ACADEMIC PRESS

Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE @ DIRECT®

Information and Computation 186 (2003) 165–193

Information and Computation

www.elsevier.com/locate/ic

Nominal logic, a first order theory of names and binding

Andrew M. Pitts*

University of Cambridge Computer Laboratory, William Gates Building, J.J. Thomson Avenue, Cambridge CB3 0FD, UK

Received 18 March 2002; revised 27 May 2002

Abstract

This paper formalises within first-order logic some common practices in computer science to do with representing and reasoning about syntactical structures involving lexically scoped binding constructs. It introduces *Nominal Logic*, a version of first-order many-sorted logic with equality containing primitives for renaming via name-swapping, for freshness of names, and for name-binding. Its axioms express properties of these constructs satisfied by the *FM-sets* model of syntax involving binding, which was recently introduced by the author and M.J. Gabbay and makes use of the Fraenkel–Mostowski permutation model of set theory. Nominal Logic serves as a vehicle for making two general points. First, name-swapping has much nicer logical properties than more general, non-bijective forms of renaming while at the same time providing a sufficient foundation for a theory of structural induction/recursion for syntax modulo α-equivalence. Secondly, it is useful for the practice of operational semantics to make explicit the *equivariance property* of assertions about syntax – namely that their validity is invariant under name-swapping.

Keywords: Abstract syntax; Variable binding; Permutation; Fresh names

1. Introduction

It is commonplace, when using formal languages in computer science or mathematical logic, to abstract away from details of concrete syntax in terms of strings of symbols and instead work solely with parse trees – the ‘abstract syntax’ of a language. Doing so gives one access to two extremely useful and

* Fax: +44-1223-334678.
E-mail address: Andrew.Pitts@cl.cam.ac.uk.

0890-5401/$ - see front matter © 2003 Elsevier Science (USA). All rights reserved.
doi:10.1016/S0890-5401(03)00138-X

inter-related tools: definition by recursion on the structure of parse trees and proof by induction on that structure. However, conventional abstract syntax is not abstract enough if the formal language involves variable-binding constructs. In this situation the common practice of human (as opposed to computer) provers is to say one thing and do another. We say that we will quotient the collection of parse trees by a suitable equivalence relation of α-conversion, identifying trees up to renaming of bound variables; but then we try to make the use of α-equivalence classes as implicit as possible by dealing with them via suitably chosen representatives. How to make good choices of representatives is well understood, so much so that it has a name – the 'Barendregt Variable Convention': choose a representative parse tree whose bound variables are *fresh*, i.e., mutually distinct and distinct from any (free) variables in the current context. This informal practice of confusing an α-equivalence class with a member of the class that has sufficiently fresh bound variables has to be accompanied by a certain amount of hygiene on the part of human provers: our constructions and proofs have to be independent of which particular fresh names we choose for bound variables. Nearly always, the verification of such independence properties is omitted, because it is tedious and detracts from more interesting business at hand. Of course this introduces a certain amount of informality into 'pencil-and-paper' proofs that cannot be ignored if one is in the business of producing fully formalised, machine-checked proofs. But even if you are not in that business and are content with your pencil and paper, I think there is a good reason to examine this informal use of 'sufficiently fresh names' and put it on a more precise, mathematical footing.

The reason I have in mind has to do with those intuitive and useful tools mentioned above: structural recursion for defining functions on parse trees and structural induction for proving properties of them. Although it is often said that the Barendregt Variable Convention allows one to work with α-equivalence classes of parse trees as though they were just parse trees, this is not literally the case when it comes to structural recursion/induction. For example, when dealing with an induction step for a variable-binding construct, it often happens that the step can be proved for a sufficiently fresh bound variable, but not for an arbitrary one, as the induction principle demands. The Barendregt Variable Convention papers over the crack in the proof at this point by preventing one considering the case of an arbitrary bound variable rather than a fresh one, but the crack is still there. Although one can sometimes side-step the problem by using a suitable size function on parse trees and replacing structural induction with mathematical induction, this is not a very satisfying solution. The size function will be defined by structural recursion and the crucial fact that α-equivalent parse trees have the same size will be proved by structural induction; so we are using structural recursion/induction anyway, but somehow not in the direct way we would like. We can do better than this.

Indeed, the work reported in [16,17,35] does do better, by providing a mathematical notion of 'sufficiently fresh name' that remains very close to the informal practice described above while enabling α-equivalence classes of parse trees to gain useful inductive/recursive properties. The theory stems from the somewhat surprising observation that all of the concepts we need (α-equivalence, freshness, variable-binding, ...) can be defined purely in terms of the operation of *swapping* pairs of names. In particular, the freshness of a name for an object is expressed by saying that the name is not in some finite set of names that *supports* the object, which means that the finite set has the property that swapping any pair of names not in it leaves the object unchanged. This notion of support is weak second order, since it involves an existential quantification over finite sets of names. However, much of the development in [17] only makes use of certain first-order properties of the freshness (i.e., 'not-in-the-support-of') predicate in combination with the swapping operation. This paper presents this first-order theory of names, swapping and freshness, called *Nominal Logic*.

1.1. Outline of the paper

Section 2 presents some motivations for basing a theory of syntax and binders upon the notions of *atoms* (names), *swapping* atoms, and *freshness* of atoms. Section 3 introduces the syntax we use for these concepts, together with some typical examples of what can be expressed with them. As explained in [17], the Nominal Logic notions of atom, swapping and freshness can be given a meaning independent of any particular object-level syntax using *FM-sets* – the Fraenkel–Mostowski permutation model of set theory; in Section 4 we describe the category of *nominal sets*, which provides a simplified presentation of FM-sets emphasising swapping over more general permutations of atoms. Then in Section 5 we axiomatise the key first-order properties of the nominal sets model of atoms, swapping and freshness. Section 6 makes a definitional extension of this theory with a quantifier expressing a characteristic 'some/any' property of fresh atoms. In Section 7 we make another definitional extension to deal with variable-binding operations in a more uniform way. This completes the definition of Nominal Logic, which is summarised in Appendix A. Section 7 illustrates its use by presenting a first-order theory of λ-terms modulo α-equivalence containing a convenient structural induction axiom. Section 8 discusses the fact that Nominal Logic is incompatible with the use of choice functions to select a 'next' fresh atom in any particular context. Finally, Sections 9 and 10 describe some related approaches to fully formal treatments of names and binding and draw some conclusions.

2. Equivariant predicates

The fundamental assumption underlying Nominal Logic is that *the only predicates we ever deal with* (when describing properties of syntax) *are equivariant ones, in the sense that their validity is invariant under swapping* (i.e., transposing, or interchanging) *names*.

Names of what? Names of entities that may be subject to binding by some of the syntactical constructions under consideration. In Nominal Logic these sorts of names, the ones that may be bound and hence that may be subjected to swapping without changing the validity of predicates involving them, will be called *atoms*. The terminology refers back to the origins of the theory in the Fraenkel–Mostowski permutation model of set theory. Atoms turn out to have quite different logical properties from *constants* (in the usual sense of first-order logic) which, being constant, are not subjected to swapping. Note that this distinction between atom and constant has to do with the issue of binding, rather than substitution: a syntactic category of *variables*, by which is usually meant entities that may be subject to substitution, might be represented in Nominal Logic by atoms or by constants, depending upon circumstances: constants will do if we are in a situation where variables are never bound, but can be substituted for; otherwise we should use atoms. The interesting point is that we can make this (useful!) distinction between 'bindable' names and names of constants entirely in terms of properties of swapping names, prior to any discussion of substitution and its properties.

Why the emphasis on the operation of *swapping* two names, rather than on the apparently more primitive notion of *renaming* one name by another? The answer to this question lies in the combination of the following two facts.

- First, even though swapping seems less general than renaming (since after all, the act of swapping a and b can be expressed as the simultaneous renaming of b by a and a by b), it is possible to found a theory of syntax modulo α-equivalence, free and bound variables, substitution, etc., upon this notion – this is the import of the work in [17].

- Secondly, swapping is an involutive operation: a swap followed by the same swap is equivalent to doing nothing. This means that the class of equivariant predicates, i.e., those whose validity is invariant under atom-swapping, has excellent logical properties. It contains the equality predicate and is closed under negation, conjunction, disjunction, existential and universal quantification, formation of least and greatest fixed points of monotone operators, etc. The same is not true for renaming. For example, the validity of a negated equality between atoms is not necessarily preserved under renaming.

In other words we can found a theory of variable-binding upon swapping, and it is convenient to do so because of its good logical properties. Here are a couple of examples to illustrate these points, taken from λ-calculus and type theory.

Example 1 (*α-equivalence of λ-terms*)**.** Consider the terms of the untyped λ-calculus, which we can take to be α-equivalence classes $[t]_\alpha$ of parse trees t given by the grammar

$$t ::= a \mid \lambda a.t \mid t\, t \qquad (1)$$

where a ranges over an infinite set of variables. The relation of α-equivalence between such parse trees, $t \sim_\alpha t'$, is usually defined to be the congruence generated by relating $\lambda a.t$ and $\lambda b.\{b/a\}t$ if there are no occurrences of b in t (be they free, bound or binding occurrences). Here $\{b/a\}t$ is the parse tree obtained from t by replacing all free occurrences of a with b. The properties of this form of renaming are rather inconvenient for our aim of developing a theory of variable-binding in which logical equality subsumes α-equivalence. This is because the operation $\{b/a\}(-)$, as a total function on all parse trees, does not necessarily respect α-equivalence when applied to trees that do contain occurrences of b – because of the possible 'capture' of b by binders $\lambda b.(-)$ occurring in t. (For example $\lambda b.a \sim_\alpha \lambda c.a$ holds, but $\{b/a\}(\lambda b.a) = \lambda b.b \nsim_\alpha \lambda c.b = \{b/a\}(\lambda c.a)$.) In the development of the theory of λ-calculus [1], this inconvenient fact immediately leads to the formulation of more complicated, 'capture-avoiding' notions of renaming and substitution. However, it is possible to go in the other direction and replace $\{b/a\}(-)$ with another, equally simple form of renaming which does respect α-equivalence whatever term it is applied to. For as pointed out in [17, Section 2], if b does not occur in t, then $\{b/a\}t$ is α-equivalent to the parse tree obtained from t by *swapping all occurrences of a and b* (be they free, bound, or even binding occurrences): we denote this parse tree by $(a\, b)\cdot t$. The total function $(a\, b)\cdot(-)$ on parse trees is in a sense more fundamental than $\{b/a\}(-)$, because its definition does not depend upon knowing what is a free variable, i.e., upon knowing which of the syntax-constructors is supposed to be a binder: for the definition of $(a\, b)\cdot(-)$ on a λ-abstraction term takes just the same form as for an application term – one just applies the swap to all immediate subtrees:

$$\begin{aligned}(a\, b)\cdot(\lambda c.t) &= \lambda((a\, b)\cdot c).((a\, b)\cdot t)\\ (a\, b)\cdot(t\, t') &= ((a\, b)\cdot t)((a\, b)\cdot t')\\ \text{where } (a\, b)\cdot c &= \begin{cases} a & \text{if } c = b\\ b & \text{if } c = a\\ c & \text{otherwise}\end{cases}\end{aligned}$$

