

Proof. Analogous to the proof of Theorem 21.

In the case that σ mentions a 'fresh' atom used in an instance of (**fr**), we rename that atom to be 'fresher'. The inductive hypothesis is valid also for the 'freshened' derivation because of the mathematical principle of ZFA equivariance ([16, Appendix A] or [17]); induction on a measure of the depth of derivations is also possible, subject to uninteresting lemmas that renaming atoms does not change depth. ■

3 Untyped λ-terms

DEFINITION 41. We call a term **ground** when it mentions no unknowns. Ground terms are inductively characterised by the grammar in Definition 1.

It is no coincidence that ground terms are characterised by Definition 1; as discussed in Subsection 2.1 the syntax of nominal terms used in this paper is specialised to the intended application.

The rest of this section sketches a formal development of the syntax and operational semantics of λ-terms and $\alpha\beta$-reduction, and links it to the 'nominal' exposition. For more detailed treatments of λ-terms and $\alpha\beta$-reduction see elsewhere [2, 6].

DEFINITION 42. Define the **free atoms** $fa(g)$ inductively by:

$$fa(a) = \{a\} \quad fa(\lambda a.g) = fa(g) \setminus \{a\} \quad fa(gg') = fa(g) \cup fa(g') \quad fa(\mathsf{c}) = \{\}.$$

This is standard.

LEMMA 43. *$a \notin fa(g)$ if and only if $\vdash a\#g$.*
Also, if $a \notin g$ (Definition 7) then $\vdash a\#g$.

Proof. By routine inductions on the structure of g. ■

DEFINITION 44. Define the **size** of a ground term inductively by:

$$|a| = 1 \qquad |\lambda a.g| = |g| + 1 \qquad |g'g| = |g'| + |g| + 1 \qquad |\mathsf{c}| = 1.$$

We define a **capture-avoiding substitution** action $g[h/a]$ inductively on the size of g by:

$$\begin{array}{l} a[h/a] \equiv h \qquad b[h/a] \equiv b \qquad (\lambda a.g)[h/a] \equiv \lambda a.g \\ (\lambda b.g)[h/a] \equiv \lambda b.(g[h/a]) \quad (b \notin fa(h)) \\ (\lambda b.g)[h/a] \equiv \lambda c.(g[c/b][h/a]) \quad (b \in fa(h),\ c \text{ fresh}) \\ (g'g)[h/a] \equiv (g'[h/a])(g[h/a]) \qquad \mathsf{c}[h/a] = \mathsf{c}. \end{array}$$

In the clause for $(\lambda b.g)[h/a]$ we make some fixed and arbitrary choice of fresh c (the 'c fresh'), for each b, g, h, a.

A basic property is useful in the proofs of the results which follow:

LEMMA 45. *Capture-avoiding substitution of atoms for atoms preserves size. More formally,* $|g[a/a]| = |g|$ *and* $|g[b/a]| = |g|$.

Proof. By an easy induction on $|g|$. ■

LEMMA 46. $fa(g[h/a]) \subseteq (fa(g) \backslash \{a\}) \cup fa(h)$.
Also, if $\vdash a\#h$ *then* $\vdash a\#g[h/a]$.

Proof. The first part is by a routine induction on size. For the second part we prove the contrapositive. By Lemma 43 it suffices to show that $a \in fa(g[h/a])$ implies $a \in fa(h)$. This also follows by a routine induction on the size of g. ■

We introduce the usual notion of α-equivalence.

DEFINITION 47. We write $=_\alpha$ for the **α-equivalence** relation, which is obtained by extending syntactic equivalence with the following rule to rename bound variables:

$$\lambda a.g =_\alpha \lambda b.h \quad \text{when} \quad g[c/a] =_\alpha h[c/b] \quad \text{for some fresh atom } c.$$

The following lemma shows how permutations interact with $=_\alpha$:

LEMMA 48.

1. *If* $a, b \notin fa(g)$ *then* $(a\ b) \cdot g =_\alpha g$.
2. *If* $b \notin fa(g)$ *then* $g[b/a] =_\alpha (b\ a) \cdot g$.

Proof. For the first part, we observe that all a and b in g must occur in the scope of λa and λb. We traverse the structure of g bottom-up and rename these to fresh atoms (for example $\lambda a'$ and $\lambda b'$ which do not occur anywhere in g). Call the resulting term g'. Now $(a\ b) \cdot g' \equiv g'$ because $a, b \notin g'$. Equality is symmetric, so we reverse the process to return to g.

The second part follows by an induction on $|g|$. ■

THEOREM 49. *Derivable equality in* CORE *coincides with* $=_\alpha$ *on ground terms.*

Proof. See [15, Theorem 3.9] or [23, Theorem 4.3.13]. ■

REMARK 50. We do not quotient terms by α-conversion and we do not use a syntax based on a nominal-style datatype of syntax-with-binding [17]. This is because the proof of Theorem 57 involves keeping careful track of what atoms do and do not appear in terms, and if we quotient by α-equivalence now we lose information — names of abstracted atoms — which is useful for expressing that proof.

DEFINITION 51. Let **β-reduction** $g \to_\beta h$ be inductively defined by:

$$\frac{}{g[a \mapsto h] \to_\beta g[h/a]} \qquad \frac{g \to_\beta g'}{\lambda a.g \to_\beta \lambda a.g'} \qquad \frac{g \to_\beta g' \quad h \to_\beta h'}{gh \to_\beta g'h'}$$

We call g a **β-normal form** when there is no g' such that $g \to_\beta g'$.

We write $g \to_{\alpha\beta} h$ when there exist g' and h' such that

$$g =_\alpha g', \qquad g' \to_\beta h', \quad \text{and} \quad h' =_\alpha h.$$

We write $=_{\alpha\beta}$ for the transitive reflexive symmetric closure of $\to_{\alpha\beta}$.

THEOREM 52. *$\to_{\alpha\beta}$ is confluent.*

Proof. See elsewhere [6]. ■

4 Soundness, completeness, conservativity

4.1 Soundness

DEFINITION 53. We call a substitution ς **ground** for a set of unknowns $\mathcal{X}$ when $\varsigma(X)$ is ground for every $X \in \mathcal{X}$. We call ς ground for Δ, t, u when ς is ground for the set of unknowns appearing anywhere in Δ, t, or u.

