A Nominal Axiomatization of the Lambda Calculus

MURDOCH J. GABBAY,1 School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh EH14 4AS, Scotland, Great Britain.

AAD MATHIJSSEN, Department of Mathematics and Computer Science, Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands.
E-mail: a.h.j.mathijssen@tue.nl

Abstract

The lambda calculus is fundamental in computer science. It resists an algebraic treatment because of capture-avoidance side-conditions. Nominal algebra is a logic of equality designed for specifications involving binding. We axiomatize the lambda calculus using nominal algebra, demonstrate how proofs with these axioms reflect the informal arguments on syntax and we prove the axioms to be sound and complete. We consider both non-extensional and extensional versions (alpha-beta and alpha-beta-eta equivalence). This connects the nominal approach to names and binding with the view of variables as a syntactic convenience for describing functions. The axiomatization is finite, close to informal practice and it fits into a context of other research such as nominal rewriting and nominal sets.

Keywords: Lambda calculus, equational logic, nominal techniques.

1 Introduction

Functions are widely used in what we now call computer science; a development that can be traced back to Church [15]. They are the basis of functional programming languages [42, 52]; they also find application in logic [8, 36], theorem provers [4, 43], rewriting [6] and more. Functions are a basic mathematical entity of computer science.

Functions have been studied in different ways. Models include Scott domain models, graph and filter models, game models and more [10, 11, 18, 35]. There is much study carried out using rewriting, starting with the proof of confluence of β-reduction (a particularly neat proof is in [50]). There are also axiomatizations, including axioms by Andrews [3], λ-algebras [48], Salibra’s lambda-abstraction algebras [45] and (if one does not care about representing λ-abstraction, which we do) λ-lifting [34].

This article will axiomatize the λ-calculus using a recent logic by the authors, nominal algebra [27, 28, 30, 39]. Nominal algebra extends universal algebra [17] with support for names and binding, while preserving much of its flavour and its good mathematical properties. Using nominal algebra to axiomatize the λ-calculus brings several benefits:

• Nominal algebra seems quite good at bringing us close to informal practice. It is easy to write down plausible axioms for the λ-calculus; they look just like well-known informal αβ(η)-equivalences. In this article, we shall consider two nominal algebra theories—
502 Nominal Axiomatization of the Lambda Calculus

(\beta_{\text{var}}) \vdash (\lambda a.a)X = X
(\beta_{\#}) a\#Z \vdash (\lambda a.Z)X = Z
(\beta_{\text{app}}) \vdash (\lambda a.(Z')X)((\lambda a.Z)X)
(\beta_{\text{abs}}) b\#X \vdash (\lambda a.(\lambda b.Z))X = \lambda b.((\lambda a.Z)X)
(\beta_{\text{id}}) \vdash (\lambda a.Z)a = Z

Figure 1. Axioms of ULAM.

• ULAM (Figure 1): this axiomatizes the \(\lambda\)-calculus up to \(\alpha\beta\)-equivalence (Theorems 4.3 and 4.7); and
• ULAME (Figure 5): this axiomatizes the \(\lambda\)-calculus up to \(\alpha\beta\eta\)-equivalence (Theorems 5.3 and 5.4).

The sets of axioms ULAM and ULAME are finite. Figures 1 and 5 do not describe infinite schemes of axioms (unlike is the case for example in [45] or [3]; see Remark 2.13). This is because name binding is handled by nominal algebra itself.2

Nominal algebra is not a stand-alone logic; it is part of a body of research that includes nominal unification and nominal rewriting [22, 54] (both of which have good computational properties)—and nominal sets.

ULAM and ULAME are nominal algebra theories, and so they take models in nominal sets [28, 32].3

The theory of these nominal sets models is investigated in [26]. They have properties that set models of universal algebra theories do not have. The notion of variety is richer, and nominal sets models are inherently finite-dimensional and exclude infinite-dimensional models.4 Thus, using nominal techniques give us the option of working purely (nominal) algebraically in a finite-dimensional world, which is an option that universal algebra does not offer.

So we can write down axioms that look plausibly like they axiomatize functional abstraction, in an algebraic logic which looks plausibly like universal algebra and which is compatible with a broader body of research. But, in what sense are the axioms correct? We can ask three questions:

• If we quotient \(\lambda\)-terms by \(\alpha\beta\)-equivalence (respectively, \(\alpha\beta\eta\)-equivalence) do we obtain, in some natural way, a model of ULAM (respectively, ULAME)?
• 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 \(\alpha\beta\)?) Similarly for ULAME.
• Does nominal algebra reasoning capture a useful fragment of the kind of reasoning steps we would like to represent?

In this article, we explore to what extent our two theories ULAM and ULAME capture ‘the \(\lambda\)-calculus and its theory’. We will demonstrate that ULAM and ULAME are the theories for the untyped \(\lambda\)-calculus, in the sense that they are sound and complete for \(\lambda\)-terms quotiented by

---

2See [41, Section 7] for an outline of how a similar idea can be implemented in a completely different kind of language.

3For the impatient, a nominal set is a set with a name-permutation action. Name binding is formed by a construction that is very similar to taking \(\alpha\)-equivalence classes; we use the permutation action to ‘rename’ the name to be bound and taking an equivalence class of all the renamed variants.

Nominal sets can be considered as sheaves, or as algebraic structures. They were introduced in [32]; see that paper for full details.

4This is terminology from cylindric techniques (see [33, Definition 7] or [45]). Nominal sets terminology calls this as finitely supported and infinitely supported, respectively.
$\alpha\beta$-equivalence and $\alpha\beta\eta$-equivalence, respectively. We will demonstrate with examples as to how they can express informal reasoning as formal derivations.

*Nota bene*: Nominal techniques were first applied to construct datatypes of syntax-with-binding [32] with good inductive reasoning principles. One datatype often used is $\lambda$-term (up to $\alpha$-equivalence). This article is not another such study, like those in nominal sets [32], higher order abstract syntax [44], de Bruijn terms [20] and so on—which are about collections of syntax trees.

**Overview of the article**: We introduce nominal algebra in Section 2, with a syntax specialized to our application to the $\lambda$-calculus (a general treatment is provided in [39]). In Section 3, we provide a brief formal treatment of the $\lambda$-calculus. In Section 4, we show that the nominal algebra theory $ULAM$ is sound and complete with respect to $\alpha\beta$-equality. In Section 5, we show that the nominal algebra theory $ULAME$ is sound and complete with respect to $\alpha\beta\eta$-equality. In the conclusions (Section 6), we discuss related and future work. In Appendix 1 we include, for the reader’s convenience, statements and proofs of some underlying properties of nominal algebra; these do not depend on $ULAM$ or $ULAME$ but they are relevant background, in much the same way as the definitions and results in Section 3 are relevant background.

## 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 $\lambda$-calculus application; general treatments are elsewhere [29, 39].

**Definition 2.1** Fix the following disjoint sets:

- A countably infinite set of **atoms** $A$. Atoms represent object-level variables. $a, b, c, \ldots$ will range over atoms. We use a **permutative convention** that $a, b, c, \ldots$ range over distinct atoms. Thus for example in $(#ab)$ and $(#\lambda b)$ from Figure 2, and in $(\text{perm})$ from Figure 3, $a$ and $b$ represent two distinct atoms.\(^5\)
- A countably infinite collection of **unknowns**. Unknowns represent meta-variables (as in ‘take a term $t’’; $t$ is a meta-variable ranging over terms). $X, Y, Z, \ldots$ will range over distinct unknowns.
- A possibly infinite collection of **constant symbols**. $c$ will range over constant symbols (we will never need to consider more than one at a time).

We set about constructing the machinery of nominal algebra.

\(^5\)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 over the symbols which they actually commit to the page. What matters is that the two variable symbols are different.
A permutation $\pi$ of atoms is a bijection on atoms with finite support, which means that the set $\text{supp}(\pi)$, defined by $\{a | \pi(a) \neq a\}$, is finite. In words: for ‘most’ atoms $\pi$ is the identity.

The following notation will prove convenient:

- Write $\text{id}$ for the identity permutation on atoms, $\pi^{-1}$ for the inverse of $\pi$, and $\pi \circ \pi'$ for the composition of $\pi$ and $\pi'$, i.e. $(\pi \circ \pi')(a) = \pi(\pi'(a))$.
- 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 $(a \ b)(b \ c)$.

**Definition 2.3**

*Terms* $t, u, v$ are inductively defined by:

$$t ::= a | \pi \cdot X | \lambda a. t | \text{tt} | \text{c}$$

We will use the following conventions:

- Application is left associative, so for example ‘$tuv$’ means ‘$(tu)v$’.
- Abstraction extends as far to the right as possible, so for example ‘$\lambda a.tu$’ means ‘$\lambda a.(tu)$’.
- We may write $\text{id} \cdot X$ just as $X$. However, note that ‘$X$’ is an unknown and not a term—‘$\text{id} \cdot X$’ is a term.
- We write $\equiv$ for syntactic identity. That is, ‘$t \equiv u$’ means ‘$t$ and $u$ denote the same term’.

A typed syntax is possible [21]. Types would cause no essential difficulties for the results to follow.

We now give some basic definitions of the atoms-permutation action and of the capturing substitution action, which are characteristic of nominal terms [54].

**Definition 2.4**

Define the set $\text{atoms}(t)$ of atoms that occur anywhere in $t$ inductively by:

$$\text{atoms}(a) = \{a\} \quad \text{atoms}(\pi \cdot X) = \text{supp}(\pi) \quad \text{atoms}(\lambda a.t) = \text{atoms}(t) \cup \{a\}$$

$$\text{atoms}(t' t) = \text{atoms}(t') \cup \text{atoms}(t) \quad \text{atoms(c)} = \emptyset$$

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

For example,

$$\text{atoms}(\lambda a.a) = \{a\} \quad \text{atoms}((b \ a) \cdot X) = \{b, a\} \quad \text{atoms(t(a,a)))} = \{a\}.$$

**Definition 2.5**

Define a permutation action $\pi \cdot t$ by:

$$\pi \cdot a \equiv \pi(a) \quad \pi \cdot (\pi' \cdot X) \equiv (\pi \circ \pi') \cdot X \quad \pi \cdot \lambda a.t \equiv \lambda(\pi(a)).(\pi \cdot t)$$

$$\pi \cdot (t' t) \equiv (\pi \cdot t')(\pi \cdot t) \quad \pi \cdot c \equiv c$$

In the clause for $\lambda$. $\pi$ acts also on the ‘$a$’. For example $(a \ b) \cdot \lambda a.X \equiv \lambda b.(a \ b) \cdot X$. In the clause for $\pi' \cdot X$, $\pi' \cdot X$ is a term (recall that ‘$X$’, on its own, is not a term; it must always be paired with a permutation, even if it is $\text{id}$).
Nominal Axiomatization of the Lambda Calculus

Figure 2. Freshness derivation rules for nominal terms.

Definition 2.6
A substitution $\sigma$ is a function from unknowns to terms.

