A nominal axiomatisation of the lambda-calculus

Murdoch J. Gabbay[1,2]

School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, Great Britain

Aad Mathijssen[3]

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

The lambda-calculus is a fundamental syntax in computer science. It resists an algebraic treatment because of capture-avoidance side-conditions.
Nominal algebra is a logic of equality designed with formalisation of specifications involving binding in mind. In this paper we axiomatise the lambda-calculus using nominal algebra, demonstrate how proofs with these axioms reflect the informal arguments on syntax, and we prove the axioms sound and complete.
This makes a formal connection between a 'nominal' approach to variables, and the more traditional view of variables as a syntactic convenience for describing functions.

Keywords: lambda calculus, equational logic, nominal techniques

1 Introduction

Functions are widely-used in what we now call computer science; a development which can be traced back to Church [Chu40]. They are the basis of functional programming languages [Pau96,Tho96]; they also find application in logic [Bar77,Lei94], theorem-provers [ABI$^+$96,Pau89], rewriting [BN98], and more. Functions are a basic mathematical entity of computer science. Three techniques are widely used to study basic entities of this kind:

(1) Operational: define a syntax; study congruences or rewrite relations.

(2) Denotational: define an intended model or class of models; study that.

(3) Logical: write axioms; study derivability, soundness, and completeness.

[1] We thank an anonymous referee of a previous paper for comments which set us on the path to writing this one.

[2] Homepage: http://www.gabbay.org.uk

[3] Email: a.h.j.mathijssen@tue.nl

The syntax of the λ-calculus with $\alpha\beta$-equivalence, or more generally λ-theories [Bar84, Chapter 4] (congruences that include $\alpha\beta$) are instances of approach (1). Scott domain models and graph models [Bar84] are instances of approach (2).

Approach (3) has λ-algebras [Sel02], Salibra's lambda-abstraction algebras [Sal00], and (if one does not care about representing λ-abstraction, which we do) λ-lifting [Joh85].

A recent logic by the authors can be applied to Approach (3) above, i.e. to give a logical axiomatisation of functions: *nominal algebra*. Nominal algebra is a form of universal algebra [Mat07,GM07a] with built-in support for names, binding, and freshness conditions. It has semantics in nominal sets [GP02], which in a suitable mathematical sense also have built-in support for names, binding, and freshness conditions.

One benefit of nominal algebra is that it is easy to write down plausible axioms for the λ-calculus — they look just like well-known informal $\alpha\beta$-equivalences, see the nominal algebra theory ULAM in Figure 1. The signature has term-formers for application (binary, taking two terms), λ (unary, but taking an abstraction) and constants (nullary). [4]

A general nominal algebra semantics in nominal sets immediately gives a notion of model: a model of ULAM is a nominal set with functions on it to interpret application, λ and constants, such that the equalities in ULAM are valid for every valuation of unknowns to denotational elements that satisfies the freshness side-conditions.

Plausible this might be ... but is it correct? Let us take λ-terms quotiented by $\alpha\beta$-equivalence as our yardstick for correctness. Then we have three questions:

- If we quotient λ-terms by $\alpha\beta$-equivalence do we obtain, in some natural way, a model of ULAM?
- Are the equalities which are valid in all models of ULAM, precisely the equalities described by that one set or is there something missing? (Is ULAM *complete* for $\alpha\beta$?)
- Does reasoning in the logic capture a useful fragment of the kind of reasoning steps we would like to represent?

A comparison with numbers (another basic mathematical entity) may be instructive: *Operational:* A rewrite system for arithmetic terms. *Denotational:* Numbers, as in 0, 1, 2, ... This will be our yardstick. *Logical:* Peano arithmetic. And for our questions: Numbers *are* a model of Peano arithmetic. Famously, Peano arithmetic is *not* complete for this model. However, Peano arithmetic *does* capture a useful fragment of arithmetic reasoning, and it is widely used.

In this paper we explore ULAM (Figure 1) and explore to what extent it and the nominal algebra framework in which it is embedded, capture 'the λ-calculus and its theory'. We will demonstrate with examples how ULAM expresses informal reasoning as formal derivations. We will demonstrate that ULAM *is* a theory for the untyped λ-calculus, in the sense that it is sound and complete for λ-terms quotiented by $\alpha\beta$-equivalence.

This investigation brings several benefits. The axioms of ULAM are finite: there is no need for infinitely many term-formers, variables, etc. This is itself a symptom of the fact that issues of binding and α-conversion are systematically dissected and uniformly handled by a nominal algebra framework which serves as a primitive level within which functions, as well as other things which tend to exhibit binding at the syntactic level, can be studied. This framework is compatible with a broader body of research, such as nominal unification

[4] For full definitions, terminology, and more on the general theory of nominal algebra see [Mat07,GM07a].

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

Figure 1. Axioms of ULAM

and nominal rewriting [UPG04,FG07], with good computational properties. Semantics are based on a by now well-understood model in nominal sets [GP02,GM07a] — in this paper we only study the one syntactic model we need for completeness, however general nominal sets models exist and can be investigated in future work. For more comment see Future Work.

Some perspective on the current literature may be helpful. Traditionally variables are considered a syntactic adjunct to functions; variables are symbols intended to denote elements of the underlying domain, but they do not themselves inhabit the underlying domain. On the other hand, nominal techniques (of which this paper is a part) subscribe to a mathematical view in which names are entities in the denotation. This is based on modelling names as atoms (urelemente) in set theory [Bru96,Gab00], an idea originally raised in the Gabbay-Pitts model of α-abstraction and syntax-with-binding and also used to develop the Gabbay-Pitts И quantifier useful for reasoning on this model of syntax [GP02]. These ideas find uses beyond the initial applications to syntax: in game theory and reasoning about pointers [AGM$^+$04,Tze07,BL05], spatial logics [CC04], and more.

Thus, there is now a body of work based on atoms, some concerned with reasoning on syntax-with-binding, some concerned with representing other things. Nevertheless, it always treats atoms as denotational entities in their own right rather than as a purely syntactic adjunct to a notion of function. In this paper we state and prove a fundamental correctness result: we write down ULAM and show that it soundly and completely expresses the properties which, when added to nominal-style atoms, convert them into λ-calculus style variables. Conveniently, nominal-style α-abstraction then becomes λ-calculus style functional abstraction.

Nota bene: Nominal techniques were first applied to construct datatypes of syntax-with-binding [GP02] with good inductive reasoning principles. One datatype often used is λ-term (up to α-equivalence). This paper is not another such study, like those in nominal sets [GP02], higher-order abstract syntax [PE88], de Bruijn terms [dB72], and so on — which are about collections of syntax trees.

2 Nominal algebra

In this section, we present the proof theory of nominal algebra. It consists of an equational logic on nominal terms, and has built-in support for binding, freshness and meta-variables.

2.1 Nominal terms

We define a syntax of nominal terms tailored to our λ-calculus application; general treatments are elsewhere [GM08,Mat07].