LEMMA 54. *$fa(\pi \cdot g) = \{\pi(a) \mid a \in fa(g)\}$.*

Proof. By Lemmas 43 and 19. The result also follows directly by a routine induction on the syntax of g. ■

It is now easy to state and prove a notion of *soundness* for ULAM:

THEOREM 55 (Soundness). *Suppose that ς is a ground substitution for Δ, t, and u. Suppose further that $a \notin fa(\varsigma(X))$ for every $a\#X \in \Delta$. Then:*

- *$\Delta \vdash a\#t$ implies $a \notin fa(t\varsigma)$.*
- *$\Delta \vdash_{\mathsf{ULAM}} t = u$ implies $t\varsigma =_{\alpha\beta} u\varsigma$.*

Proof. We proceed by induction on ULAM derivations.

We consider the rules in Figure 3 in turn:

- By assumption. We must show that if $a\#X \in \Delta$ then $a \notin fa(\varsigma(X))$, which follows by our assumptions on ς.
- The case (#**ab**). It is a fact of Definition 42 that $a \notin fa(b)$.
- The case (#λ**a**). It is a fact that $a \notin fa(\lambda a.(t\varsigma))$.
- The cases of (#λ**b**), (#**app**), and (#c) are no harder.
- The case (#**X**). By Lemma 54, $a \in fa(\pi \cdot g)$ if and only if $\pi^{-1}(a) \in fa(g)$.

We consider the rules in Figure 4 in turn:

- The cases (**refl**), (**symm**), (**tran**), (**cong**λ) and (**congapp**) follow by induction using the fact that $=_{\alpha\beta}$ is an equivalence relation and a congruence.
- The case (**perm**). Suppose that $a, b \notin fa(g)$. Then $(a\ b) \cdot g =_{\alpha\beta} g$ follows by Lemma 48, since $=_\alpha$ implies $=_{\alpha\beta}$.
- The case (**ax**). It remains to check the validity of the axioms of ULAM. It suffices to verify that:
  - (**var**$\mapsto$). We must show
    $$\pi(a)[\pi(a) \mapsto (\pi \cdot \sigma(X))\varsigma] =_{\alpha\beta} (\pi \cdot \sigma(X))\varsigma$$
    for any permutation π and substitution σ. This follows from the property that
    $$b[b \mapsto h] =_{\alpha\beta} h$$
    always (i.e. for any atom b and ground term h), which is a fact about $\alpha\beta$-equivalence.
  - (#$\mapsto$). Suppose that $(\pi \cdot \sigma(Z))[\pi(a) \mapsto \pi \cdot \sigma(X)] = \pi \cdot \sigma(Z)$ is derived from $\pi(a)\#\pi \cdot \sigma(Z)$ using the assumptions from Δ. By inductive hypothesis, we know $\pi(a) \notin fa((\pi \cdot \sigma(Z))\varsigma)$. We must show $((\pi \cdot \sigma(Z))\varsigma)[\pi(a) \mapsto (\pi \cdot \sigma(X))\varsigma] =_{\alpha\beta} (\pi \cdot \sigma(Z))\varsigma$. This follows from the basic fact about $\alpha\beta$-equivalence that
    $$b \notin fa(g) \quad \text{implies} \quad g[b \mapsto h] =_{\alpha\beta} g$$
    always.

    Other cases are similar:
  - (**app**$\mapsto$). $(g'g)[b \mapsto h] =_{\alpha\beta} (g'[b \mapsto h])(g[b \mapsto h])$.
  - (**abs**$\mapsto$). If $c \notin fa(h)$ then $(\lambda c.g)[b \mapsto h] =_{\alpha\beta} \lambda c.(g[b \mapsto h])$.
  - (**id**$\mapsto$). $g[b \mapsto b] =_{\alpha\beta} g$.

- The case (**fr**). If necessary, we rename the fresh atom in (**fr**) to a 'fresher' atom. By ZFA equivariance ([16, Appendix A]), the inductive hypothesis also holds for the 'freshened' derivation, and the result easily follows.

■

4.2 Completeness

We start with a weak form of completeness:

LEMMA 56. *For ground terms g and h, if $g =_{\alpha\beta} h$ then $\vdash_{\mathsf{ULAM}} g = h$.*

It is not hard to prove this by concrete calculations, but it also follows as a corollary of the following more powerful result:

THEOREM 57 (Completeness). *Suppose that $t\varsigma =_{\alpha\beta} u\varsigma$ for every substitution ς such that*

- *ς is ground for Δ, t, u and*
- *$a \notin fa(\varsigma(X))$ for every $a\#X \in \Delta$.*

Then $\Delta \vdash_{\mathsf{ULAM}} t = u$ is derivable.

The proof occupies the rest of this section. Fix some freshness context Δ and two terms t and u.

DEFINITION 58. Let $\mathcal{A}$ be the atoms mentioned anywhere in Δ, t, or u. Let $\mathcal{X}$ be the unknowns mentioned anywhere in Δ, t, or u. For each $X \in \mathcal{X}$ fix the following data:

- An order $a_{X1}, \ldots, a_{Xk_X}$ on the atoms in $\mathcal{A}$ such that $a\#X \notin \Delta$.
- Some entirely fresh atom c_X.

We write $\mathcal{C}$ for the set $\{c_X \mid X \in \mathcal{X}\}$.

DEFINITION 59. Specify ς a ground substitution for $\mathcal{X}$ by:

- $\varsigma(X) \equiv c_X a_{X1} \ldots a_{Xk_X}$ if $X \in \mathcal{X}$.
- $\varsigma(Y) \equiv Y$ for $Y \notin \mathcal{X}$.

LEMMA 60. *If $a\#X \in \Delta$ then $a \notin fa(\varsigma(X))$.*

Proof. By construction of $\varsigma(X)$. ■

By assumption $t\varsigma =_{\alpha\beta} u\varsigma$. The untyped λ-calculus is confluent (Theorem 52) so $t\varsigma$ and $u\varsigma$ rewrite to a common term, call it p:

DEFINITION 61. Fix two chains

$$t\varsigma \equiv g_1 =_\alpha g_2 \to_\beta g_3 =_\alpha g_4 \to_\beta \ldots \to_\beta g_{m-1} =_\alpha g_m \equiv p$$
$$u\varsigma \equiv h_1 =_\alpha h_2 \to_\beta h_3 =_\alpha h_4 \to_\beta \ldots \to_\beta h_{n-1} =_\alpha h_n \equiv p$$

Without loss of generality we assume these α-conversions and β-reductions do not introduce abstractions by atoms from $\mathcal{C}$.