Definition 2.7
Define a substitution action $t\sigma$ by:

$$
\begin{align*}
\frac{}{a \# b} & \quad \frac{\pi^{-1}(a) \# X}{a \# \pi \cdot X} \\
\frac{a \# \lambda a.t}{a \# \lambda . a.t} & \quad \frac{a \# t}{a \# \lambda . b.t} \\
\frac{a \# t' \# t}{a \# t'} & \quad \frac{\# t}{a \# \pi}
\end{align*}
$$

Note that substitution does not avoid capture; note also that when $\sigma$ encounters $\pi \cdot X$, the permutation $\pi$ is applied to $\sigma(X)$. For example, if $\sigma(X) \equiv a$ then:

$$
\begin{align*}
(\lambda a.X)\sigma & \equiv \lambda a.(X\sigma) \\
& \equiv \lambda a.a \\
(\lambda b.(a \cdot b) \cdot X)\sigma & \equiv \lambda b.((a \cdot b) \cdot X)\sigma \\
& \equiv \lambda b.(a \cdot b).a \\
& \equiv \lambda b.b
\end{align*}
$$

2.2 Freshness, equality, axioms and theories

Definition 2.8
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 $\Delta$ and $\nabla$ for (finite, and possibly empty) sets of primitive freshenesses and call them freshness contexts.

We may drop set brackets in freshness contexts. For example, we may write $a \# X, b \# Y$ for $\{a \# X, b \# Y\}$.

Definition 2.9
Define derivability on freshenesses by the rules in Figure 2. In this figure, $a$ and $b$ permutatively range over atoms, $t$ and $t'$ range over nominal terms, $\pi$ 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 $\Delta$. 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.

Remark 2.10
Freshness has an accepted technical denotation as a notion in nominal sets, introduced in [32], but it has a broader pedigree and it might be convenient to briefly mention it here. Freshness is a notion of independence. Notions of dependence and independence have been studied abstractly before; two different examples are in [17, Chapter VII, Section 2] or [23]. Freshness is also related with specific
Nominal Axiomatization of the Lambda Calculus

\[
\begin{align*}
\frac{}{(\text{refl})} & \quad t = t & t = u \quad (\text{symm}) & \quad u = t \\
\frac{}{(\text{tran})} & \quad t = v & \quad a \# t \quad b \# t & \quad (a \cdot b) \cdot t = t
\end{align*}
\]

\[
\begin{align*}
\frac{}{(\text{cng})} & \quad \lambda a. t = \lambda a. u \\
\frac{}{(\text{cngapp})} & \quad t' = u' \quad t = u \\
\frac{}{(\text{fr})} & \quad (a \not\in \text{atoms}(t, u))
\end{align*}
\]

\[
\begin{align*}
\frac{}{(\text{ax})} & \quad \Delta \vdash t = u \\
\frac{}{(\text{perm})} & \quad \pi \cdot (t \sigma) = \pi \cdot (u \sigma)
\end{align*}
\]

\[
\frac{}{(\text{fr})} & \quad (a \# X) \quad \pi \cdot (t \sigma) = \pi \cdot (u \sigma)
\]

\[\{a \# X \mid a \# X \in \Delta\}
\]

\[
\begin{align*}
\frac{}{(\text{fr})} & \quad (a \# X) \quad \Delta \vdash t = u
\end{align*}
\]

\[
\begin{align*}
\frac{}{(\text{fr})} & \quad (a \not\in \text{atoms}(t, u))
\end{align*}
\]

Figure 3. Derivation rules for nominal equality.

constructions studied for example by Salibra (consider, for example, the function $\Delta_1$ in Definition 3 of [45]).

Definition 2.11
An equality is a pair $t = u$. An axiom is a pair $\Delta \vdash t = u$ of a freshness context $\Delta$ 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 considered in this article are:

- **CORE**: the empty set of axioms.
- **ULAM**: the axioms from Figure 1. In Section 4, we give a formal sense in which this is a nominal algebra theory of the $\lambda$-calculus (with $\alpha\beta$).
  
  In Figure 1, note that the $a$ and $b$ are specific atoms and the $X$, $Z$ and $Z'$ are specific unknowns.
- **ULAME**: the axioms from Figure 1, plus an extra axiom $a \# Z \vdash \lambda a.(Za) = Z$.
  
  This is, of course, an $\eta$-equality. In Section 5, we give a formal sense in which this is a nominal algebra theory of the $\lambda$-calculus with $\alpha\beta\eta$.

Definition 2.12
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, $\Delta$ over freshness contexts, $\pi$ over permutations and $\sigma$ over substitutions.

Write $\Delta \vdash t = u$ when a derivation of $t = u$ exists using these rules such that:

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

We write $\emptyset \vdash t = u$ as $\vdash t = u$.

Remark 2.13
We discuss the most interesting rules of Figure 3.

- $(\text{ax})\Delta \vdash t = u$: this axiom rule expresses how we obtain instances of axioms: we instantiate unknowns by terms using substitutions (using the substitution action defined in Definition 2.7)
and also we rename atoms using permutations (using the permutation action defined in Definition 2.5).

This allows ULAM to have finitely many axioms; Figure 1 does not describe axiom-schemes. For example consider the axiom \((\beta\text{var})\!\!\!\!\!\!(\lambda a. a)X = X\). This uses an arbitrary, but fixed atom \(a\) and an arbitrary, but fixed unknown \(X\). We deduce \(\vdash_{ULAM} (\lambda b. b)X = X\) using \((\text{ax}_{\beta\text{var}})\) taking \(\pi = (b\ a)\) and \(\sigma\) mapping \(X\) to \((b\ a)\cdot X\) (and all other \(Y\) to \(id\cdot Y\)) as follows:

\[
\frac{(a\ a)X = X}{(\lambda b. b)(b\ a)\cdot ((b\ a)\cdot X) = ((b\ a)\cdot X)} \quad (\text{ax}_{(\lambda a.a)X = X})
\]

The reader can easily check from Definition 2.5 that \((b\ a)\cdot ((b\ a)\cdot X) \equiv id\cdot X\).

Both versions are correct, because of a known property of nominal terms which we have called object-level equivariance:

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

for any \(\Delta\), \(a\), \(t\) and \(\pi\). This is characteristic of nominal techniques (e.g. [54, Lemma 2.7], [22, Lemma 20], [31, Appendix A], [32, Lemma 4.7]).

- \((\text{fr})\): 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 instantiated, since our syntax is finite and must mention finitely many atoms. \((\text{fr})\) adds no deductive power to CORE but it does in the presence of axioms; a full discussion with an example is in [39, Lemma 2.3.18].

- \((\text{perm})\): this rule expresses \(\alpha\)-equivalence (Lemma 2.16 and Theorem 3.10). For instance, \((\text{perm})\) allows us to show the following standard \(\alpha\)-equivalence property:

\[
\frac{(\#a\ b)}{a\#(\lambda a. b) = (\lambda b. b)\#b} \quad (\text{perm}) \quad (\lambda a.a \equiv (a\ b)\cdot \lambda b.b)
\]

\((\text{perm})\) captures several rules from [54, Figure 2] (but not in a syntax-directed manner).

- \((\text{refl})\): choosing \(a, b \notin \text{atoms}(t)\) we can construct the derivation sketched below:

\[ \vdash t = (a\ b)\cdot t \]

\[ \vdash a\#(a\ b)\cdot t \quad \vdash b\#(a\ b)\cdot t \]

\[ \vdash t = (a\ b)\cdot t \quad (\text{perm}) \]

\[ \vdash a\#t \quad \vdash b\#t \]

\[ \vdash (a\ b)\cdot t = t \quad (\text{perm}) \]

\[ t = t \quad (\text{tran}) \]

\[ t = t \quad (\text{fr}) \quad \text{introducing } a\#X, b\#X \text{ for all unknowns } X \text{ in } t \]
Figure 4. β-equality with an α-conversion.

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.6

Remark 2.14
Nominal algebra is algebraic (even though the judgement form \( \Delta \vdash t = u \) has ‘\( \Delta \vdash \)’, which looks like an implication) in the following two senses:

- Nominal algebra is sound and complete for models in nominal sets [28, 39].
- These models are closed under notions of product, subalgebra, quotient (just like for traditional algebra)—and a nominal sets notion which 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 characterized by a nominal algebra theory [26].

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.

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

Proof. We give the derivation in full:

\[
\begin{align*}
\frac{b \# X}{a \# (b a) \cdot X} \quad \text{(refl)} & \quad \frac{a \# (b a) \cdot X}{b \# \lambda b.(b a) \cdot X} \quad \text{(perm)} & \quad \frac{\lambda a.X = \lambda b.(b a) \cdot X}{Y = Y} \quad \text{(cngapp)} \\
\frac{a \# (b a) \cdot X}{a \# \lambda b.(b a) \cdot X} \quad \text{(perm)} & \quad \frac{\lambda a.X = \lambda b.(b a) \cdot X}{Y = Y} \quad \text{(cngapp)} \\
\end{align*}
\]

The instance of (perm) relies on the fact that \( (b a) \cdot \lambda a.X = \lambda b.(b a) \cdot X \).

6It is a fact that (symm) and (tran) are admissible rules in \( \mathit{CORE} \) (a proof can be constructed using Theorem 2.19 below). This property fails in the presence of axioms, for instance, the axioms of ULAM and ULAME. On the other hand, we can view (refl) as sugar for the derivation given above using (perm), in any theory.
Lemma 2.17 is a nominal algebra rendering of the substitution lemma [7, Lemma 2.1.6] in terms of \( \beta \)-redexes.

**Lemma 2.17**

\[ a\#Y \vdash_{ULAM} (\lambda.b.((\lambda.a.Z)X))Y = (\lambda.a.((\lambda.b.Z)Y))((\lambda.b.X)Y). \]

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 [39]). But ULAM proves this, in logic.

**Proof.** By (\textbf{tran}) the proof obligation follows from:

\[ \begin{align*}
 (\lambda.b.((\lambda.a.Z)X))Y &= ((\lambda.b.((\lambda.a.Z))Y)((\lambda.b.X)Y) \\
 ((\lambda.b.((\lambda.a.Z))Y)((\lambda.b.X)Y) &= (\lambda.a.((\lambda.b.Z)Y))((\lambda.b.X)Y) \end{align*} \]

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

\[ \begin{array}{c}
 a\#Y \\
 \hline
 (\lambda.b.((\lambda.a.Z)X))Y = \lambda a.((\lambda.b.Z)Y) \\
 (\lambda.b.X)Y = (\lambda.b.X)Y \\
 \hline
 ((\lambda.b.((\lambda.a.Z))Y)((\lambda.b.X)Y) = (\lambda.a.((\lambda.b.Z)Y))((\lambda.b.X)Y) \\
 \end{array} \]

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

**Definition 2.18**

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

**Theorem 2.19**

\( \Delta \vdash_{CORE} t = u \) precisely when one of the following holds:

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

For a proof see Appendix 1; expanded details are also in [39, Corollary 2.5.4]. Thus, CORE induces the same theory of equality as the rules for equality from [54].