Definition 2.1 Fix the following disjoint sets:

- a countably infinite set of **atoms** $a, b, c, \ldots \in \mathbb{A}$ representing object-level variables;
- a countably infinite collection of **unknowns** $X, Y, Z, \ldots$ representing unknown elements in nominal algebra axioms;
- a possibly infinite collection of **constant symbols** $\mathsf{c} \in \mathsf{C}$.

We let $a, b, c, \ldots$ range *permutatively* over atoms unless stated otherwise. For example in $(\#\mathbf{ab})$ and $(\#\lambda\mathbf{b})$ from Figure 2, and in $(\mathbf{perm})$ from Figure 3, a and b represent two *distinct* atoms. Besides being useful in what follows, this models common practice: if we ask the reader to 'consider two variable symbols x and y' then we have no control over, for example, their handwriting, and thus the symbols they actually commit to the page. All that matters is that the two variable symbols are *different*.

We set about constructing the machinery of nominal algebra.

Definition 2.2 A **permutation** π of atoms is a bijection on atoms with **finite support**, which means that the set $\mathsf{supp}(\pi)$, defined by $\{a \mid \pi(a) \neq a\}$, is finite. In words: For 'most' atoms π is the identity.

Definition 2.3 **Terms** t, u, v are inductively defined by:

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

We write **syntactic identity** of terms t, u as $t \equiv u$.

A typed syntax is possible; see [FG06]. Types would cause no essential difficulties for the results to follow.

2.2 Permutation, substitution, freshness

Definition 2.4 Write id for the **identity** permutation on atoms. Write 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, i.e. $id \circ \pi = \pi$ and $\pi \circ id = \pi$.

Write $(a\ b)$ for the permutation that **swaps** a and b, i.e. $(a\ b)(a) = b$, $(a\ b)(b) = a$, and $(a\ b)(c) = c$. We may omit $\circ$ between swappings, writing $(a\ b) \circ (b\ c)$ as $(a\ b)(b\ c)$, and we may write X as shorthand for $id \cdot X$.

Definition 2.5 Define the set $\mathsf{Atms}(t)$ of atoms that occur anywhere in t inductively by:

$$\mathsf{Atms}(a) = \{a\} \qquad \mathsf{Atms}(\pi \cdot X) = \mathsf{supp}(\pi) \qquad \mathsf{Atms}(\mathsf{c}) = \emptyset$$
$$\mathsf{Atms}(\lambda a.t) = \mathsf{Atms}(t) \cup \{a\} \qquad \mathsf{Atms}(t't) = \mathsf{Atms}(t') \cup \mathsf{Atms}(t)$$

We also write $\mathsf{Atms}(t_1, \ldots, t_n)$ as a shorthand for $\mathsf{Atms}(t_1) \cup \cdots \cup \mathsf{Atms}(t_n)$.

$$\frac{}{a\#b}\,(\#\mathbf{ab}) \qquad \frac{\pi^{\text{-}1}(a)\#X}{a\#\pi\cdot X}\,(\#\mathbf{X}) \qquad \frac{}{a\#\mathsf{c}}\,(\#\mathsf{c})$$

$$\frac{}{a\#\lambda a.t}\,(\#\lambda\mathbf{a}) \qquad \frac{a\#t}{a\#\lambda b.t}\,(\#\lambda\mathbf{b}) \qquad \frac{a\#t' \; a\#t}{a\#t't}\,(\#\mathbf{app})$$

Figure 2. Freshness derivation rules for nominal terms

Definition 2.6 Define a **permutation action** $\pi \cdot t$ by:

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

Note that in the clause for λ, π acts also on the 'a'. For example $(a\ b)\cdot\lambda a.X \equiv \lambda b.(a\ b)\cdot X$.

Definition 2.7 Call a **substitution** σ a function from unknowns to terms. Write $[t/X]$ for the substitution mapping X to t, and mapping Y to Y for all other Y.

Definition 2.8 Define a **substitution action** $t\sigma$ by:

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

For example:

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

Substitution does not avoid capture but when σ encounters $\pi\cdot X$ the permutation is applied to $\sigma(X)$.

Definition 2.9 A **freshness** is a pair $a\#t$ of an atom and a term. Call a freshness of the form $a\#X$ (so $t\equiv X$) **primitive**. 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 $a\#X, b\#X$.

Definition 2.10 Define **derivability on freshnesses** by the rules in Figure 2. In this figure, a and b permutatively range over atoms, t and t' range over nominal terms, π over permutations of atoms, X over unknowns, and c over constants.

Write $\Delta\vdash a\#t$ when a derivation of $a\#t$ exists using these rules such that the assumptions are elements of Δ. We usually write $\emptyset\vdash a\#t$ as $\vdash a\#t$.

For example from the rules in Figure 2, $\vdash a\#\lambda b.b$, $\vdash a\#\lambda a.a$, and $a\#X\vdash a\#X(\lambda a.Y)$ are all derivable.

2.3 Equality, axioms, theories

Definition 2.11 An **equality** is a pair $t=u$. An **axiom** is a pair $\nabla\vdash t=u$ of a freshness context ∇ and an equality $t=u$. We may write $\emptyset\vdash t=u$ as $\vdash t=u$.

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

$$\frac{}{t=t}(\mathbf{refl}) \qquad \frac{t=u}{u=t}(\mathbf{symm}) \qquad \frac{t=u \quad u=v}{t=v}(\mathbf{tran}) \qquad \frac{a\#t \quad b\#t}{(a\ b)\cdot t=t}(\mathbf{perm})$$

$$\frac{t=u}{\lambda a.t=\lambda a.u}(\mathbf{cng}\lambda) \qquad \frac{t'=u' \quad t=u}{t't=u'u}(\mathbf{cngapp})$$

$$\frac{\{a\#\sigma(X) \mid a\#X \in \nabla\}}{\pi\cdot(t\sigma)=\pi\cdot(u\sigma)}(\mathbf{ax}_{\nabla\vdash\mathbf{t=u}}) \qquad \frac{\begin{matrix}[a\#X]\\ \vdots \\ t=u\end{matrix}}{t=u}(\mathbf{fr}) \quad (a \notin \mathsf{Atms}(t,u))$$

Figure 3. Derivation rules for nominal equality

- CORE: the empty set of axioms.
- ULAM: the axioms from Figure 1. In this figure the a and b are *specific* atoms and the X, Z and Z' are *specific* unknowns.

Remark 2.12 We obtain an extensional version of ULAM if we add the axiom

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

to Figure 1. We see no difficulties in adapting the results and proofs in this paper to the extensional case.

Definition 2.13 Define **derivability on equalities** by the rules in Figure 3. In this figure, a and b permutatively range over atoms, t, t', u and u' range over nominal terms, X over unknowns, ∇ over freshness contexts, π over permutations, and σ over substitutions.