Proposition 2.2 of [17] proves that the relation $\sim_\alpha$ can be inductively generated by syntax-directed rules of the following three kinds:

$$\frac{}{a \sim_\alpha a} \qquad (2)$$

$$\frac{t_1 \sim_\alpha t_1' \qquad t_2 \sim_\alpha t_2'}{t_1\, t_2 \sim_\alpha t_1'\, t_2'} \tag{3}$$

$$\frac{(a\,b)\cdot t \sim_\alpha (a'\,b)\cdot t'}{\lambda a.t \sim_\alpha \lambda a'.t'}\; b \neq a, a' \text{ and } b \text{ does not occur in } t \text{ or } t' \tag{4}$$

It is immediate from this characterisation of $\sim_\alpha$ that it is equivariant, in the sense that

$$\text{for all } t \text{ and } t', \quad t \sim_\alpha t' \quad \text{implies} \quad (a\,b)\cdot t \sim_\alpha (a\,b)\cdot t'$$

(a property that we have noted does not hold for the renaming operation $\{b/a\}(-)$). The reason why the equivariance property holds is quite general: *any relation inductively defined by an* equivariant *set of rules* (in the sense that swapping a pair of names throughout the hypotheses and conclusion of any rule yields another element of the set of rules) *is easily seen to be an equivariant relation, i.e., closed under applying the swapping operation.* And as we mentioned above, once we know that $\sim_\alpha$ is equivariant, so will be predicates built up from it using the usual logical operations. To illustrate the usefulness of this observation, consider proving from the above inductive characterisation of $\sim_\alpha$ that it is transitive. We can proceed by 'rule induction' and show that the relation

$$\varphi(t, t') \quad \triangleq \quad (\forall t'')\; t' \sim_\alpha t'' \Rightarrow t \sim_\alpha t'' \tag{5}$$

is closed under the rules (2)–(4) inductively defining $\sim_\alpha$. We will just consider the case of the third rule, since it illustrates the usefulness of equivariance.

So suppose we have

$$\varphi((a\,b)\cdot t, (a'\,b)\cdot t') \tag{6}$$

where $b \neq a, a'$ and b does not occur in t or t'. We have to show that $\varphi(\lambda a.t, \lambda a'.t')$ holds, i.e., that for any t'', $\lambda a'.t' \sim_\alpha t''$ implies $\lambda a.t \sim_\alpha t''$. Now the syntax-directed nature of the rules comes to our aid: if $\lambda a'.t' \sim_\alpha t''$ holds, it must have been deduced by an application of rule (4): so $t'' = \lambda a''.t'''$ say, and

$$(a'\,c)\cdot t' \sim_\alpha (a''\,c)\cdot t''' \tag{7}$$

holds for some $c \neq a', a''$ with c not occurring in t' or t'''. Let d be a *fresh* variable, i.e., one not occurring in t, t', or t'' and not in $\{a, a', a'', b, c\}$. Now we use the equivariance property of φ: since (6) holds, so does the predicate with b and d swapped throughout; and since b and d do not occur in t or t' and are not equal to a or a', the result of this swapping is provably equivalent to

$$\varphi((a\,d)\cdot t, (a'\,d)\cdot t') \tag{8}$$

Similarly, the equivariance property of $\sim_\alpha$ itself means that by swapping c and d in (7), we also have

$$(a'\,d)\cdot t' \sim_\alpha (a''\,d)\cdot t''' \tag{9}$$

Remembering the definition of φ, (8) and (9) combine to yield

$$(a\,d)\cdot t \sim_\alpha (a''\,d)\cdot t'''$$

and since $d \neq a, a''$ and d does not occur in t or t'', we can apply rule (4) to this to deduce $\lambda a.t \sim_\alpha \lambda a'''.t'''$, i.e., $\lambda a.t \sim_\alpha t''$, as required. □

Example 2 (*Weakening in type theory*)**.** McKinna and Pollack [28] note that in the naïve approach to named bound variables referred to in Section 1, there is a difficulty with proving the weakening property of type systems by rule induction. For example, consider the usual typing relation assigning simple types to terms of the untyped λ-calculus. As in the previous example, we take the latter to mean α-equivalence classes $[t]_\alpha$ of parse trees t given by the grammar (1). The typing relation takes the form $\Gamma \vdash [t]_\alpha : \tau$, where types τ are given by the grammar $\tau ::= X \mid \tau \to \tau$ (with X ranging over an infinite collection of type variables); and where the typing context Γ is a finite partial function from variables to types. The typing relation is inductively generated by rules following the structure of the parse tree t. (If the reader is not familiar with these rules, see [19, Chapter 2], for example; but note that as mentioned in Section 1, the literature usually does not bother to make a notational distinction between t and $[t]_\alpha$.)

When trying to prove the weakening property of the typing relation, namely

$$(\forall \Gamma)(\forall t)(\forall \tau)\ \Gamma \vdash [t]_\alpha : \tau \ \Rightarrow\ (\forall \tau')(\forall a' \notin \mathrm{dom}(\Gamma))\ \Gamma, a' : \tau' \vdash [t]_\alpha : \tau \tag{10}$$

it is natural to try to proceed by rule induction and show that the predicate $\varphi(\Gamma, [t]_\alpha, \tau)$ given by

$$(\forall \tau')(\forall a' \notin \mathrm{dom}(\Gamma))\ \Gamma, a' : \tau' \vdash [t]_\alpha : \tau$$

defines a relation that is closed under the rules inductively defining the typing relation and hence contains that relation. But the induction step for the rule for typing λ-abstractions

$$\frac{\Gamma, a : \tau_1 \vdash [t]_\alpha : \tau_2}{\Gamma \vdash [\lambda a.t]_\alpha : \tau_1 \to \tau_2}\ a \notin \mathrm{dom}(\Gamma) \tag{11}$$

is problematic: we have to prove

$$\varphi(\Gamma, a : \tau_1,\ [t]_\alpha,\ \tau_2) \wedge a \notin \mathrm{dom}(\Gamma) \ \Rightarrow\ \varphi(\Gamma,\ [\lambda a.t]_\alpha,\ \tau_1 \to \tau_2);$$

i.e., given

$$\varphi(\Gamma, a : \tau_1,\ [t]_\alpha,\ \tau_2) \tag{12}$$

and

$$a \notin \mathrm{dom}(\Gamma) \tag{13}$$

we have to prove that

$$\Gamma, a' : \tau' \vdash [\lambda a.t]_\alpha : \tau_1 \to \tau_2 \tag{14}$$

holds for *all* $a' \notin \mathrm{dom}(\Gamma)$ (and all τ') – and there is a problem with doing this for the case $a' = a$.

But this difficulty with the induction step is easily circumvented if we take equivariance into account. The axioms and rules defining typing are closed under the operations of swapping pairs of variables (and also under swapping pairs of type variables, but we do not need to use that here). For example, if we have an instance of rule (11) and we swap any pair of variables throughout both the hypotheses and the conclusion, we get another valid instance of this rule.[1] As we mentioned in the previous example, it follows from this swapping property of the axioms and rules that the typing relation, being the least relation closed under the axioms and rules, is also closed under the swapping operations. Therefore any assertion about typing that we make by combining the typing relation with other such equivariant predicates (such as '$a \in \mathrm{dom}(\Gamma)$') using the usual logical connectives and quantifiers will be equivariant. In particular the predicate φ defined above is equivariant. Thus if we know that (12) holds, then so does

[1] To see this, strictly speaking we have to make use of the fact, noted in Example 1, that $(a\,b)\cdot(-)$ preserves $\sim_\alpha$ and hence that the result of swapping a and b throughout the set $[t]_\alpha$ is the equivalence class $[(a\,b)\cdot t]_\alpha$.

$\varphi(\Gamma, b : \tau_1,\ [(a\,b)\cdot t]_\alpha,\ \tau_2)$ for any *fresh* variable b (i.e., one not occurring in Γ, t, or $\{a, a'\}$). So by definition of φ, since $a' \notin \mathrm{dom}(\Gamma, b : \tau_1)$, we have $(\Gamma, b : \tau_1), a' : \tau' \vdash [(a\,b)\cdot t]_\alpha : \tau_2$. Since $(\Gamma, b : \tau_1), a' : \tau' = (\Gamma, a' : \tau'), b : \tau_1$ (we are using partial functions for typing contexts) and $b \notin \mathrm{dom}(\Gamma, a' : \tau')$ (by choice of b), we can apply typing rule (11) to conclude that $\Gamma, a' : \tau' \vdash [\lambda b.((a\,b)\cdot t)]_\alpha : \tau_1 \to \tau_2$. But $\lambda b.((a\,b)\cdot t)$ and $\lambda a.t$ are α-equivalent parse trees, so $\Gamma, a' : \tau' \vdash [\lambda a.t]_\alpha : \tau_1 \to \tau_2$ holds. Thus if (12) and (13) hold, so does $\varphi(\Gamma,\ [\lambda a.t]_\alpha,\ \tau_1 \to \tau_2)$ and we have completed the induction step. $\square$

From the considerations of this section we abstract the following ingredients for a language to describe syntax involving names and binding: the language should contain a notion of atom together with operations for swapping atoms in expressions (in general we may need several different sorts of atoms – for example, atoms for variables and atoms for type variables in Example 2); and the formulas of the language should all be equivariant with respect to these swapping operations. Atoms and swapping are two of the three novelties of Nominal Logic. The third has to do with the crucial step in the proofs in Examples 1 and 2 when we chose a *fresh* variable (d in the first example and b in the second one): we need to give a freshness relation between atoms and expressions with sufficient properties to make such arguments go through.