\section{The \( \lambda \)-calculus}

We give a short formal treatment of \( \lambda \)-terms and \( \alpha \beta (\eta) \)-equivalence.

**Definition 3.1**

Call a nominal term \textbf{ground} when it mentions no unknowns.\(^7\)

\(^7\)Ground terms should not be confused with closed lambda terms, i.e. terms without free atoms; closed terms are not used in this article.
As discussed in Section 2.1 our nominal terms syntax is specialized to the λ-calculus; ground terms $g, h, k$ are characterized by:

$$ g ::= a | \lambda a.g | g.g | c. $$

Thus, ‘ordinary’ λ-terms are a subset of the nominal terms from Definition 2.3, obtained by excluding unknowns. This should be no surprise, since unknowns are in Definition 2.3 to represent meta-variables.

**Definition 3.2**

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

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

**Lemma 3.3**

$a \not\in fa(g)$ if and only if $\vdash a \# g$.

Also, if $a \not\in atoms(g)$ then $\vdash a \# g$.

**Definition 3.4**

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

- $|a| = 1$
- $|\lambda a.g| = |g| + 1$
- $|g'g| = |g'| + |g| + 1$
- $|c| = 1$

**Definition 3.5**

Fix an infinite set of atoms $F \subseteq A$, such that $A \setminus F$ is also infinite.

**Definition 3.6**

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

- $a[h/a] = h$
- $b[h/a] = b$
- $(\lambda a.g)[h/a] = \lambda b.g(h[a])$ (if $b \not\in fa(h)$)
- $(\lambda b.g)[h/a] = \lambda b'.(g[b'/b][h/a])$ (if $b' \in F$, $b'$ fresh, in $F$)
- $(g'g)[h/a] = (g'[h/a])(g[h/a])$
- $c[h/a] = c$

In the clause for $(\lambda b.g)[h/a]$ we make some fixed but arbitrary choice of fresh $b' \in F$ for each $b, g, h, a$ (so $b' \in F \setminus atoms(g) \cup atoms(h) \cup \{a, b\}$).

**Remark 3.7**

We fix $F$ and make our choice of fresh $b'$ be from $F$ because this is convenient for our later proof of completeness of ULAM. This choice allows us to prove Lemma 3.15, which is needed to prove Lemma 4.13, and this in turn guarantees Lemma 4.17, which makes it easier for us to express a compact definition of the inverse translation in Definition 4.18, which is a key component of the proof of completeness (Theorem 4.7). This pattern is repeated in Section 5.

**Definition 3.8**

Let $\alpha$-**equivalence** $=_{\alpha}$ be the least transitive reflexive symmetric relation such that:

- (Name-abstraction.)
  
  If $g[c/a] =_{\alpha} g'[c/b]$ for fresh $c$ (so $c \not\in atoms(g, g')$) then $\lambda a.g =_{\alpha} \lambda b.g'$. 

510 Nominal Axiomatization of the Lambda Calculus

• (Congruence for \( \lambda \)) If \( g = a \cdot g' \) then \( \lambda a.g = a \cdot \lambda a.g' \).

• (Congruence for application.) If \( g = a \cdot g' \) and \( h = a \cdot h' \) then \( gh = a \cdot g' \cdot h' \).

**Lemma 3.9**
Suppose \( g \) is a ground term.

1. If \( a, b \notin \text{fat}(g) \) then \( (a \cdot b) \cdot g = a \cdot g \).
2. If \( b \notin \text{fat}(g) \) then \( g[b/a] = (b \cdot a) \cdot g \).

**Proof.** For the first part, we observe that since \( a, b \notin \text{fat}(g) \), any \( a \) and \( b \) that occur in \( g \) must occur in the scope of \( \lambda a \) and \( \lambda b \). 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 \( (a \cdot b) \cdot g' = g' \) because \( a, b \notin \text{atoms}(g') \). Equality is symmetric, so we reverse the process to return to \( g \).

The second part then follows by an induction on \( |g| \).

**Theorem 3.10**
On ground terms, derivable equality in \( \text{CORE} \) coincides with \( =_a \). (See also [39, Theorem 4.3.13].)

**Proof.** We must show that for ground terms \( g, h \),

\[ \vdash_{\text{CORE}} g = h \quad \text{if and only if} \quad g = a \cdot h. \]

We prove the left-to-right implication by induction on the structure of \( g \), using the syntactic criteria for \( \text{CORE} \) equality (Theorem 2.19). The cases of \( g \equiv a \) and \( g \equiv c \) follow by reflexivity, and the case of \( g \equiv g' \cdot g' \) follows by congruence using the inductive hypothesis. Now suppose \( g \equiv \lambda a.g' \), then there are two possibilities:

(i) \( h = \lambda a.h' \) and \( \vdash_{\text{CORE}} g' = h' \). Then \( g' = a \cdot h' \) by the inductive hypothesis, and we conclude \( \lambda a.g' = a \cdot \lambda a.h' \) by congruence.

(ii) \( h = \lambda b.h' \), \( \vdash b \# g' \) and \( \vdash_{\text{CORE}} (b \cdot a) \cdot g' = h' \). By Lemma 3.3 and some easy calculations we know \( a, b \notin \text{fat}(\lambda a.g') \), so \( \lambda a.g' = a \cdot \lambda b.(b \cdot a) \cdot g' \) by part 1 of Lemma 3.9 and symmetry. Also \( \lambda b.(b \cdot a) \cdot g' = a \cdot \lambda b.h' \) by congruence and the inductive hypothesis. We conclude \( \lambda a.g' = a \cdot \lambda b.h' \) by transitivity.

Conversely suppose that \( g = a \cdot h \). It suffices to show that equality in \( \text{CORE} \) can simulate every derivation rule of \( =_a \). We treat the only non-trivial case.

Suppose we have deduced \( \lambda a.g = \lambda b.h \) from \( g[c/a] = a \cdot h[c/b] \), where \( c \) is fresh (so \( c \notin \text{atoms}(g, h) \)). By Lemma 3.3 we know \( \vdash c \# g \) and \( \vdash c \# h \), with which we can show

\[ \vdash_{\text{CORE}} g[c/a] = (c \cdot a) \cdot g \quad \text{and} \quad \vdash_{\text{CORE}} h[c/b] = (c \cdot b) \cdot h \]

by an induction on \( |g| \) and \( |h| \).

Now also \( \vdash_{\text{CORE}} g[c/a] = h[c/b] \) by the inductive hypothesis. Then we obtain

\[ \vdash_{\text{CORE}} \lambda c.(c \cdot a) \cdot g = \lambda c.(c \cdot b) \cdot h. \]

by (symm), (tran) and (cong[]).

By (perm) \( \vdash_{\text{CORE}} \lambda c.(c \cdot a) \cdot g = \lambda a.g \) and \( \vdash_{\text{CORE}} \lambda c.(c \cdot b) \cdot h = \lambda b.h \), since \( \vdash c \# g \) and \( \vdash c \# h \). Using (symm) and (tran) we conclude that \( \vdash_{\text{CORE}} \lambda a.g = \lambda b.h \).
Nominal Axiomatization of the Lambda Calculus

Remark 3.11
We do not quotient terms by $\alpha$-conversion and we do not use a nominal-style datatype of syntax-with-binding [32]. Later on the proof method for Theorem 4.7 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 4.18.

We conclude this section with some basic definitions and lemmas, which will be useful later.

Definition 3.12
Let (one step) $\beta$-reduction $g \rightarrow_\beta h$ be defined by:

1. $(\lambda a.g)h \rightarrow_\beta g[h/a]$.
2. If $g \rightarrow_\beta g'$ then $\lambda a.g \rightarrow_\beta \lambda a.g'$.
3. If $g \rightarrow_\beta g'$ then $gh \rightarrow_\beta g'h$.
4. If $h \rightarrow_\beta h'$ then $gh \rightarrow_\beta gh'$.

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

Let (multi step) $\alpha\beta\eta$-equality $=_{\alpha\beta\eta}$ be the least transitive reflexive relation containing $\leftrightarrow_\beta$ and $=_{\alpha}$.

Definition 3.13
Let (one step) $\eta$-contraction be defined by:

1. $\lambda a.(ga) \rightarrow_\eta g$ if $a \notin \text{fa}(g)$.
2. If $g \rightarrow_\eta g'$ then $\lambda a.g \rightarrow_\eta \lambda a.g'$.
3. If $g \rightarrow_\eta g'$ then $gh \rightarrow_\eta g'h$.
4. If $h \rightarrow_\eta h'$ then $gh \rightarrow_\eta gh'$.

Let (one step) $\eta$-equality $\leftrightarrow_\eta$ be defined by $g \leftrightarrow_\eta h$ when $g \rightarrow_\eta h$ or $h \rightarrow_\eta g$.

Let (multi step) $\alpha\beta\eta$-equality $=_{\alpha\beta\eta}$ be the least transitive reflexive relation containing $\leftrightarrow_\eta$, $\leftrightarrow_\beta$, and $=_{\alpha}$.

Definition 3.14
Define the bound atoms $ba(g)$ by:

\[ ba(a) = \emptyset \quad ba(\lambda a.g) = ba(g) \cup \{a\} \quad ba(g')g = ba(g') \cup ba(g) \quad ba(c) = \emptyset \]

Lemma 3.15
For ground terms $g, g'$:

(i) If $g \rightarrow_\beta g'$ and $ba(g) \subseteq \mathcal{F}$, then $ba(g') \subseteq \mathcal{F}$.
(ii) If $g \rightarrow_\eta g'$ and $ba(g) \subseteq \mathcal{F}$, then $ba(g') \subseteq \mathcal{F}$.

Proof: The first part is by routine calculations, using the fact that in Definition 3.6 we choose fresh atoms from $\mathcal{F}$.

The second part is trivial, since $\eta$-contraction eliminates a bound variable.

Lemma 3.16
For ground terms $g, g', h$:

(i) If $g =_{\alpha} g'$ and $g \rightarrow_\beta h$ then there exists some $h'$ such that $g' \rightarrow_\beta h'$ and $h' =_{\alpha} h$.
(ii) If $g =_{\alpha} g'$ and $g \rightarrow_\eta h$ then there exists some $h'$ such that $g' \rightarrow_\eta h'$ and $h' =_{\alpha} h$. 
4  Soundness, completeness and conservativity for $\alpha\beta$

In Section 2, we presented nominal algebra and the theory ULAM. We saw formal derivations reminiscent of the ‘informal meta-level’. This informal meta-level is made formal using nominal terms; object-level variables become atoms, and meta-level variables become unknowns. ULAM axiomatizes the $\lambda$-calculus up to $\alpha\beta$-equivalence within this framework. Theorems 4.3 and 4.7 make that formal.

4.1 Soundness

Definition 4.1

Call $\sigma$ a ground substitution for a set of unknowns $X$ when $\sigma(X)$ is ground for every $X \in X$. Call $\sigma$ ground for $\Delta, t, u$ when $\sigma$ is ground for the set of unknowns appearing anywhere in $\Delta, t, or u$.