Write $\Delta \vdash_{\mathsf{T}} t=u$ when a derivation of $t=u$ exists using these rules such that:

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

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

We discuss the most interesting rules of Figure 3:

- $(\mathbf{ax}_{\nabla\vdash\mathbf{t=u}})$. This rule expresses how we obtain instances of axioms: instantiate unknowns by terms (using substitutions) and rename atoms (using permutations).

  We might expect the premises of the axiom rule to be $\{\pi(a)\#\pi\cdot\sigma(X) \mid a\#X \in \nabla\}$. Both versions are correct, because of *equivariance*:

$$\Delta \vdash a\#t \quad \text{if and only if} \quad \Delta \vdash \pi(a)\#\pi\cdot t$$

  for any Δ, a, t and π. This is characteristic of nominal techniques (e.g. [UPG04, Lemma 2.7], [FG07, Lemma 20], [GM07c, Appendix A], [GP02, Lemma 4.7]).
- $(\mathbf{fr})$. This introduces a fresh atom into the derivation. Square brackets denote *discharge* of the assumption. We can *always* find a fresh atom no matter how unknowns are instan-

tiated, since our syntax is finite and must mention finitely many atoms.

(**fr**) adds no deductive power to CORE but it does in the presence of axioms; a full discussion with an example is in [Mat07, Lemma 2.3.18].

- (**perm**). This rule expresses α-equivalence (see Lemma 2.16 and Theorem 3.7). For instance, (**perm**) allows us to show the following standard α-equivalence property:

$$\dfrac{\dfrac{\dfrac{}{a\#b}(\#\mathbf{ab})}{a\#\lambda b.b}(\#\lambda\mathbf{b}) \qquad \dfrac{}{b\#\lambda b.b}(\#\lambda\mathbf{a})}{\lambda a.a = \lambda b.b}(\mathbf{perm}) \quad (\lambda a.a \equiv (a\ b)\cdot\lambda b.b)$$

(**perm**) captures several rules from [UPG04, Figure 2] (but not in a syntax-directed manner).

- (**refl**). Choosing $a, b \not\in \mathsf{Atms}(t)$ we can construct the derivation sketched below:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{a\#(a\ b)\cdot t} \qquad \dfrac{\vdots}{b\#(a\ b)\cdot t}}{t = (a\ b)\cdot t}(\mathbf{perm}) \qquad \dfrac{\dfrac{\vdots}{a\#t} \qquad \dfrac{\vdots}{b\#t}}{(a\ b)\cdot t = t}(\mathbf{perm})}{t = t}(\mathbf{tran})}{t = t}(\mathbf{fr}) \quad \text{(introducing } a, b\#X \text{ for all unknowns } X \text{ in } t\text{)}$$

So we can view (**refl**) as sugar, but (if only for cleaner example derivations) we retain it. All our proofs treat (**refl**) as a 'real' rule.

Remark 2.14 Perhaps remarkably, nominal algebra is algebraic even though the judgement-form $\Delta \vdash t = u$ has '$\Delta \vdash$', which looks like an implication. It is sound and complete for a notion of model in nominal sets [Mat07]. In addition, models are closed under notions of product, subalgebra, quotient (just like for traditional algebra) — and a nominal sets notion we call *atoms-abstraction*. A version of the HSP theorem (Birkhoff's theorem) holds; any class of nominal algebra models closed under product, subalgebra, quotient, and atoms-abstraction, is characterised by a nominal algebra theory [GM07b]. So, perhaps unexpectedly, nominal algebra retains much of the flavour and mathematical properties of universal algebra.

Example 2.15 In ULAM we can prove β-equivalences as illustrated in Figure 4 — we choose one requiring an α-conversion.

We now consider some useful examples of derivations in the presence of unknowns:

Lemma 2.16 $b\#X \vdash_{\mathsf{CORE}} (\lambda a.X)Y = (\lambda b.((b\ a)\cdot X))Y$.

Proof We give the derivation in full:

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a\#b}\,(\#\mathbf{ab})}{a\#\lambda c.b}\,(\#\lambda\mathbf{b}) \quad \dfrac{}{c\#\lambda c.b}\,(\#\lambda\mathbf{a})}{\lambda a.b=\lambda c.b}\,(\mathbf{perm})}{\lambda b.\lambda a.b=\lambda b.\lambda c.b}\,(\mathbf{cng}\lambda) \quad \dfrac{}{a=a}\,(\mathbf{refl})}{(\lambda b.(\lambda a.b))a=(\lambda b.(\lambda c.b))a}\,(\mathbf{cngapp}) \quad \dfrac{\dfrac{\dfrac{}{c\#a}\,(\#\mathbf{ab})}{(\lambda b.(\lambda c.b))a=\lambda c.((\lambda b.b)a)}\,(\mathbf{ax}_{\beta\mathbf{abs}}) \quad \dfrac{\dfrac{}{(\lambda b.b)a=a}\,(\mathbf{ax}_{\beta\mathbf{var}})}{\lambda c.((\lambda b.b)a)=\lambda c.a}\,(\mathbf{cng}\lambda)}{(\lambda b.(\lambda c.b))a=\lambda c.a}\,(\mathbf{tran})}{(\lambda b.(\lambda a.b))a=\lambda c.a}\,(\mathbf{tran})$$

Figure 4. β-equality with an α-conversion

The instance of $(\mathbf{perm})$ relies on the fact that $(b\ a)\cdot\lambda a.X \equiv \lambda b.(b\ a)\cdot X$. □

The following is a nominal version of the substitution lemma [Bar84, Lemma 2.1.16] in terms of β-redexes:

Lemma 2.17 $a\#Y \vdash_{\mathsf{ULAM}} (\lambda b.((\lambda a.Z)X))Y = (\lambda a.((\lambda b.Z)Y))((\lambda b.X)Y)$.

A proof by induction on Z is impossible — Z need not range over syntax, only over elements of nominal algebra models of ULAM (for the general theory of nominal algebra denotations see elsewhere [Mat07]). But ULAM proves this, in logic:

Proof By $(\mathbf{tran})$ the proof obligation follows from:

$$(\lambda b.((\lambda a.Z)X))Y = ((\lambda b.(\lambda a.Z))Y)((\lambda b.X)Y) \tag{1}$$

$$((\lambda b.(\lambda a.Z))Y)((\lambda b.X)Y) = (\lambda a.((\lambda b.Z)Y))((\lambda b.X)Y) \tag{2}$$

Part (1) follows by axiom $(\beta\mathbf{app})$; for part (2) we give the full derivation:

$$\dfrac{\dfrac{a\#Y}{(\lambda b.(\lambda a.Z))Y=\lambda a.((\lambda b.Z)Y)}\,(\mathbf{ax}_{\beta\mathbf{abs}}) \quad \dfrac{}{(\lambda b.X)Y=(\lambda b.X)Y}\,(\mathbf{refl})}{((\lambda b.(\lambda a.Z))Y)((\lambda b.X)Y)=(\lambda a.((\lambda b.Z)Y))((\lambda b.X)Y)}\,(\mathbf{cngapp})$$

□

The rules of CORE are not syntax-directed (consider $(\mathbf{tran})$), but we can derive syntactic criteria for equality in CORE, which will also be useful later:

Definition 2.18 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 $a\in\mathrm{ds}(\pi,\pi')$.