DEFINITION 62. Let $\mathcal{A}^+$ be the set of *all* atoms mentioned anywhere in the chains of Definition 61, extended with a set of fresh atoms

$$\mathcal{B} = \{b_{Xi} \mid X, i \text{ such that } a_{Xi} \in \mathcal{A}\}.$$

in bijection with $\mathcal{A}$. (So $\mathcal{B}$ is disjoint from $\mathcal{C}$ and from all atoms mentioned in Δ, t, u, $g_1, \ldots, g_m$, $h_1, \ldots, h_n$.) Let Δ^+ be Δ enriched with freshness assumptions $a\#X$ for every $a \in \mathcal{A}^+ \setminus \mathcal{A}$ and every $X \in \mathcal{X}$.

DEFINITION 63. We call a ground term g **accurate** when:

- g mentions only atoms in $\mathcal{A}^+ \setminus \mathcal{B}$.
- $c_X \in \mathcal{C}$ never occurs abstracted. That is, no term contains 'λc_X'.
- $c_X \in \mathcal{C}$ appears, if it appears at all, in head position applied to a list of terms in a subterm of the form '$c_X g_1 \ldots g_{k_X}$'.

Note that $g_1, \ldots, g_m$ and $h_1, \ldots, h_n$ are accurate by construction of $t\varsigma$ and $u\varsigma$ and by the nature of α-conversions and β-reductions.

DEFINITION 64. Define an **inverse translation** from accurate terms to (possibly non-ground) terms inductively by:

$$a^{-1} \equiv a \quad (a \notin \mathcal{C}) \qquad (\lambda a.g)^{-1} \equiv \lambda a.(g^{-1}) \qquad (gh)^{-1} \equiv (g^{-1})(h^{-1}) \qquad \mathsf{c}^{-1} \equiv \mathsf{c}$$
$$(c_X)^{-1} \equiv \lambda b_{X1}.\cdots\lambda b_{Xk_X}.(b_{X1}\ a_{X1})\cdots(b_{Xk_X}\ a_{Xk_X})\cdot X \quad (c_X \in \mathcal{C})$$

The inverse translation $_^{-1}$ is an inverse of ς in the following sense:

LEMMA 65. *$\Delta^+ \vdash_{\mathsf{ULAM}} (t\varsigma)^{-1} = t$, and $\Delta^+ \vdash_{\mathsf{ULAM}} (u\varsigma)^{-1} = u$.*

Proof. We prove by induction that if v is a subterm of t or u then $\Delta^+ \vdash_{\mathsf{ULAM}} (v\varsigma)^{-1} = v$.

The only interesting case is when $v \equiv \pi \cdot X$. We must show

$$\Delta^+ \vdash_{\mathsf{ULAM}} (\pi_1 \cdot X)[b_{Xk_X} \mapsto \pi(a_{Xk_X})]\cdots[b_{X1} \mapsto \pi(a_{X1})] = \pi \cdot X$$

where $\pi_1 = (b_{X1}\ a_{X1})\cdots(b_{Xk_X}\ a_{Xk_X})$.

Take $\pi_2 = (b_{X1}\ \pi(a_{X1}))\cdots(b_{Xk_X}\ \pi(a_{Xk_X}))$. By transitivity it suffices to show:

1. $\Delta^+ \vdash_{\mathsf{ULAM}} (\pi_1 \cdot X)[b_{Xk_X} \mapsto \pi(a_{Xk_X})] \cdots [b_{X1} \mapsto \pi(a_{X1})] = (\pi_2 \circ \pi_1) \cdot X$.
2. $\Delta^+ \vdash_{\mathsf{ULAM}} (\pi_2 \circ \pi_1) \cdot X = \pi \cdot X$.

The first part follows using Lemma 30 when, for $1 \leq i \leq k_X$,

$$\Delta^+ \vdash \pi(a_{Xi}) \# (\pi_1 \cdot X)[b_{Xk_X} \mapsto \pi(a_{Xk_X})] \cdots [b_{Xi+1} \mapsto \pi(a_{Xi+1})].$$

By the rules for freshness, this follows from $\Delta^+ \vdash \pi(a_{Xi}) \# \pi_1 \cdot X$ since the $\pi(a_{X1}), \ldots, \pi(a_{Xk_X})$ are all disjoint. We conclude using a case distinction on $\pi(a_{Xi})$:

- $\pi(a_{Xi}) \neq a_{Xj}$ for all j: then $\pi(a_{Xi}) \# X \in \Delta^+$ since $\pi(a_{Xi}) \# X \in \Delta$.
- $\pi(a_{Xi}) = a_{Xj}$ for some j: then $b_{Xj} \# X \in \Delta^+$ by definition.

We still have to show that $\Delta^+ \vdash_{\mathsf{ULAM}} (\pi_2 \circ \pi_1) \cdot X = \pi \cdot X$. It is convenient to show the stronger property $\Delta^+ \vdash_{\mathsf{CORE}} (\pi_2 \circ \pi_1) \cdot X = \pi \cdot X$. By Theorem 35 we need only show that $\Delta^+ \vdash \mathrm{ds}(\pi_2 \circ \pi_1, \pi) \# X$. That is, we must show that $\Delta^+ \vdash a \# X$ for every a such that $(\pi_2 \circ \pi_1)(a) \neq \pi(a)$. We consider every possible a (every $a \in \pi_2 \circ \pi_1$ and $a \in \pi$):

- $a = b_{Xi}$: then $b_{Xi} \# X \in \Delta^+$ by definition, and the result follows.
- $a = a_{Xi}$: then $(\pi_2 \circ \pi_1)(a_{Xi}) = \pi(a_{Xi})$ and there is nothing to prove.
- $a = \pi(a_{Xi})$: then we distinguish two cases:
  - if $\pi(a_{Xi}) = a_{Xj}$ for some j, the result follows by the case of a_{Xi};
  - if $\pi(a_{Xi}) \neq a_{Xj}$ for all j, then $\pi(a_{Xi}) \# X \in \Delta^+$ by definition.
- $a \in \pi$, but $a \neq a_{Xj}$ for all j, then $a \# X \in \Delta^+$ by definition.