So ground substitutions eliminate all metavariables.

Definition 4.2

Write $\Delta \models_{\alpha\beta} t = u$ when $t \sigma =_{\alpha\beta} u \sigma$ (Definition 3.12) for all ground substitutions $\sigma$ for $\Delta, t, u$ such that $a \notin f_a(\sigma(X))$ for every $a \in X \in \Delta$.

Theorem 4.3 (Soundness)

For any $\Delta, t, u$, if $\Delta \vdash_{ULAM} t = u$ then $\Delta \models_{\alpha\beta} t = u$.

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

- The cases (refl), (symm), (tran), (cng$\lambda$) and (cngapp) follow by induction using the fact that $=_{\alpha\beta}$ is an equivalence relation and a congruence.
- The case (perm). Suppose $a, b \notin f_a(g)$. By part 1 of Lemma 3.9, $(a b) \cdot g =_{\alpha\beta} g$ and $(a b) \cdot g =_{\alpha\beta} g$

follows.
- The case (fr). Unknowns are irrelevant because ground terms by definition do not contain them. If $\sigma(X)$ mentions an atom which (fr) generates fresh for some $X$ in $\Delta, t, or u$, then we ‘freshen’ the atom further to avoid an ‘name clash’.8
- The case (ax). The axioms of ULAM are all standard properties of the $\lambda$-calculus:
  - $(\lambda a.a) h =_{\alpha\beta} a h$.
  - If $a \notin f_a(g)$ then $(\lambda a.g) h =_{\alpha\beta} a g$.
  - $(\lambda a.(g' g)) h =_{\alpha\beta} (\lambda a.g' g) h ((\lambda a.g) h)$.
  - If $b \notin f_a(h)$ then $(\lambda a.(\lambda b.g)) h =_{\alpha\beta} a b . ((\lambda a.g) h)$.
  - $(\lambda a.g) a =_{\alpha\beta} a g$.

4.2 Completeness and conservativity

Recall our choice of $\mathcal{F}$ from Definition 3.5.

---

8We retain the inductive hypothesis of the ‘freshened’ derivation using the mathematical principle of ZFA equivariance [31, Appendix A]—or by performing induction instead on the depth of derivations, and proving that freshening atoms does not affect this measure.
Nominal Axiomatization of the Lambda Calculus

Definition 4.4
Fix a freshness context $\Delta$ and two terms $t$ and $u$. Let $A$ be the atoms mentioned anywhere in $\Delta$, $t$, or $u$, i.e. $A = \{ a \mid a \neq X \in \Delta \} \cup \text{atoms}(t,u)$. Let $X$ be the unknowns mentioned anywhere in $\Delta$, $t$, or $u$. For each $X \in X$ fix the following data:

- an order $a_1, \ldots, a_{3x}$ on the atoms in $A$ such that $a \neq X \notin \Delta$;
- two entirely fresh atoms $d_x$ and $e_x$ (so $d_x \notin F \cup A$, and $e_x \notin F \cup A$).

Write $D$ for $\{ d_x \mid X \in X \}$ and $E$ for $\{ e_x \mid X \in X \}$.

Definition 4.5
Specify $\varsigma$ a ground substitution for $X$ by:

- $\varsigma(X) = e_x(d_x, a_1, \ldots, a_{3x})$ when $X \in X$, and
- $\varsigma(X) = X$ otherwise (the choice of $X$ in the right-hand side is irrelevant).

Lemma 4.6 is easy and will be useful.

Lemma 4.6
If $a \neq X \in \Delta$ then $a \notin \text{fat}(\varsigma(X))$.

Proof: By construction of $\Delta$ and $\varsigma$ (Definitions 4.4 and 4.5) $a$ differs from all of the $a_i$.

Theorem 4.7 (Completeness)
If $\Delta \vdash_{ULAM} t = u$ then $\Delta \vdash_{\text{ULAM}} t = u$.

The proof of this result uses the $\varsigma$ constructed above, and it is technical. Therefore, we defer it to Section 4.3 and mention Corollary 4.8 and Theorem 4.12, which are two forms of conservativity result for ULAM over CORE.

Corollary 4.8
Suppose $g$ and $h$ are ground. Then $\vdash_{ULAM} g = h$ if and only if $g =_{ULAM} h$.

Proof: By Theorems 4.3 and 4.7, using that fact that $g \equiv h$ and $h \equiv h$.

Definition 4.9
Call a ground term $g$ a $\beta$-normal form when no $g'$ exists with $g \rightarrow \beta g'$.

Lemma 4.10
Fix $\Delta$. Suppose that $t$ and $u$ contain no subterm of the form $(\lambda a.v)w$. Then for $\varsigma$ the ground substitution from Definition 4.5, $t \varsigma$ and $u \varsigma$ are $\beta$-normal forms.

Proof. $\varsigma(X) = e_x(d_x, a_1, \ldots, a_{3x})$ for every $X$ appearing in $\Delta$, $t$, $u$. Applying this substitution to $t$ and $u$ cannot introduce subterms of the form $(\lambda a.v)w$.

Lemma 4.11
$t \varsigma =_{\alpha} u \varsigma$ implies $\Delta, t \vdash_{\text{CORE}} t = u$.

Proof. We prove by induction on $t'$ that if $t'$ is a subterm of $t$ and $u'$ a subterm of $u$ then $t' \equiv_{\alpha} u' \varsigma$ implies $\Delta, t \vdash_{\text{CORE}} t' = u'$.

The interesting case is when $t' = \pi \cdot X$. Suppose $e_x(d_x, \pi(a_1), \ldots, \pi(a_{3x})) =_{\alpha} u' \varsigma$. Then it must be that $u' \varsigma \equiv e_x(d_x, \pi(a_1), \ldots, \pi(a_{3x}))$. By the construction of $u' \varsigma$ and the way we chose $a_1, \ldots, a_{3x}$ to be the atoms mentioned in $\Delta$, $t$, or $u$ which are not provably fresh for $X$ in $\Delta$, it follows that $u'$ must take the form $\pi' \cdot X$, for some $\pi'$ such that $\Delta, t \vdash_{\text{ULAM}} t = u'$ as required.
Theorem 4.12 (Conservativity)  
Suppose that \( t \) and \( u \) contain no subterm of the form \((\lambda a.v)w\). Then

\[
\Delta \vdash_{ULAM} t = u \quad \text{if and only if} \quad \Delta \vdash_{CORE} t = u.
\]

Proof. A derivation in \( \text{CORE} \) is also a derivation in \( \text{ULAM} \) so the right-to-left implication is immediate.

Now suppose that \( \Delta \vdash_{ULAM} t = u \). We construct \( \zeta \) as in Definition 4.5. By Theorem 4.3, \( t \equiv_{\alpha \beta} u \). By Lemma 4.10 we know that \( t \equiv \alpha u \). By Definition 4.11 \( \Delta \vdash_{\text{CORE}} t = u \), as required.

4.3 Proof of Theorem 4.7  
Recall the definition of \( bat(g) \) from Definition 3.14.

Lemma 4.13  
There exists a chain

\[
t \equiv g_1 \leftrightarrow g_2 \leftrightarrow g_3 \leftrightarrow \ldots \leftrightarrow g_{m-1} \leftrightarrow g_m \equiv u \zeta
\]

where each \( \leftrightarrow \) is one of \( \equiv_{\alpha} \) or \( \equiv_{\beta} \), which is such that \( bat(g_i) \cap (D \cup E) = \emptyset \) for \( 1 \leq i \leq m \) (so none of the terms in the chain contains ‘\( \alpha d_k \)’ or ‘\( \lambda e_k \)’ for any \( d_k \in D \) or \( e_k \in E \)).

Proof. By assumption \( \Delta \vdash_{\alpha} t = u \), so by Lemma 4.6, we know \( t \equiv_{\alpha} u \). It follows that there is a chain

\[
t \equiv g'_1 \leftrightarrow g'_2 \leftrightarrow g'_3 \leftrightarrow \ldots \leftrightarrow g'_{m'-1} \leftrightarrow g'_m \equiv u \zeta
\]

where each \( \leftrightarrow \) is one of \( \equiv_{\alpha} \) or \( \equiv_{\beta} \).

By construction, \( bat(g_i) \cap (D \cup E) = \emptyset \) for \( i = 1 \) and \( i = m' \). We transform this into a chain such that all terms satisfy this property.

Suppose \( h \rightarrow_{\beta} h' \) is a link in the chain and such that \( bat(h) \cap (D \cup E) \neq \emptyset \) (so \( h \equiv g'_i \) and \( h' \equiv g'_{i+1} \), or \( h \equiv g'_i \) and \( h' \equiv g'_{i+1} \), for some \( i \)). We use Lemma 3.16 to replace this link with \( h \equiv_{\alpha} h' \to_{\beta} h'' \equiv_{\alpha} h' \), where we choose \( h'' \) such that \( bat(h'') \cap (D \cup E) = \emptyset \). By part 1 of Lemma 3.15 also \( bat(h'') \cap (D \cup E) \neq \emptyset \).

We iterate the replacement above as much as possible. Now suppose \( h \equiv_{\alpha} h' \equiv_{\alpha} h'' \) is a pair of links in the chain. Since \( \equiv_{\alpha} \) is transitive, we replace this with \( h \equiv_{\alpha} h'' \).

It is easy to check that the final chain has the form

\[
t \equiv g_1 \leftrightarrow g_2 \leftrightarrow g_3 \leftrightarrow \ldots \leftrightarrow g_{m-1} \leftrightarrow g_m \equiv u \zeta
\]

where \( bat(g_i) \cap (D \cup E) = \emptyset \) for all \( 1 \leq i \leq m \), as required.

We can make the following definition.

Definition 4.14  
Make a fixed but arbitrary choice of chain as specified in Lemma 4.13.

Definition 4.15  
Let \( \mathcal{A}^+ \) be the set of all atoms mentioned anywhere in the chain we fixed in Definition 4.14, and let \( \Delta^+ \) be \( \Delta \) enriched with freshness assumptions \( a \# X \) for every \( a \in \mathcal{A}^+ \setminus \mathcal{A} \) and every \( X \in \mathcal{X} \).
Nominal Axiomatization of the Lambda Calculus

Definition 4.16
Call a ground term \( g \) accurate when:

- atoms\((g) \subseteq A^+ \) (\( g \) mentions only atoms in \( A^+ \)); and
- \( \text{ba}(g) \cap (D \cup E) = \emptyset \) (\( g \) does not contain \( \lambda d \cdot X \) or \( \lambda e \cdot X \) for any \( d \in D \) or \( e \in E \)).

Lemma 4.17
\( g_1, \ldots, g_m \) are accurate.

Proof. Trivially, from the construction in Lemma 4.13 and from our choice of \( A^+ \). \(\square\)

Definition 4.18
Define an inverse translation from accurate ground terms to (possibly non-ground) terms inductively by:

\[
\begin{align*}
a^{-1} &= a \\
(\lambda a \cdot g)^{-1} &= \lambda a \cdot (g^{-1}) \\
(gh)^{-1} &= (g^{-1})(h^{-1}) \\
c^{-1} &= c \\
(d_k)^{-1} &= \lambda x_1 \cdots \lambda x_{k+1} \cdot X \\
(e_x)^{-1} &= \lambda e_x \cdot e_x \\
(c^{-1})^{-1} &= c
\end{align*}
\]

We need some technical lemmas.