Theorem 2.19 *$\Delta\vdash_{\mathsf{CORE}} t=u$ precisely when:*

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

For proofs see elsewhere [Mat07, Cor. 2.5.4]; thus, CORE induces the same theory of equality as the rules for equality from [UPG04].

3 The lambda-calculus

We give a short formal treatment of λ-terms and $\alpha\beta$-equivalence. For a detailed discussion, see elsewhere [Bar84].

Definition 3.1 Call a term **ground** when it mentions no unknowns. [5]

As discussed in Subsection 2.1 our nominal terms syntax is specialised to the λ-calculus; ground terms g, h, k are characterised by:

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

Definition 3.2 Define the **free atoms** $fa(g)$ by:

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

Lemma 3.3 *$a \notin fa(g)$ if and only if $\vdash a\#g$.*
Also, if $a \notin \mathsf{Atms}(g)$ then $\vdash a\#g$.

Definition 3.4 Define the **size** $|g|$ of a ground term g 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 $|g|$ by:

$$a[h/a] \equiv h \qquad b[h/a] \equiv b \qquad (\lambda a.g)[h/a] \equiv \lambda a.g$$

$$(\lambda b.g)[h/a] \equiv \lambda b.(g[h/a]) \quad (b \notin fa(h))$$

$$(\lambda b.g)[h/a] \equiv \lambda c.(g[c/b][h/a]) \quad (b \in fa(h),\ c \text{ fresh})$$

$$(g'g)[h/a] \equiv (g'[h/a])(g[h/a]) \qquad \mathsf{c}[h/a] = \mathsf{c}$$

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.

Definition 3.5 Write $=_\alpha$ for the α-**equivalence** relation which is obtained by extending syntactic equivalence $\equiv$ with the following rule to rename bound variables:

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

Lemma 3.6

(i) *If $a, b \notin fa(g)$ then $(a\ b) \cdot g =_\alpha g$.*

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

[5] Ground terms should not be confused with closed lambda terms, i.e. terms without free atoms; closed terms are not used in this paper.

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 $a, b \notin \mathsf{Atms}(g')$. Equality is symmetric, so we reverse the process to return to g.

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

Theorem 3.7 *On ground terms, derivable equality in* CORE *coincides with* $=_\alpha$.

Proof See Theorem 4.3.13 of [Mat07]. □

Remark 3.8 We do not quotient terms by α-conversion and we do not use a nominal-style datatype of syntax-with-binding [GP02]. Later on the proof-method for Theorem 4.4 involves delicate accounting of what atoms appear abstracted in terms. In particular, we do not want to have to invent names (and keep track of our invented names) for abstracted atoms in Definition 4.10.

Definition 3.9 Let (one step) **β-reduction** $g \to_\beta h$ be defined by

- $(\lambda a.g)h \to_\beta g[h/a]$.
- If $g \to_\beta g'$ then $\lambda a.g \to_\beta \lambda a.g'$.
- If $g \to_\beta g'$ then $gh \to_\beta g'h$.
- If $h \to_\beta h'$ then $gh \to_\beta gh'$.

Let (one step) **β-equality** $\leftrightarrow_\beta$ be defined by $g \leftrightarrow_\beta h$ when $g \to_\beta h$ or $h \to_\beta g$.

Definition 3.10 Let **one step $\alpha\beta$-equality** $\leftrightarrow_{\alpha\beta}$ be defined such that $g \leftrightarrow_{\alpha\beta} h$ when there exist g' and h' satisfying

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

Let (multi step) **$\alpha\beta$-equality** $=_{\alpha\beta}$ be the transitive reflexive closure of $\leftrightarrow_{\alpha\beta}$.

4 Soundness, completeness and conservativity

In Section 2 we presented nominal algebra and the theory ULAM. We saw formal derivations reminiscent of the 'informal meta-level'. What exactly is ULAM an axiomatisation of? Theorems 4.3 and 4.4 are mathematical answers but perhaps a semi-formal intuition is also interesting: ULAM is an axiomatisation of the informal meta-level of λ-terms up to $\alpha\beta$-equivalence. Put another way, nominal term syntax has unknowns X which behave like meta-variables (the 't' in '$\lambda a.t$ where t is a term') and like the (unique) hole in Felleisen-style evaluation contexts [PS98, Page 15]. ULAM axiomatises equality for contexts, implemented and generalised using nominal term syntax.

Definition 4.1 Call σ a **ground substitution** for a set of unknowns $\mathcal{X}$ when $\sigma(X)$ is ground for every $X \in \mathcal{X}$. Call σ ground for Δ, t, u when σ is ground for the set of unknowns appearing anywhere in Δ, t, or u.

So: ground substitutions eliminate all metavariables.

Definition 4.2 Write $\Delta \models t = u$ when $t\sigma =_{\alpha\beta} u\sigma$ for all ground substitutions σ for Δ, t, u such that $a \notin fa(\sigma(X))$ for every $a\#X \in \Delta$.

Theorem 4.3 (Soundness) *For any Δ, t, u, if $\Delta \vdash_{\mathsf{ULAM}} t = u$ then $\Delta \models t = u$.*

Proof We proceed by induction on ULAM derivations. We sketch the proof (some reasoning on freshnesses is elided):

- The cases $(\mathbf{refl})$, $(\mathbf{symm})$, $(\mathbf{tran})$, $(\mathbf{cng}\lambda)$ and $(\mathbf{cngapp})$ follow by induction using the fact that $=_{\alpha\beta}$ is an equivalence relation and a congruence.
- The case $(\mathbf{perm})$. Suppose $a, b \notin fa(g)$. By Lemma 3.6 we obtain $(a\ b)\cdot g =_\alpha g$. Since $=_\alpha$ implies $=_{\alpha\beta}$ we conclude $(a\ b)\cdot g =_{\alpha\beta} g$.
- The case $(\mathbf{fr})$. Unknowns are irrelevant because ground terms by definition do not contain them. If $\sigma(X)$ mentions an atom which $(\mathbf{fr})$ generates fresh for some X in Δ, t, or u, then we 'freshen' the atom further to avoid an 'name clash'.[6]
- The case $(\mathbf{ax})$. The axioms of ULAM are all standard properties of the λ-calculus:
  · $(\lambda a.a)h =_{\alpha\beta} h$.
  · If $a \notin fa(g)$ then $(\lambda a.g)h =_{\alpha\beta} g$.
  · $(\lambda a.(g'g))h =_{\alpha\beta} (\lambda a.g')h)((\lambda a.g)h)$.
  · If $b \notin fa(h)$ then $(\lambda a.(\lambda b.g))h =_{\alpha\beta} \lambda b.((\lambda a.g)h)$.
  · $(\lambda a.g)a =_{\alpha\beta} g$.