3. Syntax of swapping and freshness

The syntax of Nominal Logic is that of many-sorted first-order logic with equality, augmented by the following extra features.

- The collection of sorts (typical symbol S) is partitioned into two kinds: *sorts of atoms* (typical symbol A) and *sorts of data*.
- For each sort of atoms A and each sort S there is a distinguished function symbol of arity $A, A, S \longrightarrow S$ whose effect on terms $t_1 : A$, $t_2 : A$ and $t_3 : S$ we write as the term $(t_1\,t_2)\cdot t_3$ and pronounce '*swap t_1 and t_2 in t_3*'.
- For each sort of atoms A and each sort S there is a distinguished relation symbol of arity A, S whose effect on terms $t_1 : A$ and $t_2 : S$ we write as the formula $t_1 \# t_2$ and pronounce '*t_1 is fresh for t_2*'.

Later on we will add extra syntax for *freshness quantification* (Section 6) and *atom-abstraction sorts and terms* (Section 7). These extra concepts are first-order definable in terms of the basic ones given above, so we stick with these for the moment for simplicity's sake.

Just as for ordinary first-order logic, a *theory* in Nominal Logic is specified by a *signature* of sort, function and relation symbols, together with a collection of (non-logical) *axioms*, which are first-order formulas built up in the usual way from variables and the symbols of the signature, but now of course possibly using the swapping functions and the freshness relation. Here is an example of how this language of Nominal Logic can be used; we formalise some familiar concepts from λ-calculus in it. Exploring the logical properties of these formalisations has to wait until we introduce the axioms of Nominal Logic in Section 5.

Example 3 (*α, β, and η equivalence*)**.** Consider the following signature for λ-calculus, with a sort of atoms for variables and a sort of data for λ-terms over those variables.
Sort of atoms: *Var*
Sort of data: *Term*

Function symbols: $\mathit{var} : \mathit{Var} \longrightarrow \mathit{Term}$
$\mathit{app} : \mathit{Term}, \mathit{Term} \longrightarrow \mathit{Term}$
$\mathit{lam} : \mathit{Var}, \mathit{Term} \longrightarrow \mathit{Term}$

As discussed in Example 1, we can use swapping and freshness to express α-conversion:

$$(\forall a, a' : \mathit{Var})(\forall t : \mathit{Term})\ a' \# t \ \Rightarrow\ \mathit{lam}(a, t) = \mathit{lam}(a', (a\, a')\cdot t) \tag{15}$$

Instead of axiomatising α-conversion on a theory-by-theory basis, in Section 7 we move it into the logical infrastructure via a notion of atom-abstraction. In particular, we can then take *lam* to be a function symbol of arity $[\mathit{Var}]\mathit{Term} \longrightarrow \mathit{Term}$, where $[\mathit{Var}]\mathit{Term}$ is a sort of atom-abstractions (see Definition 4), whose logical properties ensure that extra axioms for α-conversion like (15) are no longer necessary.

Another typical use of the freshness relation # is to internalise the usual side-condition on η-conversion, as in the following axiom:

$$(\forall a : \mathit{Var})(\forall t : \mathit{Term})\ a \# t \ \Rightarrow\ t = \mathit{lam}(a, \mathit{app}(t, \mathit{var}(a))) \tag{16}$$

How may we express β-conversion in this language? One way is to augment the signature with a function symbol for capture-avoiding substitution

$$\mathit{subst} : \mathit{Term}, \mathit{Var}, \mathit{Term} \longrightarrow \mathit{Term}$$

and then express β-conversion by

$$(\forall a : \mathit{Var})(\forall t, t' : \mathit{Term})\ \mathit{app}(\mathit{lam}(a, t'), t) = \mathit{subst}(t, a, t') \tag{17}$$

together with axioms for substitution:

$$(\forall t : \mathit{Term})(\forall a : \mathit{Var})\ \mathit{subst}(t, a, \mathit{var}(a)) = t \tag{18}$$

$$(\forall t : \mathit{Term})(\forall a, a' : \mathit{Var})\ \neg\, a = a' \ \Rightarrow\ \mathit{subst}(t, a, \mathit{var}(a')) = \mathit{var}(a') \tag{19}$$

$$\begin{aligned}(\forall t, t', t'' : \mathit{Term})(\forall a : \mathit{Var})\ &\mathit{subst}(t, a, \mathit{app}(t', t''))\\ &= \mathit{app}(\mathit{subst}(t, a, t'), \mathit{subst}(t, a, t''))\end{aligned} \tag{20}$$

$$\begin{aligned}(\forall t, t' : \mathit{Term})\,(\forall a, a' : \mathit{Var})\ &\neg\, a' = a \wedge a' \# t \ \Rightarrow\\ &\mathit{subst}(t, a, \mathit{lam}(a', t')) = \mathit{lam}(a', \mathit{subst}(t, a, t'))\end{aligned} \tag{21}$$

Since the last axiom only specifies how to substitute under a λ-binder when the bound variable a' is sufficiently fresh, i.e., when $\neg\, a' = a$ and $a' \# t$, it might seem that the axioms for *subst* do not specify it uniquely. However, in view of axiom (15), any $\mathit{lam}(a', t')$ is equal to some $\mathit{lam}(a'', t'')$ for which the freshness condition is satisfied. In Section 7, we give a theory in Nominal Logic for λ-terms modulo α-equivalence (Example 6) that includes a structural induction principle codifying this familiar practice of only dealing with λ-abstractions whose bound variables are sufficiently fresh.

4. Nominal sets

As explained in [17], the Nominal Logic notions of atom, swapping and freshness can be given a meaning independent of any particular object-level syntax using *FM-sets* – the Fraenkel–Mostowski permutation model of set theory. Here we give a simplified, but essentially equivalent, presentation of

FM-sets that emphasises swapping over more general permutations of atoms. At the same time we use a mild generalisation of [17] (mentioned in [16, Section 7]) in which the set of atoms is partitioned into countably many different kinds (and we only swap atoms of the same kind).

Fix a countably infinite family $(\mathbb{A}_n \mid n \in \mathbb{N})$ of pairwise disjoint, countably infinite sets. We write $\mathbb{A}$ for the union of all the $\mathbb{A}_n$ and call its elements *atoms*.

Definition 1 (*Nominal sets*)**.** A *nominal set* X is a set $|X|$ equipped with a well-behaved notion of swapping atoms in elements of the set. By definition this means that for each element $x \in |X|$ and each pair of atoms a, a' of the same kind (i.e., $a, a' \in \mathbb{A}_n$ for some $n \in \mathbb{N}$), we are given an element $(a\,a')\cdot_X x$ of X, called *the result of swapping a and a' in x*. These swapping operations are required to have the following properties:

(i) *Equational properties of swapping*: for each $x \in |X|$ and all pairs of atoms of equal sort, $a, a' \in \mathbb{A}_m$ and $b, b' \in \mathbb{A}_n$ (any $m, n \in \mathbb{N}$)

$$(a\,a)\cdot_X x = x \tag{22}$$

$$(a\,a')\cdot_X (a\,a')\cdot_X x = x \tag{23}$$

$$(a\,a')\cdot_X (b\,b')\cdot_X x = ((a\,a')b \ \ (a\,a')b'\,)\cdot_X (a\,a')\cdot_X x \tag{24}$$

where

$$(a\,a')b \triangleq \begin{cases} a & \text{if } b = a' \\ a' & \text{if } b = a \\ b & \text{otherwise} \end{cases} \tag{25}$$

and similarly for $(a\,a')b'$.

(ii) *Finite support property*: we require that each $x \in |X|$ only involve finitely many atoms, in the sense that given x, there exists a finite subset $w \subseteq \mathbb{A}$ with the property that $(a\,a')\cdot_X x = x$ holds for all $a, a' \in \mathbb{A}_n - w$ (any $n \in \mathbb{N}$). Then it can be shown that

$$supp_X(x) \triangleq \bigcup_{n \in \mathbb{N}} \{a \in \mathbb{A}_n \mid \{a' \in \mathbb{A}_n \mid (a\,a')\cdot_X x \neq x\} \text{ is not finite}\} \tag{26}$$

is a finite set of atoms (see the proof of [17, Proposition 3.4]), which we call the *support* of x in X.

A *morphism of nominal sets*, $f : X \longrightarrow Y$, is by definition a function from the set $|X|$ to the set $|Y|$ that respects the swapping operations in the sense that

$$f((a\,a')\cdot_X x) = (a\,a')\cdot_Y f(x) \tag{27}$$

holds for all $x \in |X|$ and all atoms a, a' (of the same kind). Clearly the composition of two such functions is another such; and identity functions are morphisms. Therefore nominal sets and morphisms form a category, which we denote by $\mathcal{N}om$.

Remark 1 (*From swapping to permutations*)**.** It is a standard result of the mathematical theory of groups and group actions that the group of all permutations of the n-element set $\{1, \ldots, n\}$ is isomorphic to the group freely generated by $n-1$ symbols g_i $(i = 1, \ldots, n-1)$, subject to the identities

$$\begin{array}{ll} (g_i)^2 = id & (i < n) \\ (g_i\,g_{i+1})^3 = id & (i < n-1) \\ (g_i\,g_j)^2 = id & (j < i-1) \end{array}$$

with the generator g_i corresponding to the permutation transposing i and $i+1$. (See for example [24, Beispiel 19.7].) From this fact one can easily deduce that the group of all (kind-respecting) finite permutations of the set of atoms $\mathbb{A}$ is freely generated by the transpositions $(a\,a')$ (with a and a' of the same kind, i.e., $a, a' \in \mathbb{A}_n$ for some $n \in \mathbb{N}$), subject to the identities

$$\begin{array}{lcl}(a\,a) & = & id\\ (a\,a')(a\,a') & = & id\\ (a\,a')(b\,b') & = & ((a\,a')b\ \ (a\,a')b'\ )(a\,a')\end{array}$$

where the atoms $(a\,a')b$ and $(a\,a')b'$ are defined as in Eq. (25). It follows that if $|X|$ is a set equipped with swapping operations satisfying Eqs. (22)–(24), then these operations extend uniquely to an action of all finite permutations on elements of $|X|$. If $|X|$ also satisfies property (ii) of Definition 1, then this action extends uniquely to all (kind-respecting) permutations, finite or not; and the elements of $|X|$ have the finite support property for this action in the sense of [17, Definition 3.3]. These observations form the basis of a proof that *the category $\mathcal{N}om$ of Definition 1 is equivalent to the Schanuel topos* [17, Section 7], which underlies the universe of FM-sets used in [17].