Lemma 4.19
Suppose that \( g \) is accurate. For any \( a \in A^+ \), if \( a \not\in \text{fa}(g) \) then \( \Delta^+ \vdash a \# g^{-1} \).

Proof. By induction on \( g \). The non-trivial case is when \( g \equiv d_k \). If \( a \not\in \text{fa}(d_k) \) then \( a \neq d_k \) and we must show

\( \Delta^+ \vdash a \# \lambda x_1 \cdots \lambda x_{k+1} \cdot X \),

which follows using the rules for freshness and the fact that the atoms that might not be fresh for \( X \) are precisely the \( a_{x_i} \). \(\square\)

Lemma 4.20
Suppose that \( g \) is accurate. Suppose that \( \pi \) is a permutation such that \( \pi(a) = a \) for all \( a \not\in A^+ \setminus (D \cup E) \). Then \( \Delta^+ \vdash \text{CORE} (\pi \cdot g)^{-1} = \pi \cdot (g^{-1}) \).

Proof. By a routine induction on \( g \). In the case of \( g \equiv d_k \) we use the fact that \( \pi(a) = a \) for all \( a \in (D \cup E) \). \(\square\)

Lemma 4.21
Suppose that \( g \) and \( h \) are accurate. Then \( g =_\alpha h \) implies \( \Delta^+ \vdash \text{CORE} g^{-1} = h^{-1} \).

Proof. By Theorem 3.10 \( g =_\alpha h \) coincides with \( \vdash \text{CORE} g = h \). 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

\[
g = \lambda a \cdot g', \quad h = \lambda b \cdot h', \quad \vdash b \# g', \quad \text{and} \quad \vdash \text{CORE} (b \cdot a) \cdot g' = h'.
\]

We must show \( \Delta^+ \vdash \text{CORE} \lambda a \cdot (g')^{-1} = \lambda b \cdot (h')^{-1} \). By transitivity and congruence, it suffices to show the following:

- \( \Delta^+ \vdash \text{CORE} \lambda a \cdot (g')^{-1} = \lambda b \cdot (h')^{-1} \). By (perm) and the rules for freshness, this follows from \( \Delta^+ \vdash b \# (g')^{-1} \). Using Lemma 4.19, this follows from our assumption that \( \vdash b \# g' \).
Then $\lambda a. g' \beta b$ is accurate so $\lambda a. g' \beta b \in \Delta^+$. The result follows by Lemma 4.20.

Lemma 4.22
Suppose that $a \in A^+ \setminus (D \cup E)$. Suppose that $g$, $h$, and $g[h/a]$ are accurate. Then $\Delta^+ \vdash_{ULAM} (\lambda a. (g^{-1}))(h^{-1}) = (g[h/a])^{-1}$.

Proof. By induction on $|g|$ (Definition 3.4). We consider the cases in turn:

1. $a[h/a]$. $\Delta^+ \vdash_{ULAM} (\lambda a.a)(h^{-1}) = h^{-1}$ by axiom (\beta var).
2. $b[h/a]$ where $b \notin (D \cup E)$. So $\Delta^+ \vdash a \# b$, and $\Delta^+ \vdash_{ULAM} (\lambda a.b)(h^{-1}) = b$ by axiom (\beta #). (Recall that by our permutative convention, $b$ ranges over atoms other than $a$.)
3. $c[h/a]$. So $\Delta^+ \vdash a \# c$, and $\Delta^+ \vdash_{ULAM} (\lambda a.c)(h^{-1}) = c$ by axiom (\beta #). (Recall from Definition 2.1 that $c$ ranges over constant symbols.)
4. $d_\ast[h/a]$ where $d_\ast \in D$. By assumption $a \neq d_\ast$, so we must show

\[
\Delta^+ \vdash_{ULAM} (\lambda a. (\lambda a_1 \cdots \lambda a_{\ast\ast} . X))(h^{-1}) = \lambda a_1 \cdots \lambda a_{\ast\ast} . X.
\]

Using axiom (\beta #) (and (\text{tran}) and (\text{refl})) it suffices to show

\[
\Delta^+ \vdash a \# \lambda a_1 \cdots \lambda a_{\ast\ast} . X.
\]

This follows from Lemma 4.19 and the fact that $a \notin fa(d_\ast) = \{d_\ast\}$.
5. $e_\ast[h/a]$ where $e_\ast \in E$. By assumption $a \neq e_\ast$, so we must show

\[
\Delta^+ \vdash_{ULAM} (\lambda a. (\lambda e_1 . e_\ast))(h^{-1}) = \lambda e_1 . e_\ast.
\]

Using axiom (\beta #) (and (\text{tran}) and (\text{refl})) it suffices to show

\[
\Delta^+ \vdash_{ULAM} (\lambda a. (\lambda e_1 . e_\ast))(h^{-1}) = \lambda e_1 . e_\ast.
\]

We derive this using the derivation rules for freshness (Figure 2).
6. $\lambda a. g[h/a]$. $\Delta^+ \vdash_{ULAM} (\lambda a. (\lambda a . (g^{-1}))(h^{-1}) = \lambda a . (g^{-1}))$ by axiom (\beta #).
7. $\lambda b. g[h/a]$ where $b \notin fa(h)$. By assumption $\lambda b . g$ is accurate, therefore $b \notin (D \cup E)$. By Definition 3.6, $(\lambda b . g)[h/a] = \lambda b . (g[b'/b][h/a])$ for fresh $b'$ (so $b' \notin atoms(g, h)$). By assumption $\lambda b . (g[b'/b][h/a])$ is accurate, and so $b' \notin (D \cup E)$ and $g[b'/b][h/a]$ is accurate. We must show

\[
\Delta^+ \vdash_{ULAM} (\lambda a. (\lambda b . (g^{-1}))(h^{-1}) = \lambda b . (g[b'/b][h/a])^{-1}).
\]
Note that by Lemma 4.19, \( \Delta^+ \vdash b' \# g^{-1} \) and \( \Delta^+ \vdash b' \# h^{-1} \), and \( \Delta^+ \vdash b' \# _b (g^{-1}) \) follows by \( (\#_b) \). Also \( \Delta^+ \vdash b \# _b.b. (g^{-1}) \) is immediate by \( (\#_a) \). We present the rest of the proof as a calculation:

\[
\begin{align*}
\lambda b'. (g[b'/b][h/a])^{-1} \\
= \{ \text{inductive hypothesis, since } (b' b) \cdot g \text{ is accurate} \} \\
\lambda b'. ((\lambda a. ((b' b) \cdot g)[h/a])^{-1} \\
= \{ \text{Lemma 4.20} \} \\
\lambda b'. ((\lambda a. ((b' b) \cdot g)^{-1})(h^{-1})) \\
= \{ \text{axiom } (\beta) \text{ since } \Delta^+ \vdash b' \# h^{-1} \} \\
(\lambda a. (\lambda b'. ((b' b) \cdot g^{-1}))(h^{-1}) \\
= \{ \text{perm} \text{ since } \Delta^+ \vdash b \# b. (g^{-1}) \text{ and } \Delta^+ \vdash b' \# _b.b. (g^{-1}) \} \\
(\lambda a. (\lambda b. (g^{-1}))(h^{-1}) \\
\}
\]

\( (g'g)[h/a] \). By axiom \( (\beta_{\text{app}}) \) and the inductive hypothesis.

The result follows.

**Corollary 4.23**
Suppose that \( g \) and \( h \) are accurate. If \( g \Leftrightarrow \beta h \) then \( \Delta^+ \vdash _{\text{ULAM}} g^{-1} = h^{-1} \).

**Proof.** By induction on the rules for \( \beta \) from Definition 3.12. It suffices to show the following (here \( g, g', h, h', \) and \( g[h/a] \) are accurate and \( a \in \mathcal{A}^+ \setminus (\mathcal{D} \cup \mathcal{E}) \)):

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

The first part is Lemma 4.22. The other parts follow by \( (\text{eng}_\lambda) \) and \( (\text{eng}_{\text{app}}) \).
Now let \( \pi_1 = (b_{x_1} \ a_{x_1}) \cdots (b_{x_k} \ a_{x_k}) \) and \( \pi_2 = (b_{a_{x_1}} \ a(\pi(a_{x_1})) \cdots (b_{x_1} \ a(\pi_2(a_{x_1}))) \). We will need the following properties of derivability in \( \text{CORE} \) involving these permutations:

\[
\Delta^+ \cup \Delta_B \vdash_{\text{CORE}} \lambda a_{x_1} \cdots \lambda a_{x_k} . X = \lambda b_{x_1} \cdots \lambda b_{x_k} . \pi_1 . X
\]

(3)

\[
\Delta^+ \cup \Delta_B \vdash_{\text{CORE}} \lambda b_{x_1} \cdots \lambda b_{x_k} . \pi_1 . X = \lambda \pi(a_{x_1}) \cdots \lambda \pi(a_{x_k}) . (\pi_2 \circ \pi_1) . X
\]

(4)

\[
\Delta^+ \cup \Delta_B \vdash_{\text{CORE}} (\pi_2 \circ \pi_1) . X = \pi . X
\]

(5)

We can see this as follows:

- Proof obligation (3) follows by \( k_x \) uses of Lemma 2.16 (and \( k_x - 1 \) instances of \( \text{eng}(\lambda) \)), since for \( 1 \leq i \leq k_x \),
  \[
  \Delta^+ \cup \Delta_B \vdash b_{x_i} \# \lambda a_{x_i,1} \cdots \lambda a_{x_i,k_x} (b_{x_i,1} a_{x_i}) \cdots (b_{x_i,k_x} a_{x_i}) . X.
  \]

- Proof obligation (4) follows by \( k_x \) uses of Lemma 2.16 (and \( k_x - 1 \) instances of \( \text{eng}(\lambda) \)), provided that for \( 1 \leq i \leq k_x \),
  \[
  \Delta^+ \cup \Delta_B \vdash \pi(a_{x_i}) \# \lambda b_{x_i,1} \cdots \lambda b_{x_i,k_x} (\pi(a_{x_i,1}) b_{x_i}) \cdots (\pi(a_{x_i,k_x}) b_{x_i}) . (\pi_2 \circ \pi_1) . X.
  \]

By the rules for freshness, this follows from \( \pi(a_{x_i}) \# (\pi_2 \circ \pi_1) . X \in \Delta^+ \cup \Delta_B \) since the \( \pi(a_{x_i}) \) are all disjoint and the \( b_{x_i} \) are different from \( \pi(a_{x_i}) \). We reason by cases on \( \pi(a_{x_i}) \):

- \( \pi(a_{x_i}) \neq a_0 \) for all \( j \): then \( \pi(a_{x_i}) \# X \in \Delta \) by Definition 4.4.
- \( \pi(a_{x_i}) = a_0 \) for some \( j \): then \( b_{x_i} \# X \in \Delta_B \).