■

REMARK 66. The reader might wonder why the inverse mapping of the c_X renames a_{Xi} to the fresh b_{Xi}. Consider for example $(a_{X1}\ a_{X2}) \cdot X$ in the empty freshness context $\emptyset$, so we do not know $a_{X1} \# X$ or $a_{X2} \# X$. Then

$$((a_{X1}\ a_{X2}) \cdot X)\varsigma^{-1} \equiv ((b_{X1}\ a_{X1})(b_{X2}\ a_{X2}) \cdot X)[b_{X2} \mapsto a_{X1}][b_{X1} \mapsto a_{X2}].$$

By calculations we can verify Lemma 65:

$$\emptyset^+ \vdash_{\mathsf{ULAM}} ((a_{X1}\ a_{X2}) \cdot X)\varsigma^{-1} = (a_{X1}\ a_{X2}) \cdot X$$

where $\emptyset^+ = \{b_{X1} \# X, b_{X2} \# X, c_X \# X\}$. Had we left out the renaming to fresh atoms then $((a_{X1}\ a_{X2}) \cdot X)\varsigma^{-1}$ would be $X[a_{X2} \mapsto a_{X1}][a_{X1} \mapsto a_{X2}]$, which is not equal to $(a_{X1}\ a_{X2}) \cdot X$, since for example

$$((a_{X1}\ a_{X2}) \cdot X)[a_{X2}/X] = a_{X1} \text{ but } X[a_{X2} \mapsto a_{X1}][a_{X1} \mapsto a_{X2}][a_{X2}/X] = a_{X2}.$$

A technical lemma about freshness will be useful.

LEMMA 67. *Suppose that g is accurate. For any $a \in \mathcal{A}^+$, if $a \notin fa(g)$ then $\Delta^+ \vdash a\#g^{-1}$.*

Proof. By induction on g. The only non-trivial case is when $g \equiv c_X$. Suppose $a \notin fa(c_X)$. Then $a \neq c_X$ and we must show

$$\Delta^+ \vdash a\#\lambda b_{X1}.\cdots\lambda b_{Xk_X}.(\pi \cdot X),$$

where $\pi = (b_{X1}\ a_{X1})\cdots(b_{Xk_X}\ a_{Xk_X})$. We distinguish two cases:

- $a = b_{Xk_j}$ for some j: then $b_{Xk_j}\#\lambda b_{Xk_j}.\cdots\lambda b_{Xk_X}(\pi \cdot X)$ by $(\#\lambda\mathbf{a})$, and the result follows by the rules of freshness using the inductive hypothesis.
- $a \neq b_{Xk_j}$ for all j: then $\pi^{-1}(a) \neq a_{Xk_j}$ for all j, so $\pi^{-1}(a)\#X \in \Delta^+$ by definition. The result follows using the rules for freshness and the inductive hypothesis.

■

We need a technical lemma:

LEMMA 68. *Suppose that g is accurate. Suppose that π is a permutation such that $\pi(a) = a$ for all $a \notin \mathcal{A}^+ \setminus (\mathcal{B} \cup \mathcal{C})$. Then $\Delta^+ \vdash_{\mathsf{CORE}} (\pi \cdot g)^{-1} = \pi \cdot (g^{-1})$.*

Proof. By a routine induction on g. In the case of $g \equiv c_X$ we use the fact that $\pi(a) = a$ for all $a \in \mathcal{B} \cup \mathcal{C}$. ■

LEMMA 69. *Suppose that g and h are accurate. Then if $g =_\alpha h$ then $\Delta^+ \vdash_{\mathsf{CORE}} g^{-1} = h^{-1}$.*

Proof. By Theorem 49 $g =_\alpha h$ coincides with $\vdash_{\mathsf{CORE}} g = h$. We therefore work by induction on the structure of g using the syntactic criteria of Theorem 35.

The only non-trivial case is when

$$g \equiv \lambda a.g', \quad h \equiv \lambda b.h', \quad \vdash b\#g', \quad \text{and} \quad \vdash_{\mathsf{CORE}} (b\ a) \cdot g' = h'.$$

By assumption $a, b \in \mathcal{A}^+ \setminus (\mathcal{B} \cup \mathcal{C})$. By Lemma 67 we have $\Delta^+ \vdash b\#g'^{-1}$. By inductive hypothesis $\Delta^+ \vdash_{\mathsf{CORE}} ((b\ a) \cdot g')^{-1} = h'^{-1}$. By Lemma 68 we have $\Delta^+ \vdash_{\mathsf{CORE}} ((b\ a) \cdot g')^{-1} = (b\ a) \cdot (g'^{-1})$. From the rules for freshness and equality $\Delta^+ \vdash_{\mathsf{CORE}} \lambda a.(g'^{-1}) = \lambda b.(h'^{-1})$ follows. ■

LEMMA 70. *Suppose that $a \in \mathcal{A}^+ \setminus \mathcal{C}$. Suppose that g, h, and $g[h/a]$ are accurate. Then $\Delta^+ \vdash_{\mathsf{ULAM}} g^{-1}[a \mapsto h^{-1}] = (g[h/a])^{-1}$.*

Proof. We work by induction on the size of g:

- $a[h/a]$. $\Delta^+ \vdash_{\mathsf{ULAM}} a[a \mapsto h^{-1}] = h^{-1}$ by (**var**$\mapsto$).
- $b[h/a]$. $\Delta^+ \vdash_{\mathsf{ULAM}} b[a \mapsto h^{-1}] = b$ by (\#$\mapsto$) since $\Delta^+ \vdash a\#b$. The cases of $(\lambda a.g)[h/a]$ and $\mathsf{c}[h/a]$ are similar.
- $(\lambda b.g)[h/a]$ where $b \notin fa(h)$. We must show

  $$\Delta^+ \vdash_{\mathsf{ULAM}} (\lambda b.(g^{-1}))[a \mapsto h^{-1}] = \lambda b.((g[h/a])^{-1}).$$

  By the inductive hypothesis and the rules for equality, it suffices to show

  $$\Delta^+ \vdash_{\mathsf{ULAM}} (\lambda b.(g^{-1}))[a \mapsto h^{-1}] = \lambda b.(g^{-1}[a \mapsto h^{-1}]).$$

  By Lemma 67 we know $\Delta^+ \vdash b\#h^{-1}$. The result follows from (**abs**$\mapsto$).
- $(\lambda b.g)[h/a]$ where $b \in fa(h)$. By assumption $\lambda b.g$ is accurate, therefore $b \notin \mathcal{B} \cup \mathcal{C}$.