It is not hard to see that products $X \times Y$ in the category $\mathcal{N}om$ are given simply by taking the cartesian product $\{(x, y) \mid x \in |X| \wedge y \in |Y|\}$ of underlying sets and defining the swapping operations componentwise:

$$(a\,a')\cdot_{X\times Y}(x, y) \triangleq ((a\,a')\cdot_X x, (a\,a')\cdot_Y y)$$

(Clearly (x, y) has the finiteness property in $X \times Y$ required by Definition 1(ii), because x has it in X and y has it in Y.) Similarly, the terminal object 1 in $\mathcal{N}om$ has a one-element underlying set and (necessarily) trivial swapping operations.

So we can interpret many-sorted first-order signatures in the category $\mathcal{N}om$: sorts S are interpreted as objects $[\![S]\!]$; function symbols f, of arity $S_1, \ldots, S_n \longrightarrow S$ say, as morphisms $[\![f]\!] : [\![S_1]\!] \times \cdots \times [\![S_n]\!] \longrightarrow [\![S]\!]$; and relation symbols R, of arity $S_1, \ldots, S_n$ say, as subobjects of $[\![S_1]\!] \times \cdots \times [\![S_n]\!]$. Indeed, $\mathcal{N}om$ has sufficient properties to soundly interpret classical first-order logic with equality [2] using the usual techniques of categorical logic – see [26] or [33, Section 5] for a brief overview. In fact, readers unfamiliar with such techniques need not become so just to understand the interpretation of first-order logic in the category of nominal sets, since it is just like the usual Tarskian semantics of first-order logic in the category of sets (at the same time remaining within the world of equivariant properties). For it is not hard to see that the subobjects of an object X in the category $\mathcal{N}om$ are in bijection with the subsets $A \subseteq |X|$ of the underlying set that are equivariant, in the sense that $(a\,a')\cdot_X x \in A$ whenever $x \in A$, for any atoms a, a' (of the same kind). As we mentioned in Section 2, the collection of equivariant subsets is closed under all the usual operations of first-order logic and contains equality. So it just remains to explain the interpretation in $\mathcal{N}om$ of the distinctive syntax of Nominal Logic – atoms, swapping and freshness.

Definition 2. Here is the intended interpretation of atoms, swapping and freshness in the category of nominal sets of Definition 1.

Atoms. A sort of atoms in a Nominal Logic signature is interpreted by a *nominal set of atoms A_n* (for some $n \in \mathbb{N}$), which by definition has underlying set $|A_n| = \mathbb{A}_n$ and is equipped with the swapping operations given by

[2] And much more besides, since it is equivalent to the Schanuel topos, but that will not concern us here.

$$(a\,a')\cdot b \triangleq \begin{cases} a & \text{if } b = a' \\ a' & \text{if } b = a \\ b & \text{otherwise} \end{cases}$$

(where $b \in \mathbb{A}_n$ and $a, a' \in \mathbb{A}_m$ for any $m \in \mathbb{N}$). We always assume that distinct sorts of atoms are interpreted by distinct kinds of atoms. (So we are implicitly assuming that signatures contain at most countably many such sorts.)

Swapping. Note that by virtue of Eq. (24), the function $a, a', x \mapsto (a\,a')\cdot_X x$ determines a morphism $A_n \times A_n \times X \longrightarrow X$ in the category $\mathcal{N}om$. This morphism is used to interpret the distinguished function symbol $A, A, S \longrightarrow S$ for swapping, assuming the nominal set of atoms A_n is the interpretation of the sort of atoms A and that X is the interpretation of S. Thus

$$[\![(a\,a')\cdot s]\!] = ([\![a]\!]\,[\![a']\!])\cdot_X [\![s]\!] \quad \text{when } s : S \text{ and } [\![S]\!] = X.$$

Freshness. The distinguished relation symbol # of arity A, S for freshness is interpreted as the 'not in the support of' relation $(-) \notin supp_X(-)$ between atoms and elements of nominal sets. Thus if the nominal set of atoms A_n is the interpretation of the sort of atoms A and X is the interpretation of the sort S, then for terms $a : A, s : S$, the formula $a \# s$ is satisfied by the interpretation if and only if $[\![a]\!] \notin supp_X([\![s]\!])$, where $supp_X$ is as in Eq. (26). (It is not hard to see that this is an equivariant subset of $\mathbb{A}_n \times |X|$ and hence determines a subobject of $[\![A]\!] \times [\![S]\!]$ in $\mathcal{N}om$.)

We turn next to an axiomatisation within first-order logic of properties of this nominal sets interpretation of atoms, swapping and freshness.

5. Nominal logic axioms

For simplicity, we will use a Hilbert-style presentation of Nominal Logic: a single rule of Modus Ponens, the usual axiom schemes of first-order logic with equality, plus axiom schemes for swapping and freshness. These latter are listed in Appendix A as (**S1**)–(**S3**), (**E1**)–(**E4**) and (**F1**)–(**F4**). (Appendix A also gives axioms for the freshness quantifier and atom-abstraction constructs that we consider in later sections.) Axiom scheme (**F4**) expresses within our first-order language the very important principle that *there is a sufficient supply of fresh atoms*, in the sense that it is not finitely exhaustible. The other axioms express rather anodyne properties of swapping and freshness. Indeed, the following results show that these axioms validate the properties of swapping given in Section 4 and the fundamental assumption mentioned at the start of Section 2, namely that all properties expressible in Nominal Logic are invariant under swapping atoms.

Proposition 1 (Equational properties of swapping)**.** *The equations in part* (*i*) *of Definition* 1 *are all provable in Nominal Logic*.

Proof. Eqs. (22), (23) and (24) are just axioms (**S1**), (**S2**) and (**E1**) respectively. Eq. (25) corresponds to four properties (taking into account the fact that we allow more than one sort of atoms):

$$(\forall a, a' : A)\ (a\,a')\cdot a = a' \tag{28}$$

$$(\forall a, a' : A)\ (a\,a')\cdot a' = a \tag{29}$$

$$(\forall a, a', b : A)\ \neg b = a \wedge \neg b = a' \Rightarrow (a\,a')\cdot b = b \tag{30}$$

$$(\forall a, a' : A)(\forall b : A')\ (a\,a')\cdot b = b \tag{31}$$

where A and A' are different sorts of atoms. Property (28) is just axiom (**S3**); applying $(a\,a')\cdot(-)$ to both sides of Eq. in (28) and using axiom (**S2**), we obtain (29). Property (30) follows from axioms (**F1**) and (**F2**); and property (31) from (**F1**) and (**F3**). □

Proposition 2 (Equivariance)**.** *For each term t and formula φ, with free variables amongst $\vec{x} : \vec{S}$ say, we have*

$$(\forall a, a' : A)(\forall \vec{x} : \vec{S})\ (a\,a')\cdot t(\vec{x}) = t((a\,a')\cdot\vec{x}) \tag{32}$$

$$(\forall a, a' : A)(\forall \vec{x} : \vec{S})\ \varphi(\vec{x}) \Leftrightarrow \varphi((a\,a')\cdot\vec{x}) \tag{33}$$

where $t((a\,a')\cdot\vec{x})$ denotes the result of simultaneously substituting $(a\,a')\cdot x_i$ for x_i in t (as x_i ranges over $\vec{x}$) and similarly for $\varphi((a\,a')\cdot\vec{x})$.

Proof. Property (32) follows from axioms (**E1**) and (**E3**), by induction on the structure of the term t. For (33) we proceed by induction on the structure of the formula φ, using standard properties of first-order logic for the induction steps for connectives and quantifiers. Note that by virtue of axiom (**S2**), Eq. (33) holds if and only if

$$(\forall a, a' : A)(\forall \vec{x} : \vec{S})\ \varphi(\vec{x}) \Rightarrow \varphi((a\,a')\cdot\vec{x}) \tag{34}$$

does. So the base case when φ is equality follows from the usual axioms for equality, the base case for the freshness predicate # follows from axiom (**E2**), and that for relation symbols from axiom (**E4**) (using (32) in each case). □

Theorem 1 (*Soundness*)**.** *The axioms of Nominal Logic* (*see Appendix A*) *are all satisfied by the nominal sets interpretation of atoms*, *swapping and freshness given in Definition* 2.

(At the moment we are only considering the axioms (**S1**)–(**S3**), (**E1**)–(**E4**) and (**F1**)–(**F4**) of Appendix A, but the proposition remains true for the nominal sets interpretation of the freshness quantifier and atom-abstraction given below.)

Proof. Satisfaction of axioms (**S1**)–(**S3**) and (**E1**) is guaranteed by part (i) of Definition 1 (since the swapping action for a nominal set of atoms is given by Eq. (25)). Satisfaction of axioms (**E2**) and (**F1**)–(**F3**) is a simple consequence of the definition of support in Eq. (26). Axioms (**E3**) and (**E4**) are satisfied because function and relation symbols are interpreted by morphisms and subobjects in the category of nominal sets, which have these equivariance properties. Finally, axiom (**F4**) is satisfied because the support of an element of a nominal set is a *finite* subset of the fixed, countably infinite set $\mathbb{A}$ of all atoms. □

Did we forget any axioms? In other words are the axiom schemes in Appendix A complete for the intended interpretation in the category of nominal sets? Axiom (**F4**) says that there is an inexhaustible supply of atoms that are fresh, i.e., not in the support of elements in the current context. This is certainly a consequence of property (ii) of Definition 1, which guarantees that elements of nominal sets have finite support. However, that property is ostensibly a statement of weak second order logic, since it quantifies over finite sets of atoms. So we should not expect Nominal Logic, a *first-order* theory, to completely axiomatise the notion of finite support. Example 4 confirms this expectation. Before giving it we state a useful property of freshness in Nominal Logic that we need below.