- Proof obligation (5): by Theorem 2.19 it suffices to show \( \Delta^+ \cup \Delta_B \vdash ds(\pi_2 \circ \pi_1, \pi) \# X \). That is, we must show that \( a \# X \in \Delta^+ \cup \Delta_B \) for every \( a \) such that \( (\pi_2 \circ \pi_1)(a) \neq \pi(a) \), which follows by a case distinction on \( a \) (considering every \( a \in \text{supp}(\pi_2 \circ \pi_1) \cup \text{supp}(\pi) \)) using Definitions 4.4 and 4.15.

We use the properties above to construct the following ULAM derivation, presented in a calculational style:

\[
(\lambda . e_1. e_2)((\lambda a_{x_1} \cdots \lambda a_{x_k} . X)(a_{x_1}) \cdots (a_{x_k}))
\]

\[=\]

\{ Axiom (\text{beta}) \}

\[(\lambda a_{x_1} \cdots \lambda a_{x_k} . X)(a_{x_1}) \cdots (a_{x_k})\]

\[=\]

\{ (\text{cngapp}), (\text{refl}), and (3) \}

\[(\lambda b_{x_1} \cdots \lambda b_{x_k} . \pi_1 . X)(a_{x_1}) \cdots (a_{x_k})\]

\[=\]

\{ (\text{cngapp}), (\text{refl}), and (4) \}

\[(\lambda \pi(a_{x_1}) \cdots \lambda \pi(a_{x_k}) . (\pi_2 \circ \pi_1) . X)(a_{x_1}) \cdots (a_{x_k})\]

\[=\]

\{ Axiom (\text{bid}) \ (k_x \ times) \}

\[(\pi_2 \circ \pi_1) . X\]

\[=\]

\{ (5) \}

\[\pi . X\]

The result follows.

We are now ready to prove Theorem 4.7.
Nominal Axiomatization of the Lambda Calculus

Proof (of Theorem 4.7). Recall from Definition 4.14 the chain we fixed:
\[ t \equiv g_1 \leftrightarrow g_2 \leftrightarrow g_3 \leftrightarrow \ldots \leftrightarrow g_{m-1} \leftrightarrow g_m \equiv u \]

By Lemma 4.21 and Corollary 4.23:
\[ \Delta^+ \vdash_{ULAM} (t \equiv g_1 \leftrightarrow g_2 \leftrightarrow g_3 \leftrightarrow \ldots \leftrightarrow g_{m-1} \leftrightarrow g_m \equiv u) \]

so \[ \Delta^+ \vdash_{ULAM} (t \equiv g_1 \leftrightarrow g_2 \leftrightarrow g_3 \leftrightarrow \ldots \leftrightarrow g_{m-1} \leftrightarrow g_m \equiv u) \]

Our proof method works for the \( \lambda \)-calculus augmented with \( \eta \)-conversion (extensionality) as well, and the proofs change slightly. We outline how this works, emphasizing only the parts of previous sections that need changing.

Definition 5.1
Let ULAME be the nominal algebra theory with the axioms of ULAM from Figure 1, and with the additional axiom (\( \eta \)), as illustrated in Figure 5.

5 Soundness, completeness and conservativity for \( \alpha \beta \eta \)

We modify Definition 4.2 in the natural way.

Definition 5.2
Write \( \Delta \models_{\alpha \beta \eta} t = u \) when \( t \sigma =_{\alpha \beta \eta} u \sigma \) (Definition 3.13) for all ground substitutions \( \sigma \) for \( \Delta, t, u \) such that \( a \not\in fa(\sigma(X)) \) for every \( a \# X \in \Delta \).

Soundness is just like Theorem 4.3.

Theorem 5.3 (Soundness)
For any \( \Delta, t, u \), if \( \Delta \vdash_{ULAME} t = u \) then \( \Delta \models_{\alpha \beta \eta} t = u \).

Proof. Just as for Theorem 4.3. The induction on ULAME derivations has one extra axiom case:

- If \( a \not\in fa(g) \) then \( \lambda a.(ga) =_{\alpha \beta \eta} g \).

\[
\begin{align*}
(\beta_{\text{var}}) & \vdash (\lambda a.a)X = X \\
(\beta_{\#}) & \vdash a\#Z \rightarrow (\lambda a.Z)X = Z \\
(\beta_{\text{app}}) & \vdash (\lambda a.(Z'Z))X = ((\lambda a. Z')X)((\lambda a.Z)X) \\
(\beta_{\text{abs}}) & \vdash b\#X \rightarrow (\lambda a. (\lambda b.Z))X = \lambda b.((\lambda a.Z)X) \\
(\beta_{\text{id}}) & \vdash (\lambda a.Z)a = Z \\
(\eta) & \vdash a\#Z \rightarrow \lambda a.(Za) = Z
\end{align*}
\]

Figure 5. Axioms of ULAME.
5.2 Completeness and conservativity

Definitions 4.4, 4.5 and Lemma 4.6 are unchanged. Completeness Theorem 4.7 becomes as follows.

**Theorem 5.4 (Completeness)**
If $\Delta \vdash_{\eta\beta} t = u$ then $\Delta \vdash_{\text{ULAME}} t = u$.

We refer the proof to Subsection 5.3.

**Corollary 5.5**
For any ground terms $g, h$, $\Delta \vdash g = h$ if and only if $g =_{\eta\beta} u = h$.

**Proof.** By Theorems 5.3 and 5.4, using the fact that $g$ and $h$ are ground. $\blacksquare$

Definition 4.9 and Lemma 4.10 are changed as follows.

**Definition 5.6**
Call $g$ a $\eta\beta$-normal form when no $g'$ exists with $g \rightarrow_{\beta} g'$ or $g \rightarrow_{\eta} g'$.

**Lemma 5.7**
Fix $\Delta$. Suppose that $t$ and $u$ contain no subterm of the form $(\lambda a.v)w$ or $\lambda a.(va)$ where $\Delta \vdash a\#v$. Then for $\varsigma$ the ground substitution from Definition 4.5, $t\varsigma$ and $u\varsigma$ are $\beta\eta$-normal forms.

**Proof.** $\varsigma(X) \equiv e_{\varsigma}(d_{\varsigma}a_{\varsigma}1...a_{\varsigma}k_{\varsigma})$ for every $X$ appearing in $\Delta$, $t$, $u$. Applying this substitution to $t$ and $u$ cannot introduce subterms of the form $(\lambda a.v)w$ or $\lambda a.(va)$. $^9$

**Lemma 4.11** is unchanged, as it does not concern ULAM. Conservativity Theorem 4.12 becomes:

**Theorem 5.8 (Conservativity)**
Fix some $\Delta$. Suppose that $t$ and $u$ contain no subterm of the form $(\lambda a.v)w$ or $\lambda a.(va)$ where $\Delta \vdash a\#v$. Then

$\Delta \vdash_{\text{ULAME}} t = u$ if and only if $\Delta \vdash_{\text{CORE}} t = u$.

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

Now suppose that $\Delta \vdash_{\text{ULAME}} t = u$. We construct $\varsigma$ as in Definition 4.5. By Theorem 5.3, $t\varsigma =_{\eta\beta} u\varsigma$. By Lemma 5.7 we know that $t\varsigma$ and $u\varsigma$ are $\beta\eta$-normal forms. By confluence [7, Theorem 3.3.9 (i)] $t\varsigma =_{\eta} u\varsigma$. By Lemma 4.11 $\Delta \vdash_{\text{CORE}} t = u$, as required. $\blacksquare$

5.3 Proof of Theorem 5.4

Lemma 4.13 and Definition 4.14 are changed as follows.

**Lemma 5.9**
There exists a chain

$t\varsigma = g_1 \leftrightarrow ? g_2 \leftrightarrow ? g_3 \leftrightarrow ? \ldots \leftrightarrow ? g_{m-1} \leftrightarrow ? g_m =_{\eta\beta} u\varsigma$

where each $\leftrightarrow ?$ is one of $=_{\eta}, \leftrightarrow_{\beta},$ or $\leftrightarrow_{\eta}$, which is such that $ba(g_i) \cap (D \cup \xi) = \emptyset$ for $1 \leq i \leq m$.

$^9$We chose $e_{\varsigma}$ in Definition 4.4, and ‘wrapped’ $d_{\varsigma}a_{\varsigma}1...a_{\varsigma}k_{\varsigma}$ in $e_{\varsigma}$ in Definition 4.5, specifically to prevent accidental $\eta$-contracts here.
Nominal Axiomatization of the Lambda Calculus

Proof. By assumption $\Delta \models_{u\beta} t = u$, so by Lemma 4.6 we know $t_\xi = \alpha \beta \xi u \xi$. It follows that there is a chain

$$t_\xi \equiv g_1 \leftrightarrow g_2' \leftrightarrow \gamma g_3' \leftrightarrow \gamma \gamma g_4' \ldots \leftrightarrow \gamma g_{m-1}' \leftrightarrow g_m' \equiv u \xi$$

where each $\leftrightarrow \gamma$ is one of $\equiv_{u\alpha}$, $\equiv_{u\beta}$, or $\equiv_{u\eta}$. By construction, $ba(g_i) \cap (\mathcal{D} \cup \mathcal{E}) = \emptyset$ for $i = 1$ and $i = m'$. We use parts 1 and 2 of Lemma 3.15, and Lemma 3.16, to construct a chain

$$t_\xi \equiv g_1 \leftrightarrow g_2 \leftrightarrow g_3 \leftrightarrow \ldots \leftrightarrow g_{m-1} \leftrightarrow g_m \equiv u \xi$$

such that $ba(g_i) \cap (\mathcal{D} \cup \mathcal{E}) = \emptyset$ for all $1 \leq i \leq m$, just as in the proof of Lemma 4.13.

Definition 5.10

Make a fixed but arbitrary choice of chain as specified in Lemma 5.9.

Definitions 4.15 ($A^+$) and 4.16 ($g$ accurate) are unchanged. As before (Lemma 4.17), $g_1, \ldots, g_m$ are accurate by construction. Definition 4.18 (the inverse translation $g^{-1}$) is unchanged.

Lemmas 4.19–4.21 do not concern ULAME, and are unchanged. Lemma 4.22 becomes:

Lemma 5.11

Suppose that $a \in A^+ \setminus (\mathcal{D} \cup \mathcal{E})$. Suppose that $g$, $h$, and $g[h/a]$ are accurate. Then $\Delta^+ \vdash_{ULAME} (\lambda a. (g^{-1})(h^{-1})) = (g[h/a])^{-1}$.

Proof. By Lemma 4.22, since a derivation in ULAME is also one in ULAME.

Corollary 4.23 changes as follows.

Corollary 5.12

Suppose that $g$ and $h$ are accurate.

- If $g \leftrightarrow \beta h$ then $\Delta^+ \vdash_{ULAME} g^{-1} = h^{-1}$.
- If $g \leftrightarrow \eta h$ then $\Delta^+ \vdash_{ULAME} g^{-1} = h^{-1}$.