$\square$

Theorem 4.4 (Completeness) *For any Δ, t, u, if $\Delta \models t = u$ then $\Delta \vdash_{\mathsf{ULAM}} t = u$.*

The proof of Theorem 4.4 occupies the rest of this section.

Definition 4.5 Fix a freshness context Δ and two terms t and u. Let $\mathcal{A}$ be the atoms mentioned anywhere in Δ, t, or u, i.e. $\mathcal{A} = \{a \mid a\#X \in \Delta\} \cup \mathsf{Atms}(t, u)$. 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.

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

Definition 4.6 Specify ς a ground substitution for $\mathcal{X}$ by:

- $\varsigma(X) \equiv c_X a_{X1} \ldots a_{Xk_X}$ when $X \in \mathcal{X}$, and
- $\varsigma(X) \equiv X$ otherwise (the choice of X in the right-hand side is irrelevant).

Lemma 4.7 *If $a\#X \in \Delta$ then $a \notin fa(\varsigma(X))$.*

By assumption $\Delta \models t = u$, so by Lemma 4.7, we know $t\varsigma =_{\alpha\beta} u\varsigma$. By definition 3.10, we also know that there exist chains of alternating α- and (one step) β-equalities.

Definition 4.8 Fix a chain

$$t\varsigma \equiv g_1 =_\alpha g_2 \leftrightarrow_\beta g_3 =_\alpha \ldots \leftrightarrow_\beta g_{m-1} =_\alpha g_m \equiv u\varsigma.$$

[6] We retain the inductive hypothesis of the 'freshened' derivation using the mathematical principle of ZFA equivariance [GM07c, Appendix A] — or by performing induction instead on the depth of derivations, and proving that freshening atoms does not affect this measure.

Without loss of generality assume the α- and β-equalities do not introduce abstractions by atoms from $\mathcal{C}$.

Furthermore, let $\mathcal{A}^+$ be the set of all atoms mentioned anywhere in the chain of Definition 4.8, and 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 4.9 Call a ground term g **accurate** when:

- it mentions only atoms in $\mathcal{A}^+$;
- if $c_X \in \mathcal{C}$ appears it is always in head position applied to a list of terms, like this: '$c_X g_1 \dots g_{k_X}$'.

The final condition implies that $c_X \in \mathcal{C}$ never occurs abstracted: no term contains 'λc_X'. Also note that $g_1, \dots, g_m$ are accurate by construction.

Definition 4.10 Define an **inverse translation** from accurate ground terms to (possibly non-ground) terms inductively by:

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

Lemma 4.11 *$\Delta^+ \vdash_{\mathsf{ULAM}} (t\varsigma)^{\text{-}1} = t$, and $\Delta^+ \vdash_{\mathsf{ULAM}} (u\varsigma)^{\text{-}1} = u$.*

Proof We prove by induction that if v is a subterm of t or u then $\Delta^+ \vdash_{\mathsf{ULAM}} (v\varsigma)^{\text{-}1} = v$. The only interesting case is when $v \equiv \pi \cdot X$: we must show that

$$\Delta^+ \vdash_{\mathsf{ULAM}} (\lambda a_{X1}. \cdots \lambda a_{Xk_X}.X)\pi(a_{X1}) \cdots \pi(a_{Xk_X}) = \pi \cdot X.$$

Take a set of fresh atoms $\mathcal{B} = \{b_{X_i} \mid X, i \text{ such that } a_{X_i} \in \mathcal{A}\}$ in bijection with $\mathcal{A}$; so $\mathcal{B}$ is disjoint from the a_{X_i} and $\pi(a_{X_i})$.

Then by (**fr**), it suffices to show

$$\Delta^+ \cup \Delta_{\mathcal{B}} \vdash_{\mathsf{ULAM}} (\lambda a_{X1}. \cdots \lambda a_{Xk_X}.X)\pi(a_{X1}) \cdots \pi(a_{Xk_X}) = \pi \cdot X,$$

where $\Delta_{\mathcal{B}} = \{b_{Xi}\#Y \mid b_{X_i} \in \mathcal{B} \text{ and } Y \in \mathcal{X}\}$.

By transitivity, it suffices to show that the following are derivable from $\Delta^+ \cup \Delta_{\mathcal{B}}$ in ULAM, taking $\pi_1 = (b_{X1}\ a_{X1}) \cdots (b_{Xk_X}\ a_{Xk_X})$ and $\pi_2 = (b_{Xk_X}\ \pi(a_{Xk_X})) \cdots (b_{X1}\ \pi(a_{X1}))$:

$$(\lambda a_{X1}. \cdots \lambda a_{Xk_X}.X)\pi(a_{X1}) \cdots \pi(a_{Xk_X}) = (\lambda b_{X1}. \cdots \lambda b_{Xk_X}.\pi_1 \cdot X)\pi(a_{X1}) \cdots \pi(a_{Xk_X}) \tag{3}$$

$$(\lambda b_{X1}. \cdots \lambda b_{Xk_X}.\pi_1 \cdot X)\pi(a_{X1}) \cdots \pi(a_{Xk_X}) = (\lambda \pi(a_{X1}). \cdots \lambda \pi(a_{Xk_X}).(\pi_2 \circ \pi_1) \cdot X)\pi(a_{X1}) \cdots \pi(a_{Xk_X}) \tag{4}$$

$$(\lambda \pi(a_{X1}). \cdots \lambda \pi(a_{Xk_X}).(\pi_2 \circ \pi_1) \cdot X)\pi(a_{X1}) \cdots \pi(a_{Xk_X}) = (\pi_2 \circ \pi_1) \cdot X \tag{5}$$

$$(\pi_2 \circ \pi_1) \cdot X = \pi \cdot X \tag{6}$$

We consider the proof obligations in turn:

- Proof obligation (3) follows by k_X applications of Lemma 2.16 since for $1 \le i \le k_X$,

$$\Delta^+ \cup \Delta_{\mathcal{B}} \vdash b_{Xi}\#\lambda a_{Xi+1}. \cdots \lambda a_{Xk_X}.(b_{Xi-1}\ a_{Xi-1}) \cdots (b_{X1}\ a_{X1}) \cdot X.$$

- Proof obligation (4) follows by k_X applications of Lemma 2.16 when for $1 \le i \le k_X$,

$$\Delta^+ \cup \Delta_{\mathcal{B}} \vdash \pi(a_{Xi})\#\lambda b_{Xi+1}. \cdots \lambda b_{Xk_X}.((\pi(a_{Xi-1})\ b_{Xi-1}) \cdots (\pi(a_{X1})\ b_{X1}) \circ \pi_1) \cdot X.$$

By the rules for freshness, this follows from $\pi(a_{Xi})\#\pi_1 \cdot X \in \Delta^+ \cup \Delta_{\mathcal{B}}$ since the $\pi(a_{Xj})$ are all disjoint and the b_{Xj} are different from $\pi(a_{Xi})$. We reason by cases on $\pi(a_{Xi})$:
· $\pi(a_{Xi}) \neq a_{Xj}$ for all j: then $\pi(a_{Xi})\#X \in \Delta$ by Definition 4.5.
· $\pi(a_{Xi}) = a_{Xj}$ for some j: then $b_{Xj}\#X \in \Delta_{\mathcal{B}}$.