Proposition 3. *For any term t, with variables amongst the list of distinct variables $\vec{x} : \vec{S}$ say, we have*

$$(\forall a : A)(\forall \vec{x} : \vec{S})\ a \mathbin{\#} \vec{x} \Rightarrow a \mathbin{\#} t(\vec{x}) \tag{35}$$

where we write $a \mathbin{\#} \vec{x}$ for the finite conjunction of the formulas $a \mathbin{\#} x_i$ as x_i ranges over $\vec{x}$.

Proof. Given any $a : A$ and $\vec{x} : \vec{S}$, by axiom (**F4**) there is some $a' : A$ with $a' \mathbin{\#} \vec{x}$ and $a' \mathbin{\#} t(\vec{x})$. So if $a \mathbin{\#} \vec{x}$, then by axiom (**F1**) $(a\, a')\cdot x_i = x_i$ holds for each x_i. So since $a' \mathbin{\#} t(\vec{x})$ by choice of a', we have

$$\begin{aligned} a &= (a\,a')\cdot a' && \text{by axioms (S2) and (S3)} \\ &\#(a\,a')\cdot t(\vec{x}) && \text{by axiom (E2)} \\ &= t((a\,a')\cdot\vec{x}) && \text{by (32)} \\ &= t(\vec{x}) && \text{by axiom (F1)} \end{aligned}$$

as required. □

Corollary 1. *If a Nominal Logic theory contains a* closed *term $t : A$ (i.e. one with no variables*[3]*) with A a sort of atoms, then it is an inconsistent theory.*

Proof. Suppose that A is a sort of atoms and that $t : A$ is a term with no variables. By the above proposition we have $(\forall a : A)\ a \mathbin{\#} t$. Thus $t \mathbin{\#} t$ and by axiom (**F2**) this means $\neg\, t = t$, contradiction. □

Example 4 (*Incompleteness*)**.** Consider the following Nominal Logic theory.
Sort of atoms: A
Sorts of data: D, N
Function symbols: $o : N$
$s : N \longrightarrow N$
$f : D, N \longrightarrow A$
Axioms: $(\forall x : N)\ \neg\, o = s(x)$
$(\forall x, x' : N)\ s(x) = s(x') \Rightarrow x = x'$

Claim. *Any model of this theory in the category of nominal sets satisfies the formula*

$$(\forall y : D)(\exists x, x' : N)\ \neg\, x = x' \wedge f(y, x) = f(y, x') \tag{36}$$

but that formula cannot be proved in Nominal Logic from the axioms of the theory.

Proof of claim. Note that in any model of this theory in the category $\mathcal{N}om$, the interpretation of the closed terms $n_k : N$ ($k \in \mathbb{N}$) defined by

$$\begin{cases} n_0 & \triangleq o \\ n_{k+1} & \triangleq s(n_k) \end{cases}$$

[3] Since the syntax of Nominal Logic does not contain any binding constructs at the level of terms, all occurrences of variables in terms are free ones.

are distinct elements $[\![n_k]\!] \in |[\![N]\!]|$ of the nominal set $[\![N]\!]$. Therefore, to see that (36) is satisfied by the model it suffices to show for each $d \in |[\![D]\!]|$ that $[\![f]\!]([\![n_{k_1}]\!], d) = [\![f]\!]([\![n_{k_2}]\!], d) \in |[\![A]\!]|$ holds for some $k_1 \neq k_2 \in \mathbb{N}$. Note that $[\![A]\!]$ is a nominal set of atoms, A_n say. Suppose to the contrary that all the $[\![f]\!]([\![n_k]\!], d)$ are distinct atoms in $\mathbb{A}_n$. Then since the support $supp_{[\![D]\!]}(d)$ of $d \in |[\![D]\!]|$ is a finite subset of $\mathbb{A}$, we can find $k_1 \neq k_2 \in \mathbb{N}$ so that

$$a_1 \triangleq [\![f]\!]([\![n_{k_1}]\!], d) \quad \text{and} \quad a_2 \triangleq [\![f]\!]([\![n_{k_2}]\!], d)$$

satisfy $a_1, a_2 \notin supp_{[\![D]\!]}(d)$. We also have $a_1, a_2 \notin supp_{[\![N]\!]}(n_k)$ for all k (using (35) and the fact that the terms n_k are closed). Hence $a_1, a_2 \notin supp_{A_n}([\![f]\!]([\![n_k]\!]), d)$ and thus $(a_1\, a_2)\cdot_{A_n}[\![f]\!]([\![n_k]\!], d) = [\![f]\!]([\![n_k]\!], d)$, for all $k \in \mathbb{N}$. Taking $k = k_1$ and recalling the definition of a_1 and a_2, we conclude that

$$[\![f]\!]([\![n_{k_2}]\!], d) = a_2 = (a_1\, a_2)\cdot_{A_n} a_1 = (a_1\, a_2)\cdot_{A_n}[\![f]\!]([\![n_{k_1}]\!], d) = [\![f]\!]([\![n_{k_1}]\!], d)$$

with $k_1 \neq k_2$, contradicting our assumption that all the $[\![f]\!]([\![n_k]\!], d)$ are distinct.

To see that (36) is not provable in Nominal Logic it suffices to find a model, in the usual sense of first-order logic, for the general axioms of Nominal Logic and the particular axioms of this theory which does not satisfy (36). We can get such a model by modifying Definition 1 and using an *uncountable* set of atoms and sets equipped with swapping actions all of whose elements have *countable* support. More concretely, we get a model M by taking $[\![A]\!]_M$ to be an uncountable set, the set $\mathbb{R}$ of real numbers say; taking $[\![N]\!]_M$ to be a countable subset of this set, the set $\mathbb{N}$ of natural numbers say; and taking $[\![D]\!]_M$ to be the set $\mathbb{R}^{\mathbb{N}}$ of all functions from $\mathbb{N}$ to $\mathbb{R}$ (all such functions are countably supported). Define the interpretation of the function symbols o, s and f to be respectively zero, successor ($n \mapsto n + 1$) and the evaluation function $\mathbb{R}^{\mathbb{N}} \times \mathbb{N} \longrightarrow \mathbb{R}$ $(d, n \mapsto d(n))$. The interpretation of the swapping operation for sort A is as in Eq. (25) (i.e., $(r\, r')\cdot_{\mathbb{R}} r'' = (r\, r')r''$ for all $r, r', r'' \in \mathbb{R}$); for sort N, swapping is trivial (i.e., $(r\, r')\cdot_{\mathbb{N}} n = n$ for all $r, r' \in \mathbb{R}$ and $n \in \mathbb{N}$); and for sort D, it is given by $(r\, r')\cdot_{\mathbb{R}^{\mathbb{N}}} d = \lambda n \in \mathbb{N}.(r\, r')\cdot_{\mathbb{R}} d(n)$. The interpretation of the freshness predicate for sort A is $\neq$; for sort N, it is trivial (i.e., $r \# n$ holds for all $r \in \mathbb{R}$ and $n \in \mathbb{N}$); and for sort D, $r \# d$ holds if and only if $r \neq d(n)$ for all $n \in \mathbb{N}$. With these definitions one can check that all the axioms are satisfied. However (36) is not satisfied, because the inclusion of $\mathbb{N}$ into $\mathbb{R}$ gives an element $d \in \mathbb{R}^{\mathbb{N}} = [\![D]\!]_M$ for which $n \mapsto [\![f]\!]_M(d, n)$ is injective. □

Even though there is this incompleteness, it appears that the axioms of Nominal Logic are sufficient for a useful theory of names and name-binding along the lines of [17,13]. Sections 6 and 7 give some evidence for this claim. We leave to another occasion the investigation of whether the notion of ‘nominal set’ can be generalised to provide a completeness result for Nominal Logic.

6. The freshness quantifier

In this section we extend the Nominal Logic we have considered so far with a quantifier for fresh atoms. We begin by proving, within the version of Nominal Logic considered so far, the characteristic ‘some/any’ property of fresh atoms noted in [17, Proposition 4.10].

Proposition 4. *Suppose φ is a formula with free variables among the list of distinct variables $a : A, \vec{x} : \vec{S}$ (with A a sort of atoms). Then*

$$(\exists a : A)\ a \# \vec{x} \wedge \varphi(a, \vec{x}) \ \Leftrightarrow\ (\forall a : A)\ a \# \vec{x} \Rightarrow \varphi(a, \vec{x}) \tag{37}$$

is provable in Nominal Logic.

Proof. If $\varphi(a, \vec{x})$ holds, then by Proposition 2 and axiom (**S3**) we also have $\varphi(a', (a\,a')\cdot\vec{x})$; so if $a \mathbin{\#} \vec{x}$ and $a' \mathbin{\#} \vec{x}$, then axiom (**F1**) gives $\varphi(a', \vec{x})$. Thus we have the left-to-right implication in (37).

Conversely suppose $(\forall a : A)\ a \mathbin{\#} \vec{x} \Rightarrow \varphi(a, \vec{x})$ holds. For any $\vec{x} : \vec{S}$, using axiom (**F4**) we can find $a : A$ such that $a \mathbin{\#} \vec{x}$ and hence by the assumption, also satisfying $\varphi(a, \vec{x})$. $\square$

This property of freshness crops up frequently in proofs about syntax with named bound variables (see [28] for example): we choose *some* fresh name with a certain property and later on, in a wider context, we have to revise the choice to accommodate finitely many more constraints and so need to know that we could have chosen *any* fresh name with that property. For this reason it is convenient to introduce a notation that tells us we have this 'some/any' property without mentioning the context of free variables $\vec{x}$ explicitly. (Note that (37) holds for any list $\vec{x}$ of distinct variables, so long as it contains the free variables of φ other than the atom a being quantified over.)