Proof. The first part is by Corollary 4.23, since a derivation in ULAME is also one in ULAME.

For the second part, it suffices to show that if $a \notin \text{fr}(g)$ then $\Delta^+ \vdash_{ULAME} \lambda a. (g^{-1}a) = g^{-1}$. This follows using axiom (\eta) and Lemma 4.19.

Lemma 4.24 changes as follows.

Lemma 5.13

$\Delta^+ \vdash_{ULAME} (t_\xi)^{-1} = t$, and $\Delta^+ \vdash_{ULAME} (u_\xi)^{-1} = u$.

Proof. From Lemma 4.24, since a derivation in ULAME is also one in ULAME.

We can now prove Theorem 5.4.

Proof (of Theorem 5.4) Recall from Definition 5.10 the chain

$$t_\xi \equiv g_1 \leftrightarrow g_2 \leftrightarrow g_3 \leftrightarrow \ldots \leftrightarrow g_{m-1} \leftrightarrow g_m \equiv u \xi$$

By Lemma 4.21 and Corollary 5.12

$$\Delta^+ \vdash_{ULAME} (t_\xi)^{-1} \equiv g_1^{-1} = g_2^{-1} = \ldots = g_{m-1}^{-1} \equiv (u \xi)^{-1},$$

so $\Delta^+ \vdash_{ULAME} (t_\xi)^{-1} = (u \xi)^{-1}$ by transitivity. By Lemma 5.13 then also $\Delta^+ \vdash_{ULAME} t = u$. Since $\Delta^+$ extends $\Delta$ with atoms that are not mentioned in $t$ and $u$ we extend the derivation with (fr) to obtain $\Delta \vdash_{ULAME} t = u$ as required.
6 Conclusions

6.1 Related work

What are functions, from a(n algebraic) logical point of view? This question has been studied from many angles.

6.1.1 Lambda-lifting

‘Lambda lifting’ introduces constant symbols to represent functions and adds axioms for them [34]. This expresses functions, but the axiomatization is of atomic constant symbols representing individual functions (as many as we would like to add) and not of the \( \lambda \)-calculus.

The issues of variables and binding surrounding the ‘\( \lambda \)’ in the ‘\( \lambda \)-calculus’ are avoided—or more precisely, they are 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 from a logical point of view we should consider this deeply unsatisfactory.

6.1.2 Combinatory algebra

Schönfinkel and Curry [19, 46] discovered combinatory algebra. The signature contains a binary term-former application and two constants \( S \) and \( K \). Axioms are \( K_{xy} = x \) and \( S_{xyz} = (xz)(yz) \).

This syntax is parsimonious and the axioms are compact, but it is not natural or ergonomic to program because it is impossible to go directly from a rule specifying a function, to the function itself—by \( \lambda \)-abstracting. In common with lambda lifting, this must be done at the meta-level.

There is another mathematical issue inherent to combinators themselves: the natural encoding of closed \( \lambda \)-terms into combinatory algebra syntax ([7, Section 7] or [48, Section 1.4]) is unsound; it does not map \( \alpha \beta \)-equivalent \( \lambda \)-terms to provably equal terms in combinatory algebra. We can strengthen combinatory algebra to lambda algebra by adding five more axioms [48, Proposition 5] but the translation is still not sound; there exist \( \lambda \)-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 [48, Proposition 20] (Selinger calls it ‘the notorious rule’). Thus, combinators do not capture the same functions as expressed by the \( \lambda \)-calculus.

6.1.3 Calculi of explicit substitutions

Calculi of explicit substitutions capture the \( \lambda \)-calculus in a first-order rewrite system; the idea originates in [1] and we note also Lescanne’s compact but readable survey [37]. If we orient the axioms of ULAM left-to-right then they become rewrite rules, and—although ULAM has no explicit substitution term-former—if we consider a \( \beta \)-reduct \( (\lambda a.t)u \) as an implicit substitution \( t[a\rightarrow u] \) then we can view the axioms of ULAM as ‘pushing substitutions’ down into a term, step-by-step. Thus, a derivation of an equality in ULAM looks very much like a sequence of explicit substitution rewrites, just not necessarily all oriented left-to-right because equality is symmetric.

ULAM displays two other notable differences from explicit substitution:

- the management of \( \alpha \)-equivalence (name-binding) comes from nominal techniques and is distinct from that explored in [1] or the subsequent literature. For an exploration of the computational cost of \( \alpha \)-equivalence in nominal terms (in the terminology of this article; of equality up to CORE) see [14].
The syntax of ULAM includes unknowns X, Y and Z which explicitly represent unknown terms. ‘Push a substitution down into a term’ is an analogy but it cannot always be fully realized; there is no way to push \([a \mapsto X]\) into Z (i.e. reduce \(\lambda a.Z\)X using \((βid)\), \((βapp)\), or \((βabs)\)) because Z is an unknown.

\((β#)\) resembles Bloo and Rose’s [12] ‘garbage collection rule’. There the rule is dispensable—removing it does not affect the transitive symmetric closure of the reduction relation [12, Remark 2.15]. In contrast, there may be models of ULAM in which Z can denote an element which cannot be represented by a term. For this reason \((β#)\) is not admissible in the presence of \((βid)\), \((βapp)\) and \((βabs)\).

6.1.4 Lambda abstraction algebras

Salibra’s lambda abstraction algebras (LAAs) [45] are a λ-calculus version of cylindric algebra [33]. There are many similarities with ULAM. For the convenience of the reader familiar with LAA syntax and semantics, a ‘cheat-sheet’ relating the material in this article with Definition 1, page 6 of [45] is provided below.

<table>
<thead>
<tr>
<th>ULAM</th>
<th>LAA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a, b, c)</td>
<td>(x, y, z)</td>
</tr>
<tr>
<td>(λa, λb, λc)</td>
<td>(λx, λy, λz)</td>
</tr>
<tr>
<td>(X, Y, Z)</td>
<td>(ξ, µ, ν)</td>
</tr>
<tr>
<td>((βvar))</td>
<td>((β_1))</td>
</tr>
<tr>
<td>((β#))</td>
<td>((β_2), (β_4))</td>
</tr>
<tr>
<td>((βapp))</td>
<td>((β_5))</td>
</tr>
<tr>
<td>((βabs))</td>
<td>((β_6))</td>
</tr>
<tr>
<td>((βid))</td>
<td>((β_3))</td>
</tr>
<tr>
<td>(\text{CORE})</td>
<td>((α))</td>
</tr>
</tbody>
</table>

ULAM exists in the nominal algebra framework; thus, at the level of syntax, there is one term-former \(λ\) for \(λ\)-abstraction. LAA includes infinitely many term-formers \(λx, λy, λz;\ldots\); and there are finitely many axioms and term formers. This is because LAA uses ‘ordinary’ universal algebra, whereas in nominal algebra, \(α\)-equivalence is handled primitively by the nominal algebra framework, and does not require special consideration in the signature or axioms.

Similarly, freshness side conditions appear ‘hard-wired’ in LAA axioms. For example, Salibra’s axiom \((β_4)\) from [45, page 6] \((λx.(λx.ξ))µ = λx.ξ\) is a version of \((β#)\) where the freshness condition is built into the structure of the term by writing \(λx.ξ\). Nominal algebra handles freshness primitively (the ‘freshness judgements’ \(a#t\) of Definition 2.8). The notion of algebraic dependence from [45, Definition 3] \('(λx.a)x ≠ z for some z’\) corresponds with the negation of our freshness judgement ‘\(a#X\)’ (in a rather unfortunate notation clash with our notation for atoms, Salibra uses \(a\) to range over arbitrary elements of a model). Algebraic (in)dependence is handled rigorously but informally (i.e. in the prose discourse of the paper) in Salibra’s work; for us, the corresponding notion of freshness is a formal part of the syntax of nominal algebra judgements, and freshness has a denotation in nominal sets going back to the original work on nominal techniques [32], which places LAA ‘algebraic (in)dependence’ in a broader and more general mathematical setting.

For a general theory of models of ULAM, see the general theory of models of nominal algebra theories [39]; recall that nominal algebra theories take models in nominal sets. Nominal sets have a finite-support property which, from the ‘cylindric’ point of view, corresponds with a property which...
is known and studied under the name \textit{locally finite or locally finite dimensional} \cite[Definition 7]{45}. Models of ULAM are finitely supported/locally finite by construction.\footnote{It may be prudent to note that being finitely supported / locally finite does not mean that in nominal sets we cannot consider models including infinitary syntax. Nominal sets are consistent with infinitary syntax, so long as it mentions only finitely many \textit{free} variables; it can mention as many bound variables as we like, including infinitely many. For a nominal-style theory of objects with infinite support, and in particular infinitary syntax with potentially infinitely many free variables, see \cite{25}.}

See the Related Work section of \cite{45} for further references to this and similar work.

6.1.5 Syntax quotiented by $\alpha\beta$-equivalence
The definition of $\alpha\beta$-equivalence on syntax is occasionally called ‘axiomatizing the $\lambda$-calculus’, although it is just an equivalence relation on abstract syntax trees. However, if the $\lambda$-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 $Q_0$ \cite[Section 51]{3} contains five axioms (\ref{eq:41}), (\ref{eq:42}), (\ref{eq:43}), (\ref{eq:44}) and (\ref{eq:45}) \cite[p. 164]{3}). In fact, these are axiom \textit{schemes}, containing meta-variables $A$ and $B$ 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, though the axioms of ULAM exist in the formal framework of nominal algebra—a formal logic, not an informal meta-level. The interested reader might like to consider a formalization of the arguments about $Q_0$ in \cite{3} into a formal logic (not a nominal one) \cite{51}.

6.1.6 Nominal techniques
A rewrite system for the $\lambda$-calculus appeared already in \cite{22} but without any statement or proof of completeness (indeed, it was not complete). More recently, the authors have used nominal algebra to axiomatize first-order logic as a theory FOL \cite{31} and substitution as a theory SUB \cite{29}.

One might suspect that the theory of substitution should be only a hair’s breadth away from that of the untyped $\lambda$-calculus, and could be obtained by imposing a type system (just as the simply-typed $\lambda$-calculus is related to the untyped $\lambda$-calculus).\footnote{We briefly give some intuition for what SUB is: $(\lambda b.(ba))(\lambda a.a) = a$ is derivable in ULAM, but only $\text{app}(b,a)(b) \mapsto \text{lam}(a)(a) = \text{app}(\text{lam}(a)(a),a)$ is derivable in SUB (notation from \cite{29}).} 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 tantalizingly 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 $\lambda$-calculus variables and nominal algebra unknowns.