- Proof obligation (5) follows by k_X instances of axiom (β**id**).
- In order to prove (6), it is convenient to show the stronger property

$$\Delta^+ \cup \Delta_{\mathcal{B}} \vdash_{\mathsf{CORE}} (\pi_2 \circ \pi_1) \cdot X = \pi \cdot X.$$

By Theorem 2.19 we need only show that $\Delta^+ \cup \Delta_{\mathcal{B}} \vdash \mathrm{ds}(\pi_2 \circ \pi_1, \pi)\#X$. That is, we must show that $a\#X \in \Delta^+ \cup \Delta_{\mathcal{B}}$ for every a such that $(\pi_2 \circ \pi_1)(a) \neq \pi(a)$, which follows by a case distinction on a (considering every $a \in \mathsf{supp}(\pi_2 \circ \pi_1) \cup \mathsf{supp}(\pi)$) using Definitions 4.5 and 4.8.

□

We need some technical lemmas:

Lemma 4.12 *Suppose that g is accurate. For any $a \in \mathcal{A}^+$, if $a \notin fa(g)$ then $\Delta^+ \vdash a\#g^{\text{-}1}$.*

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

$$\Delta^+ \vdash a\#\lambda a_{X1}.\cdots\lambda a_{Xk_X}.X,$$

which follows using the rules for freshness and the fact that the atoms that might not be fresh for X are precisely the a_{Xi}. □

Lemma 4.13 *Suppose that g is accurate. Suppose that π is a permutation such that $\pi(a) = a$ for all $a \notin \mathcal{A}^+ \setminus \mathcal{C}$. Then $\Delta^+ \vdash_{\mathsf{CORE}} (\pi \cdot g)^{\text{-}1} = \pi \cdot (g^{\text{-}1})$.*

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 $a \in \mathcal{C}$. □

Lemma 4.14 *Suppose that g and h are accurate. Then if $g =_\alpha h$ then $\Delta^+ \vdash_{\mathsf{CORE}} g^{\text{-}1} = h^{\text{-}1}$.*

Proof By Theorem 3.7 $g =_\alpha h$ coincides with $\vdash_{\mathsf{CORE}} g = h$. The proof is then by a detailed but routine induction on g using the syntactic criteria of Theorem 2.19.

The only non-trivial case is when

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

By assumption $a, b \in \mathcal{A}^+ \setminus \mathcal{C}$. Then by Lemma 4.12 we have $\Delta^+ \vdash b\#(g')^{\text{-}1}$. By inductive hypothesis we have $\Delta^+ \vdash_{\mathsf{CORE}} ((b\ a) \cdot g')^{\text{-}1} = (h')^{\text{-}1}$, and by Lemma 4.13 we obtain $\Delta^+ \vdash_{\mathsf{CORE}} ((b\ a) \cdot g')^{\text{-}1} = (b\ a) \cdot (g')^{\text{-}1}$. Then from the rules for freshness and equality it follows that $\Delta^+ \vdash_{\mathsf{CORE}} \lambda a.(g')^{\text{-}1} = \lambda b.(h')^{\text{-}1}$. □

Lemma 4.15 *Suppose that $a \in \mathcal{A}^+ \setminus \mathcal{C}$. Suppose that g, h, and $g[h/a]$ are accurate. Then $\Delta^+ \vdash_{\mathsf{ULAM}} (\lambda a.(g^{\text{-}1}))(h^{\text{-}1}) = (g[h/a])^{\text{-}1}$.*

Proof By induction on $|g|$, the size of g. We consider a selection of cases:

- $a[h/a]$. $\Delta^+ \vdash_{\mathsf{ULAM}} (\lambda a.a)(h^{\text{-}1}) = h^{\text{-}1}$ by (β**var**).
- $b[h/a]$. $\Delta^+ \vdash_{\mathsf{ULAM}} (\lambda a.b)(h^{\text{-}1}) = b$ by ($\beta\#$) since $\Delta^+ \vdash a\#b$.

- $(\lambda b.g)[h/a]$ where $b \notin fa(h)$. By Lemma 4.12 $\Delta^+ \vdash b\#h^{\text{-}1}$. The result follows using $(\beta\mathbf{abs})$.
- $(\lambda b.g)[h/a]$ where $b \in fa(h)$. By assumption $\lambda b.g$ is accurate, therefore $b \notin \mathcal{C}$. By Definition 3.4 $(\lambda b.g)[h/a] \equiv \lambda c.(g[c/b][h/a])$ for fresh c (so $c \notin fa(g)$ and $c \notin fa(h)$). By assumption $\lambda c.(g[c/b][h/a])$ is accurate so $c \notin \mathcal{C}$ and $g[c/b][h/a]$ is accurate. We must show
$$\Delta^+ \vdash_{\text{ULAM}} (\lambda a.(\lambda b.(g^{\text{-}1})))(h^{\text{-}1}) = \lambda c.((g[c/b][h/a])^{\text{-}1}).$$
Note that by Lemma 4.12, $\Delta^+ \vdash c\#g^{\text{-}1}$ and $\Delta^+ \vdash c\#h^{\text{-}1}$, and therefore $\Delta^+ \vdash c\#\lambda b.(g^{\text{-}1})$ by $(\#\lambda\mathbf{b})$. Also $\Delta^+ \vdash b\#\lambda b.(g^{\text{-}1})$ is immediate by $(\#\lambda\mathbf{a})$. We present the rest of the proof in a calculational style:
$$\begin{array}{ll} & \lambda c.((g[c/b][h/a])^{\text{-}1}) \\ = & \{\, g[c/b] =_\alpha (c\,b)\cdot g \text{ by Lemma 3.6 since } c \notin fa(g) \,\} \\ & \lambda c.(((c\,b)\cdot g)[h/a])^{\text{-}1} \\ = & \{\, \text{inductive hypothesis, since } (c\,b)\cdot g \text{ is accurate} \,\} \\ & \lambda c.((\lambda a.((c\,b)\cdot g)^{\text{-}1})(h^{\text{-}1})) \\ = & \{\, \text{Lemma 4.13} \,\} \\ & \lambda c.(((\lambda a.(c\,b)\cdot(g^{\text{-}1})))(h^{\text{-}1})) \\ = & \{\, (\beta\mathbf{abs}), \text{ since } \Delta^+ \vdash c\#h^{\text{-}1} \,\} \\ & (\lambda a.(\lambda c.((c\,b)\cdot(g^{\text{-}1}))))(h^{\text{-}1}) \\ = & \{\, (\mathbf{perm}) \text{ since } \Delta^+ \vdash b\#\lambda b.(g^{\text{-}1}) \text{ and } \Delta^+ \vdash c\#\lambda b.(g^{\text{-}1}) \,\} \\ & (\lambda a.(\lambda b.(g^{\text{-}1})))(h^{\text{-}1}) \end{array}$$
The result follows by transitivity.
- $c_X[h/a]$ where $c_X \in \mathcal{C}$. By assumption $a \neq c_X$, so we must show
$$\Delta^+ \vdash_{\text{ULAM}} (\lambda a.(\lambda a_{X1}.\cdots\lambda a_{Xk_X}.X))(h^{\text{-}1}) = \lambda a_{X1}.\cdots\lambda a_{Xk_X}.X.$$
By axiom $(\beta\#)$ this is when $\Delta^+ \vdash a\#\lambda a_{X1}.\cdots\lambda a_{Xk_X}.X$. Since $a \notin fa(c_X)$, this follows from Lemma 4.12.