  Recall from Definition 44 that $(\lambda b.g)[h/a] \equiv \lambda c.(g[c/b][h/a])$ for some choice of fresh c (so $c \notin fa(g)$ and $c \notin fa(h)$). Now by assumption $\lambda c.(g[c/b][h/a])$ is accurate, so $c \notin \mathcal{B} \cup \mathcal{C}$ and $g[c/b][h/a]$ is accurate.

  We must show

  $$\Delta^+ \vdash_{\mathsf{ULAM}} (\lambda b.(g^{-1}))[a \mapsto h^{-1}] = \lambda c.((g[c/b][h/a])^{-1}).$$

  Note that by Lemma 67, $\Delta^+ \vdash c\#g^{-1}$ and $\Delta^+ \vdash c\#h^{-1}$, and therefore $\Delta^+ \vdash c\#\lambda b.(g^{-1})$ by (\#λ**b**). Also $\Delta^+ \vdash b\#\lambda b.(g^{-1})$ is immediate by (\#λ**a**). We present the rest of the proof in a calculational style:

  $$\begin{array}{ll} & \lambda c.((g[c/b][h/a])^{-1}) \\ = & \quad \{\ g[c/b] =_\alpha (c\ b) \cdot g \text{ by Lemma 48 since } c \notin fa(g)\ \} \\ & \lambda c.(((c\ b) \cdot g)[h/a])^{-1} \\ = & \quad \{\ \text{inductive hypothesis, since } (c\ b) \cdot g \text{ is accurate}\ \} \\ & \lambda c.(((c\ b) \cdot g)^{-1}[a \mapsto h^{-1}]) \\ = & \quad \{\ \text{Lemma 68}\ \} \\ & \lambda c.(((c\ b) \cdot (g^{-1}))[a \mapsto h^{-1}]) \\ = & \quad \{\ (\mathbf{abs}\mapsto), \text{ since } \Delta^+ \vdash c\#h^{-1}\ \} \\ & (\lambda c.((c\ b) \cdot (g^{-1})))[a \mapsto h^{-1}] \\ = & \quad \{\ (\mathbf{perm}) \text{ since } \Delta^+ \vdash b\#\lambda b.(g^{-1}) \text{ and } \Delta^+ \vdash c\#\lambda b.(g^{-1})\ \} \\ & (\lambda b.(g^{-1}))[a \mapsto h^{-1}] \end{array}$$

The result follows by transitivity.

- $(gg')[h/a]$. By the inductive hypothesis and the rules for equality, $\Delta^+ \vdash_{\mathsf{ULAM}} ((g^{-1})(g'^{-1}))[a \mapsto h^{-1}] = ((gg')[h/a])^{-1}$ follows from

$$\Delta^+ \vdash_{\mathsf{ULAM}} ((g^{-1})(g'^{-1}))[a \mapsto h^{-1}] = (g^{-1}[a \mapsto h^{-1}])(g'^{-1}[a \mapsto h^{-1}]).$$

  We conclude using axiom (**app**$\mapsto$).

- $c_X[h/a]$ where $c_X \in \mathcal{C}$. By assumption $a \neq c_X$, so we must show

$$\Delta^+ \vdash_{\mathsf{ULAM}} (\lambda b_{X1}.\cdots\lambda b_{Xk_X}.(\pi \cdot X))[a \mapsto h^{-1}] = \lambda b_{X1}.\cdots\lambda b_{Xk_X}.(\pi \cdot X),$$

  where $\pi = (b_{X1}\ a_{X1})\cdots(b_{Xk_X}\ a_{Xk_X})$.

  By assumption $b_{Xi} \notin h$ for all $1 \leq i \leq k_X$, therefore also $b_{Xi} \notin fa(h)$. By Lemma 67 also $\Delta^+ \vdash b_{Xi}\#h^{-1}$. Then we can show by a number of applications of axiom (**abs**$\mapsto$) that $(\lambda b_{X1}.\cdots\lambda b_{Xk_X}.(\pi \cdot X))[a \mapsto h^{-1}]$ is equal to $\lambda b_{X1}.\cdots\lambda b_{Xk_X}.(\pi \cdot X)[a \mapsto h^{-1}]$. By the rules for equality, it suffices to show

$$\Delta^+ \vdash_{\mathsf{ULAM}} (\pi \cdot X)[a \mapsto h^{-1}] = \pi \cdot X.$$

  By axiom (#$\mapsto$) this follows if $\pi^{-1}(a)\#X \in \Delta^+$. There are two possibilities:

  - $a = a_{Xj}$ for some j: then $\pi^{-1}(a) = b_{Xj}$, and $b_{Xj}\#X \in \Delta^+$ by definition.
  - $a \neq a_{Xj}$ for all j: then $\pi^{-1}(a) = a$, and $a\#X \in \Delta^+$ by definition.

The result follows. ■

COROLLARY 71. *Suppose that g and h are accurate. If $g \to_\beta h$ then $\Delta^+ \vdash_{\mathsf{ULAM}} g^{-1} = h^{-1}$.*

Proof. By induction on the derivation rules for $\to_\beta$ from Definition 51. It suffices to show the following (here g, g', h, h', and $g[h/a]$ are accurate and $a \in \mathcal{A}^+ \setminus \mathcal{C}$):

1. $\Delta^+ \vdash_{\mathsf{ULAM}} g^{-1}[a \mapsto h^{-1}] = (g[h/a])^{-1}$.
2. If $\Delta^+ \vdash_{\mathsf{ULAM}} g^{-1} = g'^{-1}$ then $\Delta^+ \vdash_{\mathsf{ULAM}} \lambda a.(g^{-1}) = \lambda a.(g'^{-1})$.
3. If $\Delta^+ \vdash_{\mathsf{ULAM}} g^{-1} = g'^{-1}$ and $\Delta^+ \vdash_{\mathsf{ULAM}} h^{-1} = h'^{-1}$ then $\Delta^+ \vdash_{\mathsf{ULAM}} (g^{-1})(h^{-1}) = (g'^{-1})(h'^{-1})$.