Definition 3 (*И-quantifier*)**.** We extend the syntax of formulas with a new variable-binding operation which takes a formula φ, a sort of atoms A and a variable a of that sort and produces a formula $(\text{И}a : A)\varphi$ whose free variables are those of φ except a. We add the following axiom scheme that defines this new quantifier within first-order logic in terms of the freshness relation #:

$$((\text{И}a : A)\varphi(a, \vec{x})) \Leftrightarrow (\exists a : A)a \mathbin{\#} \vec{x} \wedge \varphi(a, \vec{x}) \qquad (\mathbf{Q})$$

where $a, \vec{x}$ is a list of distinct variables containing the free variables of φ. In view of Proposition 4 we also have

$$((\text{И}a : A)\varphi(a, \vec{x})) \Leftrightarrow (\forall a : A)\ a \mathbin{\#} \vec{x} \Rightarrow \varphi(a, \vec{x})$$

and could have used this as the axiom defining И.

Remark 2. Because of the form of axiom (**Q**), it is easy to see that the equivariance property (33) of Proposition 2 continues to hold for formulas involving the И quantifier.

Evidence for the naturalness of the И-quantifier is provided by the fact that, in the nominal sets semantics given in Section 3, it coincides with a cofiniteness quantifier. For, using the right-hand side of axiom (**Q**) to give the semantics of $(\text{И}a : A)\ \varphi$ in the category of nominal sets, we find that it holds if and only if *$\varphi(a)$ holds for all but finitely many atoms a*. See [13] for the proof of this and the development of the properties and applications of the И-quantifier within the setting of FM-set theory.

Example 5 (*α, β and η equivalence, version 2*)**.** We can re-express some of the axioms considered in Example 3 using the И-quantifier. For the α-conversion axiom (15), note that modulo the axioms of Nominal Logic it is equivalent to

$$(\forall a : \mathit{Var})(\forall t : \mathit{Term})(\forall a' : \mathit{Var})\ a' \neq a \wedge a' \mathbin{\#} t \ \Rightarrow\ \mathit{lam}(a, t) = \mathit{lam}(a', (a\,a')\cdot t).$$

So by Proposition 4, we can instead use the axiom

$$(\forall a : \mathit{Var})(\forall t : \mathit{Term})(\text{И}a' : \mathit{Var})\ \mathit{lam}(a, t) = \mathit{lam}(a', (a\,a')\cdot t) \qquad (38)$$

Similarly, using И we can re-express the η-conversion axiom (16) as

$$(\forall t : \mathit{Term})(\text{И}a : \mathit{Var})\ t = \mathit{lam}(a, \mathit{app}(t, \mathit{var}(a))) \qquad (39)$$

Finally, note that the last clause in the axiomatisation of capture-avoiding substitution, axiom (21), could be expressed as

$$\begin{aligned}&(\forall t, t' : \mathit{Term})(\forall a : \mathit{Var})(\text{И}a' : \mathit{Var})\ \mathit{subst}(t, a, \mathit{lam}(a', t'))\\ &\qquad = \mathit{lam}(a', \mathit{subst}(t, a, t'))\end{aligned} \tag{40}$$

Remark 3 (*Alternative axiomatisations of* И)**.** It follows immediately from Definition 3 and Proposition 4 that the И-quantifier satisfies

$$((\forall a : A)\ a \# \vec{x} \Rightarrow \varphi(a, \vec{x})) \Rightarrow (\text{И}a : A)\varphi(a, \vec{x}) \tag{41}$$

and

$$((\text{И}a : A)\varphi(a, \vec{x})) \Rightarrow (\exists a : A)\ a \# \vec{x} \wedge \varphi(a, \vec{x}) \tag{42}$$

when $a, \vec{x}$ is a list of distinct variables containing the free variables of φ. In fact these formulas provide an alternative axiomatisation of the И-quantifier which subsumes the crucial axiom (**F4**) asserting a sufficient supply of fresh atoms. For modulo the other axioms, one can prove (**F4**) $\wedge$ (**Q**) $\Leftrightarrow$ (41) $\wedge$ (42). We chose the presentation in terms of axioms (**F4**) and (**Q**) because the former is a principle one uses continually when reasoning with freshness in this setting and the latter makes it clear that we remain within the realm of first order logic when we use the И-quantifier in Nominal Logic.

Properties (41) and (42) suggest how to formulate introduction and elimination rules for the И-quantifier within a natural deduction formulation of Nominal Logic:

$$\frac{\Phi, a \# \vec{x} \vdash \varphi}{\Phi \vdash (\text{И}a : A)\varphi} \tag{И-intro}$$

$$\frac{\Phi \vdash (\text{И}a : A)\varphi \qquad \Phi, a \# \vec{x}, \varphi \vdash \psi}{\Phi \vdash \psi} \tag{И-elim}$$

where $\mathit{fv}(\Phi) \subseteq \vec{x}$ and $\mathit{fv}(\varphi) \subseteq a, \vec{x}$. Similarly, they suggest how to formulate right and left rules for the quantifier in a sequent calculus formulation:

$$\frac{\Phi, a \# \vec{x} \vdash \varphi, \Psi}{\Phi \vdash (\text{И}a : A)\varphi, \Psi} \tag{И-right}$$

$$\frac{\Phi, a \# \vec{x}, \varphi \vdash \Psi}{\Phi, (\text{И}a : A)\varphi \vdash \Psi} \tag{И-left}$$

where $\mathit{fv}(\Phi, \Psi) \subseteq \vec{x}$ and $\mathit{fv}(\varphi) \subseteq a, \vec{x}$. The proof theoretical properties of these formulations have yet to be explored. (However, see [4] for a sequent calculus admitting cut-elimination for a modal process logic involving the И-quantifier, in which freshness predicates like $a \# x$ appear as side-conditions rather than as formulas in sequents.)

7. Binding

In this section we extend the Nominal Logic we have considered so far to deal with variable-binding operations in a more uniform way. To motivate this, consider Example 3 once again, where the fact that *lam* is a variable-binding operation is captured by axiom (15). Instead of axiomatising the properties of such binders on a theory-by-theory basis, we endow the underlying logic, Nominal Logic, with sort- and term-forming operations for *atom-abstraction*, together with appropriate axioms. This is analogous to enriching our term language with lambda-abstraction and application in order to use functionals to represent binding operations *à la* higher-order abstract syntax [32]. However, an interesting difference here is that we are able to keep within first-order logic: atom-abstractions are merely a definitional extension within first-order logic of what we have considered so far (see Remark 4 below).

Definition 4 (*Atom-abstraction*)**.** Extend the syntax of sorts by adding a sort-forming operation that takes a sort of atoms A and a sort S and produces a new sort $[A]S$, called the *sort of* A*-atom-abstractions* of elements of sort S. Extend the syntax of terms with a new operation that takes terms $t_1 : A, t_2 : S$ and produces a term $t_1.t_2 : [A]S$. The properties of these new terms are described by the following axiom schemes.

$$(\forall b, b' : A')(\forall a : A)(\forall x : S)\ (b\,b')\cdot(a.x) = ((b\,b')\cdot a).((b\,b')\cdot x) \tag{\textbf{E5}}$$

$$(\forall a, a' : A)(\forall x, x' : S)\ a.x = a'.x' \Leftrightarrow (a = a' \wedge x = x') \vee (a' \# x\ \wedge\ x' = (a\,a')\cdot x) \tag{\textbf{A1}}$$

$$(\forall y : [A]S)(\exists a : A)(\exists x : S)\ y = a.x \tag{\textbf{A2}}$$

Axiom (**E5**) ensures that the equivariance properties of Proposition 2 (and hence also the freshness property of Proposition 3) continue to hold for the extended syntax. Axiom (**A2**) just tells us that everything of atom-abstraction sort is an atom-abstraction. The crucial axiom is (**A1**), which captures an essence of α-equivalence in terms of Nominal Logic's primitives of atom-swapping and freshness. Should not we have added axioms that explain when an atom is fresh for an atom-abstraction, to complement axioms (**F1**)–(**F4**)? In fact the following proposition shows that the freshness properties of atom-abstractions we expect from [17, Section 5] turn out to be derivable without further axioms. Thus *with these additions we have completed the definition of Nominal Logic, which is summarised in Appendix A.*

Proposition 5. *The following formulas are provable in Nominal Logic*

$$(\forall a, a' : A)(\forall x : S)\ a' \# a.x\ \Leftrightarrow\ (a' = a\ \vee\ a' \# x) \tag{43}$$

$$(\forall a : A)(\forall a' : A')(\forall x : S)\ a' \# a.x\ \Leftrightarrow\ a' \# x \tag{44}$$

where in the second formula A and A' are distinct sorts of atoms.

Proof. In view of axioms (**F2**) and (**F3**), it suffices to prove

$$(\forall a : A)(\forall x : S)\ a \# a.x \tag{45}$$

$$(\forall a : A)(\forall a' : A')(\forall x : S)\ a' \# x\ \Rightarrow\ a' \# a.x \tag{46}$$

$$(\forall a : A)(\forall a' : A')(\forall x : S)\ a' \# a\ \wedge\ a' \# a.x\ \Rightarrow\ a' \# x \tag{47}$$

for all sorts of atoms A and A' (possibly equal).

For (45), given $a : A$ and $x : S$, by axiom (**F4**) we can find $a' : A$ with $a' \# a.x$ and hence

$$\begin{aligned} a &= (a\,a')\cdot a' && \text{by axioms (S2) and (S3)}\\ &\#(a\,a')\cdot(a.x) && \text{by axiom (E2) on } a' \# a.x\\ &= a'.((a\,a')\cdot x) && \text{by axioms (E5) and (S3)}\\ &= a.x && \text{by axiom (A1).}\end{aligned}$$

For (46), given $a : A$, $a' : A'$ and $x : S$ with $a' \# x$, we argue by cases according to whether A and A' are the same and whether $a' = a$ or not. If the sorts are the same and $a' = a$, then we have $a' \# a.x$ by (45); in the other three cases we always have $a' \# a$ (using axioms (**F2**) and (**F3**)); so since $a' \# a$ and $a' \# x$, we have $a' \# a.x$ by Proposition 3 (which holds for the extended syntax by virtue of axiom (**E5**)).