Nominal algebra has a cousin, nominal equational logic (NEL) \cite{16}), which is very similar but makes slightly different design decisions. To the description of the relationship with nominal algebra given in \cite{16} should be added that (unsorted) NEL is a subsystem of nominal algebra. The translation described and discussed in \cite[Section 6.1.2]{30}; it relies on the encoding of freshness using equality given in \cite[Equation (32)]{32}. NEL and nominal algebra both satisfy completeness results for a class of models in nominal sets \cite[28, 39] in other words—derivable equality coincides with validity in all models). The completeness results in this article are stronger; they prove that derivable equality coincides with validity in particular models, which we built in this article (Definitions 4.2 and 5.2). Similarly for the authors’ treatments of substitution \cite{29} and first-order logic \cite{31}. We know of no like treatments of substitution, logic and the $\lambda$-calculus in NEL. If and when this is done it will be...
Nominal Axiomatization of the Lambda Calculus

interesting to compare the results; one way to do this is to transfer results using the translation of [30, Section 6.1.2].

6.1.7 Some concluding remarks on names, variables and functional abstraction
Nominal techniques are based on modelling names as atoms (urelemente) in set theory [13, 24], an idea raised by the Gabbay–Pitts models of $\alpha$-abstraction and $\Pi$ quantification [32]. These ideas find uses beyond the initial applications to syntax: in game theory and reasoning about pointers [2, 9, 53], spatial logics [38] 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.

For us, nominal techniques are a general theory of names, and nominal algebra is a formal logic with which to instantiate this general theory to more specific theories.

$\lambda$-calculus style variables are names that can be functionally abstracted. Nominal-style atoms are names that can be permuted and $\alpha$-abstracted. We have written down ULAM and ULAME and shown that they soundly and completely express those properties which, when added to nominal-style names, convert them into $\lambda$-calculus style variables. Conveniently, permutation then behaves like abstracted-then-applied variables (the device used, for example, in Miller’s higher order patterns [40] and in Salibra’s lambda-abstraction algebras [45])—and nominal-style $\alpha$-abstraction becomes $\lambda$-calculus style functional abstraction.

6.2 Future work and conclusions
ULAM completes a trio of papers studying (untyped) nominal algebra considered as a logical framework: first-order logic [31], substitution [29], and with this article, the $\lambda$-calculus. Between them, these works cover a significant fraction of the typical syntaxes of interest in theory and practice of computer science. It would of course be interesting to seek common generalizations of the proof-techniques therein.

Our most immediate interest is in constructing a theorem prover based on nominal algebra instead of the $\lambda$-calculus. We expect this to formally represent at least some kinds of reasoning better than the $\lambda$-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 (e.g. the pervasive use of meta-variables and freshness conditions, as appear in specifications of the $\lambda$-calculus). From that perspective this article proves a vital correctness result.

It remains to understand the connections between standard models for the $\lambda$-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 $\lambda$-calculus [7, 49] 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 [5, 48] 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 [26].

\footnote{This marks a difference from combinators, which share a notion of functional application with the $\lambda$-calculus but not the notion of functionally abstracting variable symbols.}
Acknowledgements

We thank an anonymous referee of a previous paper for comments which set us on the path to writing this one, the anonymous referees, and Chantal Berline for comments and support.

References

Nominal Axiomatization of the Lambda Calculus


Appendix

A Syntactic criteria for CORE equality

For completeness we include a proof of Theorem 2.19. Note that this is a basic result of theory CORE and hence of nominal algebra in general; it does not have to do with the \( \lambda \)-calculus, or with ULAM.

We will show that theory CORE is equivalent to an existing notion of \( \alpha \)-equivalence on nominal terms, which is syntax-directed by definition [22, 54]. Definition A1 was introduced in [54, Figure 2] (except that here, our syntax is specific to the \( \lambda \)-calculus).

**Definition A1**

Let \( t \approx_\Delta u \) be an ordered tuple of a term \( t \), a freshness context \( \Delta \), and a term \( u \). Let the **derivable equalities of** \( t \approx_\Delta u \) be inductively defined by the rules in Figure A1.
Nominal Axiomatization of the Lambda Calculus

\[
\begin{align*}
\text{(Ax)} & \quad a \triangleq_\Delta a \\
\text{(Ds)} & \quad \Delta \vdash ds(\pi, \pi')\#X \\
\text{(Fapp)} & \quad t \equiv_\Delta u' \quad t \equiv_\Delta u \\
\text{(Fc)} & \quad c \equiv_\Delta c \\
\end{align*}
\]

\[
\begin{align*}
\text{(Abs)} & \quad t \equiv_\Delta u \\
\text{(Absab)} & \quad (b\ a) \cdot t \equiv_\Delta u \\
\end{align*}
\]

\begin{figure}[h]
\centering
\begin{tabular}{cccc}
\hline
\text{LEMMA A2} & \text{(perm)} & \text{(tran)} \\
\hline
(\pi' \circ \pi) \cdot t & \equiv_\Delta (\pi' \cdot t) & \pi' \cdot t & \equiv_\Delta \pi' \cdot t \\
\hline
\end{tabular}
\caption{Syntax-directed rules for \text{CORE}.}
\end{figure}

\textbf{LEMMA A2}

\((\pi' \circ \pi) \cdot t \equiv_\Delta (\pi' \cdot t)\) and \(id \cdot t \equiv_\Delta t\).

\textbf{PROOF.} By routine inductions on \(t\). The only interesting case is \(t \equiv_\Delta \pi' \cdot X\) for the first part: by Definition 2.5 we can verify that

\[
(\pi' \circ \pi) \cdot (\pi' \cdot X) \equiv (\pi' \circ \pi \circ \pi') \cdot X \quad \text{and} \quad \pi' \cdot (\pi \circ (\pi' \cdot X)) \equiv (\pi' \circ \pi \circ \pi') \cdot X.
\]

\[
\]

Recall the notion of difference set \(ds(\pi, \pi')\) of two permutations \(\pi\) and \(\pi'\) from Definition 2.18. Lemma A3 shows how theory \text{CORE} can mimic the \text{(Ds)} rule of \(\equiv_\Delta\).

\textbf{LEMMA A3}

If \(\Delta \vdash ds(\pi, \pi')\#t\) then \(\Delta \vdash_{\text{CORE}} \pi \cdot t \equiv_\Delta \pi' \cdot t\).

\textbf{PROOF.} We work by induction on the number of elements in \(ds(\pi, \pi')\). If this set is empty then \(\pi = \pi'\) and the result follows easily by \text{(refl)}. Now suppose \(a \in ds(\pi, \pi')\). We construct a partial derivation of the proof obligation:

\[
\pi \cdot t = ((\pi(a) \pi'(a)) \circ \pi') \cdot t \quad ((\pi(a) \pi'(a)) \circ \pi') \cdot t = \pi' \cdot t
\]

(The instance of \text{(perm)} here is valid because by Lemma A2 \(((\pi(a) \pi'(a)) \circ \pi') \cdot t \equiv_\Delta ((\pi(a) \pi'(a)) \cdot \pi' \cdot t)\), and \(\pi' \cdot t \equiv_\Delta id \cdot (\pi' \cdot t)\).)

The following proof obligations remain:

- \(\pi \cdot t = ((\pi(a) \pi'(a)) \circ \pi') \cdot t\) follows from \(ds(\pi, (\pi(a) \pi'(a)) \circ \pi')\#t\) by the inductive hypothesis, provided that \(|ds(\pi, (\pi(a) \pi'(a)) \circ \pi')| < |ds(\pi, \pi')|\). This condition is satisfied, since \(ds(\pi, (\pi(a) \pi'(a)) \circ \pi') = ds(\pi, \pi') \setminus \{a\}\). The remaining proof obligation \(ds(\pi, (\pi(a) \pi'(a)) \circ \pi')\#t\) follows by assumption \(ds(\pi, \pi')\#t\).

- \(\pi(a) \# \pi' \cdot t\) follows from \((\pi')^{-1}(\pi(a))\#t\) by an easy calculation. Now this is one of the assumptions \(ds(\pi, \pi')\#t\) by Definition 2.18, we know that \((\pi')^{-1}(\pi(a)) \in ds(\pi, \pi')\) when \(\pi'((\pi')^{-1}(\pi(a))) \neq \pi(a)\), and, using the fact that \(\neq\) is invariant under permutation, this follows from the assumption \(\pi(a) \neq \pi'(a)\).

- \(\pi'(a) \# \pi' \cdot t\) follows from \(a\#t\) by an easy calculation. This follows directly from assumption \(ds(\pi, \pi')\#t\), since \(a \in ds(\pi, \pi')\).
Theorem A4 (Equivalence of \( \text{CORE} \) and \( \approx \))

\[ \Delta \vdash \text{CORE} t = u \] is derivable if and only if \( t \approx \Delta u \) is derivable using the rules of Figure A1.

**Proof.** The left-to-right direction is by induction on the structure of nominal algebra derivations of \( \Delta \vdash \text{CORE} t = u \). By the inductive hypothesis it suffices to show the following:

- Syntax-directed equality \( \approx \) is an equivalence relation and a congruence. This is [22, Theorem 24].
- If \( \Delta \vdash a \# t \) and \( \Delta \vdash b \# t \) then \((a b) \cdot t \approx \Delta u \). By induction on \( t \).
- If \( t \approx \Delta aX_1 \cdots aX_n \) where \( a \not\in t, \Delta \) then \( t \approx \Delta u \). By straightforward induction on the structure of derivations of \( t \approx \Delta aX_1 \cdots aX_n u \).

For the right-to-left direction we work by induction on derivations of \( t \approx \Delta u \). By the inductive hypothesis it suffices to show:

- \( \Delta \vdash \text{CORE} a = a \). This is an instance of (refl).
- If \( \Delta \vdash ds(\pi, \pi') \# X \) then \( \Delta \vdash \text{CORE} \pi \cdot X = \pi' \cdot X \). This is Lemma A3.
- If \( \Delta \vdash \text{CORE} t' = u' \) and \( \Delta \vdash \text{CORE} t = u \) then \( \Delta \vdash \text{CORE} t' = u' \). This is (congapp).
- \( \Delta \vdash \text{CORE} \lambda a.t = \lambda c.u \). This is (refl).
- If \( \Delta \vdash \text{CORE} \lambda a.t = \lambda a.u \) then \( \Delta \vdash \text{CORE} \lambda a.t = \lambda b.u \). Suppose that \( \Pi \) and \( \Pi' \) are derivations of \( \Delta \vdash \text{CORE} \lambda a.t = \lambda b.u \) respectively. Then the following is a derivation of \( \Delta \vdash \text{CORE} \lambda a.t = \lambda b.u \):

\[
\begin{array}{c}
\Pi' \\
\hline
b \# t \\
\hline
b \# \lambda a.t \\
\hline
a \# \lambda a.t \\
\hline
\lambda b. (a \# t) = \lambda a.t \\
\hline
\lambda a.t = \lambda b. (a \# t) \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\hline
(\# \lambda b) \\
\hline
(\# \lambda a) \\
\hline
\text{perm} \\
\hline
(\text{symm}) \\
\hline
(\text{cong} \lambda) \\
\hline
(\text{tran}) \\
\hline
\end{array}
\]

**Proof (of Theorem 2.19).** From Theorem A4, since the rules in Figure A1 render the criteria of Theorem 2.19 as derivation rules.