The first part is Lemma 70. The second and third parts follow by (**cong**λ) and (**congapp**). ■

We are now ready to prove Theorem 57:

Proof. Recall from Definition 61 the chains

$$\begin{aligned} t\varsigma &\equiv g_1 =_\alpha g_2 \to_\beta g_3 =_\alpha g_4 \to_\beta \ldots \to_\beta g_{m-1} =_\alpha g_m \equiv p \\ u\varsigma &\equiv h_1 =_\alpha h_2 \to_\beta h_3 =_\alpha h_4 \to_\beta \ldots \to_\beta h_{n-1} =_\alpha h_n \equiv p. \end{aligned}$$

By Lemma 69 and Corollary 71

$$\begin{aligned} \Delta^+ &\vdash_{\mathsf{ULAM}} (t\varsigma)^{-1} \equiv g_1^{-1} = g_2^{-1} = \ldots = g_m^{-1} \equiv p^{-1} \\ \Delta^+ &\vdash_{\mathsf{ULAM}} (u\varsigma)^{-1} \equiv h_1^{-1} = h_2^{-1} = \ldots = h_n^{-1} \equiv p^{-1}. \end{aligned}$$

By transitivity

$$\Delta^+ \vdash_{\mathsf{ULAM}} (t\varsigma)^{-1} = p^{-1} \quad \text{and} \quad \Delta^+ \vdash_{\mathsf{ULAM}} (u\varsigma)^{-1} = p^{-1}$$

so by symmetry and transitivity $\Delta^+ \vdash_{\mathsf{ULAM}} (t\varsigma)^{-1} = (u\varsigma)^{-1}$. By Lemma 65 then also $\Delta^+ \vdash_{\mathsf{ULAM}} t = u$. Since Δ^+ extends Δ with atoms that are not mentioned in t and u by Lemma 38 we conclude $\Delta \vdash_{\mathsf{ULAM}} t = u$ as required. ■

4.3 Conservativity over CORE

We can exploit the ς from Subsection 4.2 to prove conservativity of ULAM over CORE.

LEMMA 72. *Fix Δ. Suppose that t and u contain no subterm of the form $v[a \mapsto w]$. Then for ς the ground substitution constructed in Subsection 4.2, $t\varsigma$ and $u\varsigma$ are β-normal forms.*

Proof. $\varsigma(X) \equiv c_X a_{X1} \ldots a_{Xk_X}$ for every X appearing in Δ, t, or u. Applying this substitution to t and u cannot introduce subterms of the form $v[a \mapsto w]$. ■

THEOREM 73 (Conservativity). *Suppose that t and u contain no subterm of the form $v[a \mapsto w]$. Then*

$$\Delta \vdash_{\mathsf{ULAM}} t = u \qquad \textit{if and only if} \qquad \Delta \vdash_{\mathsf{CORE}} t = u.$$

Proof. A derivation in CORE is also a derivation in ULAM so the right-to-left implication is immediate.

Now suppose that $\Delta \vdash_{\mathsf{ULAM}} t = u$. We construct ς as in Subsection 4.2. By Theorem 55, $t\varsigma =_{\alpha\beta} u\varsigma$. By Lemma 72 we know that $t\varsigma$ and $u\varsigma$ are β-normal forms. By confluence of the λ-calculus (Theorem 52), $t\varsigma =_\alpha u\varsigma$.

We now prove $\Delta \vdash_{\mathsf{CORE}} t = u$ by induction on t. The calculations are detailed but entirely routine. We consider just one case, the hardest one:

Suppose $t \equiv \pi \cdot X$. Then $t\varsigma \equiv c_X\pi(a_{X1}) \ldots \pi(a_{Xk_X})$. By Theorem 35, if $t\varsigma =_\alpha u\varsigma$ it *must* be that $u\varsigma \equiv c_X\pi(a_{X1}) \ldots \pi(a_{Xk_X})$.

By the construction of $u\varsigma$ and the way we chose $a_{X1}, \ldots, a_{Xk_X}$ to be the atoms mentioned in Δ, t, or u which are *not* provably fresh for X in Δ, it follows that u must have been equal to $\pi' \cdot X$, for some π', such that $\Delta \vdash \mathrm{ds}(\pi, \pi')\#X$. It follows that $\Delta \vdash_{\mathsf{CORE}} t = u$ as required. ■

5 Conclusions

5.1 Related work not using nominal techniques

$\alpha\beta$-equivalence on λ-calculus syntax is a good idea; we find it realised in different ways in different systems. If the λ-calculus syntax in question serves as the language of a logic, then $\alpha\beta$-equivalence may have the status of axioms. For example Andrews's logic $\mathcal{Q}_0$ [1, §51] contains five axioms ($\mathbf{4_1}$), ($\mathbf{4_2}$), ($\mathbf{4_3}$), ($\mathbf{4_4}$), and ($\mathbf{4_5}$) ([1, page 164]). In fact they are axiom *schemes*, containing meta-variables $\mathbf{A}$ and $\mathbf{B}$ in the informal meta-level ranging over terms (and also meta-variables x, y ranging permutatively over variable-symbols). The relationship which Figure 1 bears to them is clear but here, axioms feature in the formal framework of Nominal Algebra — a formal logic, not an informal meta-level. We claim that Nominal Algebra captures part of the 'informal meta-level' in which researchers routinely work — and that within that, ULAM captures the (untyped) λ-calculus.