For (47), given $a : A$, $a' : A'$ and $x : S$ with $a' \# a$ and $a' \# a.x$, by axiom (**F4**) we can find $a'' : A'$ with $a'' \# a$, $a'' \# x$ and $a'' \# a.x$. Then

$$\begin{aligned} a.x &= (a'\,a'')\cdot a.x && \text{by axiom (F1)}\\ &= ((a'\,a'')\cdot a).(a'\,a'')\cdot x) && \text{by axiom (E5)}\\ &= a.((a'\,a'')\cdot x) && \text{by axiom (F1)}\end{aligned}$$

and hence $x = (a'\,a'')\cdot x$ by axiom (**A1**). Since $a'' \# x$, we get $a' = (a'\,a'')\cdot a'' \# (a'\,a'')\cdot x = x$, as required. $\square$

The intended interpretation of atom-abstraction is given by the following construction on nominal sets.

Definition 5 (*Nominal set of atom-abstractions*)**.** Given a nominal set X and a nominal set of atoms A_n (cf. Definition 2), the *nominal set of atom-abstractions* $[A_n]X$ is defined as follows.

Underlying set $|[A_n]X|$ is the set of equivalence classes for the equivalence relation on $\mathbb{A}_n \times |X|$ that relates (a, x) and (a', x') if and only if $(a\,a'')\cdot_X x = (a'\,a'')\cdot_X x'$ for some (or indeed any) $a'' \in \mathbb{A}_n$ such that $a'' \notin supp_X(x) \cup supp_X(x') \cup \{a, a'\}$. We write $a.x$ for the equivalence class of the pair (a, x).

Swapping action is inherited from that for the product $A_n \times X$:

$$(b\,b')\cdot_{[A_n]X}(a.x) \triangleq a'.x', \quad \text{where } a' = (b\,b')a \text{ and } x' = (b\,b')\cdot_X x.$$

With these definitions one can check that the requirements of Definition 1 are satisfied; in particular the support of $a.x$ turns out to be the finite set $supp_X(x) - \{a\}$ (cf. Proposition 5).

Thus if a sort of atoms A gets interpreted as a nominal set of atoms $[\![A]\!]$ and a sort S gets interpreted as a nominal set $[\![S]\!]$, then the sort $[A]S$ is interpreted as the nominal set of atom-abstractions $[[\![A]\!]][\![S]\!]$. Similarly if $t_1 : A$ and $t_2 : S$, then $[\![t_1.t_2]\!] = [\![t_1]\!].[\![t_2]\!] \in [[\![A]\!]][\![S]\!]$. With these definitions, the soundness result of Theorem 1 continues to hold. To prove this one needs the following, more symmetric characterisation in Nominal Logic of equality of atom-abstractions; it matches the definition of the equivalence relation used to define $|[A_n]X|$ from $\mathbb{A}_n \times |X|$ in Definition 5 (see also the definition of $\sim_\alpha$ in Example 1).

Proposition 6. *If A is a sort of atoms and S is any sort, then the following formula is provable in Nominal Logic.*

$$(\forall a, a' : A)(\forall x, x' : S)\; a.x = a'.x' \Leftrightarrow (\text{И} a'' : A)\; (a\,a'')\cdot x = (a'\,a'')\cdot x' \tag{48}$$

Proof. First suppose $a.x = a'.x'$ holds. By axiom (**Q**), to prove $(\text{И}a'' : A)\ (a\,a'')\cdot x = (a'\,a'')\cdot x'$ we have to find some $a'' \mathrel{\#} a, a', x, x'$ such that $(a\,a'')\cdot x = (a'\,a'')\cdot x'$. By axiom (**F4**), we can certainly find a'' satisfying $a'' \mathrel{\#} a, a', x, x'$. If in addition we have $a = a'$, then by axiom (**A1**) we also have $x = x'$ and we are done. So suppose $a \neq a'$, in which case by axiom (**A1**) we also have $a' \mathrel{\#} x$ and $x' = (a\,a')\cdot x$. Hence

$$\begin{aligned}(a'\,a'')\cdot x' &= (a'\,a'')\cdot(a\,a')\cdot x \\ &= ((a'\,a'')\cdot a\ \ (a'\,a'')\cdot a')\cdot(a'\,a'')\cdot x && \text{by axiom } (\mathbf{E1}) \\ &= (a\,a'')\cdot(a'\,a'')\cdot x && \text{by Proposition 1} \\ &= (a\,a'')\cdot x && \text{by axiom } (\mathbf{F1})\end{aligned}$$

as required.

Conversely, if $(\text{И}a'' : A)\ (a\,a'')\cdot x = (a'\,a'')\cdot x'$ does hold, then by axiom (**Q**) there is some a'' with $a'' \mathrel{\#} a, a', x, x'$ and

$$(a\,a'')\cdot x = (a'\,a'')\cdot x' \tag{49}$$

If $a = a'$, then applying $(a\,a'')\cdot(-)$ to both sides of (49), by axiom (**S2**) we get

$$x = (a\,a'')\cdot(a\,a'')\cdot x = (a\,a'')\cdot(a\,a'')\cdot x' = x'$$

and hence $a.x = a'.x'$. So suppose $a \neq a'$. Applying $(a'\,a'')\cdot(-)$ to $a'' \mathrel{\#} x'$, by axioms (**E2**) and (**S3**) we get $a' \mathrel{\#} (a'\,a'')\cdot x'$ and hence by (49) that $a' \mathrel{\#} (a\,a'')\cdot x$. Hence

$$\begin{aligned}a' &= (a\,a'')\cdot a' && \text{by Proposition 1} \\ &\#(a\,a'')\cdot(a\,a'')\cdot x && \text{by axiom } (\mathbf{E2}) \\ &= x && \text{by axiom } (\mathbf{S2}).\end{aligned}$$

and then also

$$\begin{aligned}x' &= (a'\,a'')\cdot(a'\,a'')\cdot x' && \text{by axiom } (\mathbf{S2}) \\ &= (a'\,a'')\cdot(a\,a'')\cdot x && \text{by (49)} \\ &= ((a'\,a'')\cdot a\ \ (a'\,a'')\cdot a'')\cdot(a'\,a'')\cdot x && \text{by axiom } (\mathbf{E1}) \\ &= (a\,a')\cdot(a'\,a'')\cdot x && \text{by Proposition 1} \\ &= (a\,a')\cdot x && \text{by axiom } (\mathbf{F1}).\end{aligned}$$

Thus $a.x = a'.x'$ by axiom (**Q**). □

The construction in Definition 5 is used in [13,17] to treat, within the Fraenkel–Mostowski permutation model of set theory, sets of parse trees modulo α-equivalence as inductively defined sets with useful associated structural induction/recursion principles. For example, [17, Theorem 6.2] shows that the set of α-equivalence classes of λ-terms (Example 1) has an inductive characterisation as the least nominal set satisfying

$$X = A + (X \times X) + [A]X \tag{50}$$

(where A is a nominal set of atoms, $+$ indicates disjoint union and $\times$ cartesian product). From this we can derive a theory in Nominal Logic for λ-terms modulo α-equivalence, in much the same way as the inductive description of the natural numbers (namely, the least set X such that $X = 1 + X$) can lead to Peano's axioms for arithmetic.

Example 6 (*Nominal theory of λ-terms modulo α-equivalence*)**.** The signature of this theory has a sort of atoms *Var*, a sort of data *Term*, function symbols

$$\begin{aligned} var &: Var \longrightarrow Term \\ app &: Term, Term \longrightarrow Term \\ lam &: [Var]Term \longrightarrow Term \end{aligned}$$

and the following axioms.

$$(\forall a : Var)(\forall t, t' : Term)\ \neg\, var(a) = app(t, t') \tag{51}$$

$$(\forall a : Var)(\forall s : [Var]Term)\ \neg\, var(a) = lam(s) \tag{52}$$

$$(\forall s : [Var]Term)(\forall t, t' : Term)\ \neg\, lam(s) = app(t, t') \tag{53}$$

$$\begin{aligned}(\forall t : Term)\ & (\exists a : Var)\ t = var(a) \\ & \vee (\exists t', t'' : Term)\ t = app(t', t'') \\ & \vee (\exists s : [Var]Term)\ t = lam(s)\end{aligned} \tag{54}$$

$$(\forall a, a' : Var)\ var(a) = var(a') \Rightarrow a = a' \tag{55}$$

$$(\forall t, t', t'', t''' : Term)\ app(t, t') = app(t'', t''') \Rightarrow t = t'' \wedge t' = t''' \tag{56}$$

$$(\forall s, s' : [Var]Term)\ lam(s) = lam(s') \Rightarrow s = s' \tag{57}$$

$$\begin{aligned}(\forall \vec{x} : \vec{S})(\forall a : Var)\ & \varphi(var(a), \vec{x}) \\ & \wedge (\forall t, t' : Term)\ \varphi(t, \vec{x}) \wedge \varphi(t', \vec{x}) \Rightarrow \varphi(app(t, t'), \vec{x}) \\ & \wedge (\text{И} a : Var)(\forall t : Term)\ \varphi(t, \vec{x}) \Rightarrow \varphi(lam(a.t), \vec{x}) \\ & \Rightarrow (\forall t : Term)\ \varphi(t, \vec{x})\end{aligned} \tag{58}$$

Axioms (51)–(57) just state that there is a bijection (induced by *var*, *app* and *lam*) between *Term* and the disjoint union of *Var*, *Term* × *Term* and [*Var*]*Term*. The interesting axiom is the last one, (58). It is an induction principle reflecting the initiality of (50) (cf. [17, Theorem 6.8]), much as Peano's induction axiom reflects the initiality of the set of natural numbers.

To illustrate the use of axiom (58) consider adding to the theory a relation symbol *Subst* or arity *Term*, *Var*, *Term*, *Term* and the following axiom.

$$\begin{aligned} & Subst(t, a, t', t'') \Leftrightarrow \\ & \quad t' = var(a) \wedge t'' = t \\ \vee \quad & (\exists a' : Var)\ t' = var(a') \wedge \neg a' = a \wedge t'' = var(a') \\ \vee \quad & (\exists t'_1, t''_1, t'_2, t''_2 : Term)\ t' = app(t'_1, t'_2) \wedge t'' = app(t''_1, t''_2) \\ & \wedge Subst(t, a, t'_1, t''_1) \wedge Subst(t, a, t'_2, t''_2) \\ \vee \quad & (\exists a' : Var)(\exists t'_1, t''_1 : Term)\ t' = lam(a'.t'_1) \wedge t'' = lam(a'.t''_1) \\ & \wedge a' \# t \wedge Subst(t, a, t'_1, t''_1)\end{aligned} \tag{59}$$

The intention is that *Subst* is the graph of the capture-avoiding substitution function. The reason for formulating this with a relation symbol rather than a function symbol (as we did in Example 3) is to allow us to state that *Subst* is indeed the graph of a *total* function

$$(\forall t, t' : Term)(\forall a : Var)(\exists! t'' : Term)\ Subst(t, a, t', t'') \tag{60}$$

even though axiom (59) only specifies the result of substitution under a λ-binder when the bound variable a' is sufficiently fresh (i.e., when $a' \# t$ holds). Indeed, in the presence of the other axioms, (60) follows by applying the structural induction principle (58) to prove $(\forall t' : \mathit{Term})\varphi(t', t, a)$ where $\varphi(t', t, a)$ is $(\exists! t'' : \mathit{Term})\ \mathit{Subst}(t, a, t', t'')$. We omit the details.