The result follows. □

Corollary 4.16 *Suppose that g and h are accurate. If $g \leftrightarrow_\beta h$ then $\Delta^+ \vdash_{\text{ULAM}} g^{\text{-}1} = h^{\text{-}1}$.*

Proof By induction on the rules for $\rightarrow_\beta$ from Definition 3.9. It suffices to show the following (here g, g', h, h', and $g[h/a]$ are accurate and $a \in \mathcal{A}^+ \setminus \mathcal{C}$):

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

The first part is Lemma 4.15. The other parts follow by $(\mathbf{cng}\lambda)$ and $(\mathbf{cngapp})$. □

We are now ready to prove Theorem 4.4:

Proof (of Theorem 4.4) Recall from Definition 4.8 the chain

$$t\varsigma \equiv g_1 =_\alpha g_2 \leftrightarrow_\beta g_3 =_\alpha \ldots \leftrightarrow_\beta g_{m-1} =_\alpha g_m \equiv u\varsigma.$$

By Lemma 4.14 and Corollary 4.16

$$\Delta^+ \vdash_{\mathsf{ULAM}} (t\varsigma)^{\text{-1}} \equiv g_1^{\text{-1}} = g_2^{\text{-1}} = \ldots = g_m^{\text{-1}} \equiv (u\varsigma)^{\text{-1}},$$

so $\Delta^+ \vdash_{\mathsf{ULAM}} (t\varsigma)^{\text{-1}} = (u\varsigma)^{\text{-1}}$ by transitivity. By Lemma 4.11 then also $\Delta^+ \vdash_{\mathsf{ULAM}} t = u$. Since Δ^+ extends Δ with atoms that are not mentioned in t and u we extend the derivation with (**fr**) to obtain $\Delta \vdash_{\mathsf{ULAM}} t = u$ as required. $\square$

Corollary 4.17 *For any ground terms g, h, $\vdash_{\mathsf{ULAM}} g = h$ if and only if $g =_{\alpha\beta} h$.*

Proof By Theorems 4.3 and 4.4, using the fact that g and h are ground. $\square$

With the results we proved up to now, it is easy to prove conservativity of ULAM over CORE; the proof exploits the substitution ς from Definition 4.6, and confluence of the λ-calculus.

Definition 4.18 Call g a **β-normal form** when no g' exists with $g \to_\beta g'$.

Lemma 4.19 *Fix Δ. Suppose that t and u contain no subterm of the form $(\lambda a.v)w$. Then for ς the ground substitution from Definition 4.6, $t\varsigma$ 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, u. Applying this substitution to t and u cannot introduce subterms of the form $(\lambda a.v)w$. $\square$

Theorem 4.20 (Conservativity) *Suppose that t and u contain no subterm of the form $(\lambda a.v)w$. 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 Definition 4.6. By Theorem 4.3, $t\varsigma =_{\alpha\beta} u\varsigma$. By Lemma 4.19 we know that $t\varsigma$ and $u\varsigma$ are β-normal forms. By confluence of the λ-calculus [Bar84, Theorem 3.2.8] $t\varsigma =_\alpha u\varsigma$.

We now prove $\Delta \vdash_{\mathsf{CORE}} t = u$ by induction on t. The calculations are detailed but entirely routine. We consider the hardest case: $t \equiv \pi \cdot X$. Then $t\varsigma \equiv c_X \pi(a_{X1}) \ldots \pi(a_{Xk_X})$. By Theorem 2.19, 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. $\square$

5 Conclusions

What are functions, from a logical point of view?

Curry discovered *combinatory algebra* [CF58]. The signature contains a binary term-former *application* and two constants **S** and **K**. Axioms are $\mathbf{K}xy = x$ and $\mathbf{S}xyz = (xz)(yz)$.

This syntax is parsimonious and the axioms are compact, but it is not natural or ergonomic to program in. There is also a mathematical issue: the natural encoding of closed λ-terms into combinatory algebra syntax ([Bar84, Section 7] or [Sel02, Subsection 1.4]) is unsound; it does not map $\alpha\beta$-equivalent λ-terms to provably equal terms in combinatory algebra. We can strengthen combinatory algebra to *lambda algebra* by adding five more axioms [Sel02, Proposition 5] but the translation is still not sound; 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 $\lambda x.N$. For soundness we need the Meyer-Scott axiom [Sel02, Proposition 20] (Selinger calls it 'the notorious rule'). In short, combinators do not capture the same functions as expressed by the λ-calculus.

'Lambda-lifting' introduces constant symbols to represent functions and adds axioms for them [Joh85]. This expresses functions, but the axiomatisation is of atomic constant symbols representing individual functions (as many as we would like to add) and not of the λ-calculus. The issues of variables and binding surrounding the 'λ' in the 'λ-calculus' are avoided, or at least, relegated to the meta-level (into the universal quantifiers used in the formula expressing the properties of each atomic constant symbol). Implementationally this can be extremely convenient but for logicians we should consider this deeply unsatisfactory.

Salibra's Lambda Abstraction Algebra [Sal00] uses a signature with a term-former 'λa' for every a. Freshness side-conditions appear 'hard-wired' in the structure of terms in axioms. For example Salibra's rule (β_4) from [Sal00, page 6] $(\lambda x.(\lambda x.\xi))\mu = \lambda x.\xi$ is a version of $(\beta\#)$ where the freshness condition is built into the structure of the term by writing $\lambda x.\xi$. This is essentially a λ-calculus version of cylindric algebra [HMT85].

The definition of $\alpha\beta$-equivalence on syntax is occasionally called 'axiomatising the λ-calculus', although it is just an equivalence relation on abstract syntax trees. However, if the λ-calculus syntax serves as the language of a logic with an equality judgement then $\alpha\beta$-equivalence may have the status of axioms. For example Andrews's logic $\mathcal{Q}_0$ [And86, §51] contains five axioms $(\mathbf{4_1})$, $(\mathbf{4_2})$, $(\mathbf{4_3})$, $(\mathbf{4_4})$, and $(\mathbf{4_5})$ ([And86, page 164]). In fact these are axiom *schemes*, containing meta-variables $\mathbf{A}$ and $\mathbf{B}$ in the informal meta-level ranging over terms, and meta-variables x, y ranging permutatively over variable-symbols. A kinship with Figure 1 is apparent, but here, axioms feature in the formal framework of nominal algebra — a formal logic, not an informal meta-level.