Salibra's Lambda Abstraction Algebras [28] axiomatise the λ-calculus using universal algebra. The method is cylindric in the sense of cylindric algebras [18]; abstraction is represented by infinitely many term-formers (there is a term-former 'λa' for every a) and freshness is encoded in the structure of the terms in the following sense: consider Salibra's rule ($\beta_{\mathbf{4}}$) from [28, page 6]:

$$(\beta_{\mathbf{4}}) \qquad (\lambda x.(\lambda x.\xi))\mu = \lambda x.\xi.$$

We can rewrite this in our notation as $(\lambda a.Z)[a \mapsto X] = \lambda a.Z$ and this is a version of $(\#\mapsto)$, where the freshness condition $a\#Z$ has been built into the structure of the term by replacing Z with $\lambda a.Z$. Similarly Salibra's rule (α)

$$(\alpha) \qquad \lambda x.(\lambda y.\xi)z = \lambda y.(\lambda x.(\lambda y.\xi)z)y$$

can be rewritten in our notation as $\lambda a.(Z[b \mapsto c]) = \lambda b.(Z[b \mapsto c][a \mapsto b])$ and this plays an analogous rôle to our Lemma 28. The notion of *dimension set* [28, Definition 4] corresponds to freshness. The proof of a main result, Theorem 13 of [28] (first carried out by Salibra, improved by Selinger), uses definitions reminiscent of, though not identical to, Definitions 58 and 59.

The 'nominal' techniques give a clear separation of the parts having to do with names and abstraction, and the parts having to do with λ and application, which is not possible with Lambda Abstraction Algebras. However, Salibra's results [28, 22] show what can be achieved using algebraic methods.

Curry discovered *combinatory algebra* [10]. The signature contains a binary term-former *application* and two constants S and K. Axioms are $Kxy = x$ and $Sxyz = (xz)(yz)$. This syntax is parsimonious and the axioms are compact, but it is not natural or ergonomic to program in; that most ergonomic feature of the λ-calculus which makes it so very useful in practice, the λ, is missing. There is also a mathematical issue: the natural encoding of closed λ-terms into combinatory algebra syntax ([6, Section 7] or [29, Subsection 1.4]) does not map $\alpha\beta$-equivalent λ-terms ($\lambda z.(\lambda x.x)z$ and $\lambda z.z$) to provably equal terms in combinatory algebra. This is resolved if we strengthen combinatory algebra to *lambda algebra* by adding five axioms, due to Curry [29, Proposition 5]. However the translation is still not sound, in the sense that there exist λ-terms M and N such that the translation of M is derivably equal to the translation of N, but the translation of $\lambda x.M$ is not derivably equal to the translation of $\lambda x.N$. To ensure soundness, we must add the Meyer-Scott axiom [29, Proposition 20] (Selinger calls it 'the notorious rule').

In short, combinators do not capture the model of functions expressed by the λ-calculus. Selinger [29] identifies the problem with the interpretation of variables and argues for denotations with fresh 'indeterminates'; this reminds us of nominal techniques with its set of atoms in the denotation and well-developed theories of freshness.

λ-calculi of explicit substitution decompose the substitution used in β-reduction into many explicit reduction steps [21]. This is similar to the way ULAM breaks down the calculation of an equality into many explicit algebraic equalities. A calculus of explicit substitutions is a calculus, not a logic; all reasoning using calculi occurs informally in natural language. ULAM on the other hand is intended to support algebraic reasoning on the λ-calculus within a formal framework, while remaining very close to informal practice.

5.2 Related work using nominal techniques

The first application of nominal techniques was to datatypes of syntax with binding [17]. The syntax of the untyped λ-calculus is often used as a paradigmatic example of such a datatype. This paper is *not* another such study. 'The λ-calculus' studied in some other publications — using nominal sets [17], nominal logic [27], and also using higher-order abstract syntax [26], de Bruijn terms [11], and so on — is a collection of (syntax) trees. This paper

studies functions.

This paper is part of a broader research programme developing nominal techniques in general and nominal algebra in particular. A rewrite system for the λ-calculus appeared already in [13] but without any statement or proof of completeness (indeed, the system considered there was not complete). The authors have axiomatised first-order logic as a theory FOL [16] and also substitution as a theory SUB [15]. Since β-reduction is based on substitution, this paper shares technical results with the study of SUB [15]. The completeness result we give for ULAM is for a model based on untyped λ-terms quotiented by $\alpha\beta$-equality. The completeness result for SUB is for a model based on trees with an explicit substitution. Derivable equality in ULAM includes the full power of the λ-calculus and is undecidable; equality in SUB is equality of syntax with a substitution action and is decidable. For example

$$\vdash_{\mathsf{ULAM}} (ba)[b \mapsto \lambda a.a] = a$$

is derivable in ULAM, but (in notation from [15]) only

$$\vdash_{\mathsf{SUB}} \mathsf{app}(b, a)[b \mapsto \mathsf{lam}([a]a)] = \mathsf{app}(\mathsf{lam}([a]a), a)$$

is derivable in SUB. The proofs in this paper have been improved and simplified. Technical issues have been avoided and we do not rely on a strong normalisation property which the proofs of [15] required (probably unnecessarily). Note that the treatment of SUB is parametric over a range of signatures and would be the appropriate theory where substitution is precisely what we require — for example as part of an axiomatisation of quantifiers in first-order logic.

A version of ULAM for a *typed* λ-calculus should be possible. It would sit 'in between' ULAM and SUB, as one might expect, but to make this formal a typing system for nominal terms is required (such that atoms are assigned types by a typing context). This has been investigated to some extent [12]. Investigations in nominal algebra are for future work.

Nominal algebra has a cousin, nominal equational logic (**NEL**) [9], which was derived from nominal algebra but making different design decisions. NEL satisfies a completeness result for a generic class of models in nominal sets [9], as does nominal algebra [14, 23]. The completeness result of this paper is much stronger than the generic results, because it is completeness for a single elementary model; similarly for the authors' treatments of substitution [15] and first-order logic [16]. We know of no like treatments of substitution, logic, and the λ-calculus in NEL. If and when this is done it will be interesting to compare the results.

5.3 Future work

The nominal algebraic framework proved itself capable of translating, with remarkable accuracy and uniformity, informal mathematical specification into formal nominal algebra axioms, almost symbol for symbol. This paper is both a study of the algebraisation of functions using nominal algebra, and a case study in nominal techniques applied to the λ-calculus. We hope that future study of nominal techniques may benefit from the 'off-the-shelf' axiomatisation provided in this paper. We have proved a soundness result (Theorem 55) and a strong completeness result (Theorem 57). We hope that as the theory of nominal algebra itself is improved, this will be of direct benefit to λ-calculus theory.

It would be interesting to consider direct 'nominal' versions of models of the λ-calculus, such as graph models or domain models of the λ-calculus [6, 30]. It would also be interesting to consider work using the language of categories (for example [29, 4]) using categories based on nominal sets.

It remains to develop the theory of nominal algebra itself, such as to prove the HSP theorem [8]; we would then be able to apply it to ULAM to study the λ-calculus, and likewise for other nominal algebraic theories.

It is also possible to investigate how well nominal algebra, or a system like it, can serve as the basis of a theorem-prover. Theorem-provers based on the λ-calculus [3] are the state of the art, and the Isabelle theorem-prover demonstrates how a weak meta-logic (such as Isabelle/Pure) can encode powerful object-logics (such as Isabelle/HOL) [24]. Is it possible that an elaboration of Nominal Algebra could serve as the foundation of a generic theorem-prover in the spirit of Isabelle, offering a new set of reasoning-principles 'ϵ away from informal practice'? The construction of ULAM in nominal algebra is a useful preliminary step.

5.4 Conclusions

The λ-calculus is a fundamental model of functions in logic and computation. We have given axioms ULAM for the untyped λ-calculus in *nominal algebra*, a recently-developed logical framework based on nominal techniques. This gives a logical theory pleasingly close to informal practice, while remaining mathematically completely rigorous. ULAM completes a trio of papers on nominal algebra and first-order logic [16], substitution [15], and with this paper, the λ-calculus. Researchers using nominal techniques might find ULAM and its completeness result a useful off-the-shelf component in later and larger works.

ULAM formally connects 'nominal atoms' and 'λ-calculus variables'. Discussions about this connection — usually based on not-entirely-explicit criteria of practical usefulness — have continued since nominal techniques

were introduced [17] and they may continue into the forseeable future. ULAM makes a nice mathematical contribution to this discussion; the axioms of ULAM are an algebraic measure of the distance we must travel from nominal-style atoms and atoms-abstraction, to λ-calculus style variables and λ-binding. This is gives new sense of how nominal techniques fit into a long tradition of functions in logic and computation.

BIBLIOGRAPHY

[1] Peter B. Andrews. *An introduction to mathematical logic and type theory: to truth through proof*. Academic Press, 1986.

[2] Peter B. Andrews. Classical type theory. In *Handbook of Automated Reasoning*, volume 2, chapter 15, pages 965–1007. Elsevier Science, 2001.

[3] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[4] Giulio Manzonetto Antonio Bucciarelli, Thomas Ehrhard. Not Enough Points Is Enough. In *Computer Science Logic*, pages 298–312, 2007.

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

[6] H. P. Barendregt. *The Lambda Calculus: its Syntax and Semantics (revised ed.)*. North-Holland, 1984.

[7] J. Barwise. An Introduction to First-Order Logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 5–46. North Holland, 1977.

[8] S. Burris and H. Sankappanavar. *A Course in Universal Algebra*. Graduate texts in mathematics. Springer, 1981.

[9] Ranald A. Clouston and Andrew M. Pitts. Nominal Equational Logic. *Electronic Notes in Theoretical Computer Science*, 172:223–257, 2007.

[10] Haskell B. Curry and R. Feys. *Combinatory Logic*, volume 1. North Holland, 1958.

[11] N. G. de Bruijn. Lambda Calculus Notation With Nameless Dummies, A Tool For Automatic Formula Manipulation, With Application To The Church-Rosser Theorem. *Indagationes Mathematicae*, 5(34):381–392, 1972.

[12] Maribel Fernández and Murdoch J. Gabbay. Curry-style types for nominal terms. In *Types for Proofs and Programs (proceedings of TYPES'06)*, volume 4502 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2007.

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

[14] 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, 2007.

[15] Murdoch J. Gabbay and Aad Mathijssen. Capture-Avoiding Substitution as a Nominal Algebra. *Formal Aspects of Computing*, 20(4-5):451–479, August 2008.

[16] Murdoch J. Gabbay and Aad Mathijssen. One-and-a-halfth-order Logic. *Journal of Logic and Computation*, 18(4):521–562, August 2008.

[17] Murdoch J. Gabbay and A. M. Pitts. A New Approach to Abstract Syntax with Variable Binding (journal version). *Formal Aspects of Computing*, 13(3–5):341–363, 2001.

[18] L. Henkin, J. D. Monk, and A. Tarski. *Cylindric Algebras*. North Holland, 1971 and 1985. Parts I and II.

[19] Wilfrid Hodges. Elementary predicate logic. In D.M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition*, volume 1, pages 1–131. Kluwer, 2001.

[20] Daniel Leivant. Higher order logic. In D. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 2, pages 229–322. Oxford University Press, 1994.
[21] Pierre Lescanne. From Lambda-sigma to Lambda-upsilon: a Journey Through Calculi of Explicit Substitutions. In *POPL*, pages 60–69. ACM, 1994.
[22] G. Manzonetto and A. Salibra. Boolean Algebras for Lambda Calculus. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 317–326. IEEE Computer Society, 2006.
[23] Aad Mathijssen. *Logical Calculi for Reasoning with Binding*. PhD thesis, Technische Universiteit Eindhoven, 2007.
[24] Lawrence C. Paulson. The Foundation of a Generic Theorem Prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.
[25] Lawrence C. Paulson. *ML for the working programmer (2nd ed.)*. Cambridge University Press, 1996.
[26] F. Pfenning and C. Elliot. Higher-Order Abstract Syntax. In *PLDI (Programming Language design and Implementation)*, pages 199–208. ACM Press, 1988.
[27] A. M. Pitts. Nominal Logic, A First Order Theory of Names and Binding. *Information and Computation*, 186(2):165–193, 2003.
[28] Antonino Salibra. On the algebraic models of lambda calculus. *Theoretical Computer Science*, 249(1):197–240, 2000.
[29] Peter Selinger. The lambda calculus is algebraic. *Journal of Functional Programming*, 12(6):549–566, 2002.
[30] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, Cambridge, MA, USA, 1977.
[31] Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison Wesley, 1996.
[32] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal Unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.