Nominal techniques.

A rewrite system for the λ-calculus appeared already in [FG07] but without any statement or proof of completeness (indeed, it was not complete). More recently the authors have used nominal algebra to axiomatise first-order logic as a theory FOL [GM07c] and substitution as a theory SUB [GM08].

One might suspect that the theory of substitution should be only a hair's breadth away from that of the untyped λ-calculus, and could be obtained by imposing a type system (just as the simply-typed λ-calculus is related to the untyped λ-calculus).[7] Perhaps this will indeed prove to be the case, but so far all attempts by the authors to derive the properties of SUB from those of ULAM have tantalisingly failed. This may (or may not!) indicate the existence of a subtle mathematical point lurking in these systems; if there is, it seems to relate to the difference between λ-calculus variables and nominal algebra unknowns.

[7] We briefly give some intuition for what SUB is: $(\lambda b.(ba))(\lambda a.a) = a$ is derivable in ULAM, but only $\mathsf{app}(b,a)[b \mapsto \mathsf{lam}([a]a)] = \mathsf{app}(\mathsf{lam}([a]a), a)$ is derivable in SUB (notation from [GM08]).

Nominal algebra has a cousin, nominal equational logic (**NEL**) [CP07]. This was derived from nominal algebra, making some slightly different design decisions. NEL satisfies a completeness result for a class of models in nominal sets [CP07], as does nominal algebra [GM07a,Mat07] (in other words — derivable equality coincides with validity *in all models*). The completeness result in this paper is stronger: it is valid for a single elementary model — derivable equality coincides with validity *in one particular model*, which we have built in this paper. Similarly for the authors' treatments of substitution [GM08] and first-order logic [GM07c]. 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.

Future work and conclusions.

ULAM completes a trio of papers studying (untyped) nominal algebra considered as a logical framework: first-order logic [GM07c], substitution [GM08], and with this paper, the λ-calculus. Between them, these works cover a significant fraction of the typical syntaxes of interest in theory and practice in computer science. It would of course be interesting to seek common generalisations of the proof-techniques therein.

Our most immediate interest is in constructing a theorem-prover based on nominal algebra instead of the λ-calculus. We expect this to formally represent at least some kinds of reasoning better than the λ-calculus can, because the treatment of names and variables in nominal algebra, and nominal techniques in general, is very close to some kinds of informal practice (for example the pervasive use of meta-variables and freshness conditions, as appear in specifications of ... the λ-calculus). From that perspective this paper proves a vital correctness result.

It remains to understand the connections between standard models for the λ-calculus and the class of models determined by models of ULAM in nominal sets. It might also be interesting to construct versions of graph models or domain models of the λ-calculus [Bar84,Sto77] that themselves are built in nominal sets; does the presence of nominal-style atoms add any useful structure? Similarly, we can consider existing work using the language of categories (for example [Sel02,AB07]) using categories based on nominal sets.

A representation theorem for ULAM, in nominal sets models, would also be interesting. As a first step, in recent work we have proved an HSP theorem (also known as Birkhoff's theorem) for nominal algebra [GM07b].

References

[AB07] Giulio Manzonetto Antonio Bucciarelli, Thomas Ehrhard. Not enough points is enough. In *Computer Science Logic*, pages 298–312, 2007.

[ABI$^+$96] 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.

[AGM$^+$04] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *LICS '04: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS'04)*, pages 150–159. IEEE, 2004.

[And86] Peter B. Andrews. *An introduction to mathematical logic and type theory: to truth through proof.* Academic Press, 1986.

[Bar77] Jon Barwise. An introduction to first-order logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 5–46. North-Holland, 1977.

[Bar84] H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics (revised ed.).* North-Holland, 1984.

[BL05] Nick Benton and Benjamin Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. of the 7th Int'l Conf. on Typed Lambda Calculi and Applications (TLCA)*, volume 3461 of *LNCS*, pages 86–101, 2005.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[Bru96] N. Brunner. 75 years of independence proofs by Fraenkel-Mostowski permutation models. *Mathematica Japonica*, 43(1):177–199, 1996.

[CC04] Luís Caires and Luca Cardelli. A spatial logic for concurrency (part II). *Theoretical Computer Science*, 322(3):517–565, 2004.

[CF58] Haskell B. Curry and Robert Feys. *Combinatory Logic*, volume 1. North-Holland, 1958.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, pages 56–68, 1940.

[CP07] Ranald A. Clouston and Andrew M. Pitts. Nominal equational logic. *ENTCS*, 172:223–257, 2007.

[dB72] 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.

[FG06] Maribel Fernández and Murdoch J. Gabbay. Curry-style types for nominal rewriting. TYPES'06, 2006.

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

[Gab00] Murdoch J. Gabbay. *A Theory of Inductive Definitions with alpha-Equivalence*. PhD thesis, Cambridge, UK, 2000.

[GM07a] 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 *LNCS*, pages 162–176. Springer, 2007.

[GM07b] Murdoch J. Gabbay and Aad Mathijssen. Nominal algebra and the HSP theorem (technical report). Technical Report HW-MACS-TR-0057, Heriot-Watt, 2007.

[GM07c] Murdoch J. Gabbay and Aad Mathijssen. One-and-a-halfth-order logic. *Journal of Logic and Computation*, 2007. Available online.

[GM08] Murdoch J. Gabbay and Aad Mathijssen. Capture-avoiding substitution as a nominal algebra. *Formal Aspects of Computing*, 2008. Available online.

[GP02] 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, 2002.

[HMT85] L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras*. North-Holland, 1971 and 1985. Parts I and II.

[Joh85] Thomas Johnsson. Lambda lifting: transforming programs to recursive equations. In *Conference on Functional programming languages and computer architecture*, pages 190–203. Springer, 1985.

[Lei94] 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.

[Mat07] Aad Mathijssen. *Logical Calculi for Reasoning with Binding*. PhD thesis, Technische Universiteit Eindhoven, 2007.

[Pau89] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.

[Pau96] Lawrence C. Paulson. *ML for the working programmer (2nd ed.)*. Cambridge University Press, 1996.

[PE88] Frank Pfenning and Conal Elliot. Higher-order abstract syntax. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 199–208. ACM Press, 1988.

[PS98] A.M. Pitts and I.D.B. Stark. Operational reasoning for functions with local state. In A. D. Gordon and A.M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 227–273. Cambridge University Press, 1998.

[Sal00] Antonino Salibra. On the algebraic models of lambda calculus. *Theoretical Computer Science*, 249(1):197–240, 2000.

[Sel02] Peter Selinger. The lambda calculus is algebraic. *Journal of Functional Programming*, 12(6):549–566, 2002.

[Sto77] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[Tho96] Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison Wesley, 1996.

[Tze07] Nikos Tzevelekos. Full abstraction for nominal general references. In *LICS*, pages 399–410. IEEE, 2007.

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