Remark 4 (*Definability of atom-abstraction*)**.** Atom-abstraction sorts are convenient for expressing properties of binding operations, but they do not represent an essential extension of the version of Nominal Logic we presented in Section 5. The situation is analogous to the one for cartesian products, which are definable within ordinary first-order logic: given sorts S_1, S_2 and S, there is a first-order theory in all of whose models the interpretation of S is isomorphic to the cartesian product of the interpretations of S_1 and S_2. Indeed there are several such theories; for example, take a function symbol $\mathit{pair} : S_1, S_2 \longrightarrow S$ and axioms

$$(\forall x_1, x_1' : S_1)(\forall x_2, x_2' : S_2)\ \mathit{pair}(x_1, x_2) = \mathit{pair}(x_1', x_2') \Rightarrow (x_1 = x_1') \wedge (x_2 = x_2') \tag{61}$$

$$(\forall x : S)(\exists x_1 : S_1)(\exists x_2 : S_2)\ x = \mathit{pair}(x_1, x_2) \tag{62}$$

Within Nominal Logic there is a similar definability result for atom-abstraction sorts. Given sorts A, S and S' (with A a sort of atoms), and a function symbol $\mathit{abs} : A, S \longrightarrow S'$, the axioms

$$\begin{aligned}(\forall a, a' : A)\ (\forall x, x' : S)\ \mathit{abs}(a, x) = \mathit{abs}(a', x') \Leftrightarrow \\ (\reflectbox{N} a'' : A)\ (a\, a'')\cdot x = (a'\, a'')\cdot x'\end{aligned} \tag{63}$$

$$(\forall x' : S')(\exists a : A)(\exists x : S)\ x' = \mathit{abs}(a, x) \tag{64}$$

ensure that in the semantics of Section 3, the interpretation of S' is isomorphic to $[A_n]X$, where A_n and X are the nominal sets interpreting A and S respectively.

The following result shows that atom-abstraction sorts $[A]X$ have a dual nature: their elements $a.x$ embody not only the notion of abstraction as a '(bound variable, body)-pair modulo renaming the bound variable', but also the notion of abstraction as a function (albeit a partial one) from atoms to individuals (cf. Section 9).

Proposition 7. *The following formula is provable in Nominal Logic.*

$$(\forall y : [A]S)(\forall a : A)\ a \mathrel{\#} y \Rightarrow (\exists! x : S)\ y = a.x \tag{65}$$

(where $\exists!$ *means 'there exists a unique ...' and has the usual encoding in first-order logic).*

Proof. The uniqueness part of (65) follows from

$$(\forall a : A)(\forall x, x' : S)\ a.x = a.x' \Rightarrow x = x'$$

which is a corollary of axioms (**A1**) and (**S1**). For the existence part of (65), note that by Proposition 4

$$(\forall y : [A]S)(\forall a : A)\ a \mathrel{\#} y \Rightarrow (\exists x : S)\ y = a.x$$

holds if and only if

$$(\forall y : [A]S)(\exists a : A)\ a \mathrel{\#} y \wedge (\exists x : S)\ y = a.x$$

and the latter follows from axiom (**A2**) and Proposition 5 (specifically, property (45)). □

8. Choice

In informal arguments about syntax one often says things like '*choose* a fresh name such that …'. Axiom (**F4**) ensures that we can comply with such directives for Nominal Logic's formalisation of freshness. But it is important to note that *in nominal Logic such choices cannot be made uniformly in the parameters*: it is in general inconsistent with the other axioms to skolemize (**F4**) by adding function symbols $\mathit{fresh} : \vec{S} \longrightarrow A$ satisfying $(\forall \vec{x} : \vec{S})\ \mathit{fresh}(\vec{x}) \# \vec{x}$. Here is the simplest possible example of this phenomenon.

Proposition 8. *Suppose A is a sort of atoms. The formula*

$$(\forall a : A)(\exists a' : A)\ \neg\, a = a' \tag{66}$$

is a theorem of Nominal Logic. However, it is inconsistent to assume there is a function that, for each atom, picks out an atom different from it. In other words, the Nominal Logic theory with a function symbol $f : A \longrightarrow A$ and the axiom

$$(\forall a : A)\ \neg\, a = f(a) \tag{67}$$

is inconsistent.

Proof. The formula (66) is an immediate consequence of axioms (**F2**) and (**F4**). For the second part we show that $(\exists a : A)\ a = f(a)$ is a theorem. First note that by axiom (**F4**) (with the empty list of parameters $\vec{x}$), there is an atom a of sort A.[4] We show that $a = f(a)$. For any $a' : A$, by Proposition 3 we have $a' \# a \Rightarrow a' \# f(a)$, i.e., (by axiom (**F2**)) $\neg\, a' = a \Rightarrow \neg\, a' = f(a)$, i.e., $a' = f(a) \Rightarrow a' = a$. Taking a' to be $f(a)$, we get $f(a) = a$. $\square$

This phenomenon is a reflection of the fact that the category $\mathcal{N}om$ of nominal sets fails to satisfy the Axiom of Choice (see [12] for a categorical treatment of choice), which in turn reflects the fact that, by design, the Axiom of Choice fails to hold in the Fraenkel–Mostowski permutation model of set theory [25]. However, there is no problem with principles of *unique* choice (in contrast to the situation for the Theory of Contexts [23], a close cousin of Nominal Logic). For example, if a Nominal Logic theory has a model in $\mathcal{N}om$ satisfying the sentence

$$(\forall \vec{x} : \vec{S})(\exists! x' : S')\ \varphi(\vec{x}, x') \tag{68}$$

then the theory extended by a function symbol $f : \vec{S} \longrightarrow S'$ and axiom

$$(\forall \vec{x} : \vec{S})\ \varphi(\vec{x}, f(\vec{x})) \tag{69}$$

can also be modelled in $\mathcal{N}om$ (simply because in a cartesian category any subobject satisfying the properties of a single-valued and total relation is the graph of some morphism). Unfortunately a far more common situation than (68) is to have 'conditional unique existence':

$$(\forall \vec{x} : \vec{S})\ \delta(\vec{x}) \Rightarrow (\exists! x' : S')\ \varphi(\vec{x}, x') \tag{70}$$

so that $\varphi(\vec{x}, x')$ is the graph of a *partial* function with domain of definition containing those $\vec{x}$ such that $\delta(\vec{x})$. We have already seen an example of this in Proposition 7. If the formula (70) is a theorem of a Nominal Logic theory, adding a function symbol $f : \vec{S} \longrightarrow S'$ and axiom

[4] The reader can deduce at this point that the author, being of a category-theoretic bent, is not assuming a formulation of first-order logic that entails that all sorts are non-empty. Possibly empty sorts, like the empty set, have their uses!

$$(\forall \vec{x} : \vec{S})\ \delta(\vec{x}) \Rightarrow \varphi(\vec{x}, f(\vec{x})) \tag{71}$$

to the theory can result in inconsistency. This is because f represents a *total* function from $\vec{S}$ to S'. Given terms $\vec{t} : \vec{S}$, even if $\delta(\vec{t})$ does not hold and so (71) cannot be used to deduce properties of the term $f(\vec{t}) : S'$, nevertheless one may be able to use results such as Proposition 3 to deduce properties of $f(\vec{t}) : S'$ that lead to inconsistency, especially if S' happens to be a sort of atoms. The simplest possible example of this phenomenon is when $\vec{S}$ is the empty list of sorts and δ is *false*. In this case formula (70) is trivially a theorem; the skolemizing function f is a constant of sort S', so if that is a sort of atoms we get inconsistency by Corollary 1.

This difficulty with introducing notations for possibly partially defined expressions is masked in [16] by the untyped nature of FM-set theory.[5] That work introduces, among other things, a term-former for *concretion* of atom-abstractions at atoms, skolemizing the conditional unique existence formula (65) of Proposition 7. Terms involving concretion only have a definite meaning when certain preconditions are met. Nevertheless they can be given a semantics as total elements of the universe of FM-sets simply by taking their meaning when the preconditions are not met to be some default element with empty support (the empty set, say). Such a ‘hack’ is available to us in classical logic when there are enough terms of empty support. One such term is enough in an untyped setting such as FM-set theory. In a many-sorted Nominal Logic theory there is nothing to guarantee that a sort S possesses a term $t : S$ of empty support (i.e., satisfying $(\forall a : A)\ a \# t$ for all sorts of atoms A); indeed Corollary 1 shows that sorts of atoms do not possess such terms in a consistent theory. To provide Nominal Logic with a richer term language, incorporating such things as concretions of atom-abstractions at atoms and maybe more besides (such as *locally fresh atoms* – see [17, Lemma 6.3] et seq), one may be able to adapt the work of Miller [30, Section 6] on a sound treatment of skolemization in higher order logic without choice (see also [10, Section 7]). One might also consider merging Nominal Logic’s novel treatment of atoms and freshness with some conventional treatment of the logic of partial expressions (such as [2, Section VI.1] or [37]). We leave such considerations to the future and turn instead to a brief survey of existing work more directly related to the concerns of this paper.

9. Related work

One can classify work on fully formal treatments of names and binding according to the mathematical construct used to model the notion of abstraction:

- *Abstractions as (name, term)-pairs.* Here one tries to work directly with parse trees and the relation of α-equivalence between them; [28,38] are examples of work in this spirit. The drawback of this approach is not so much that many tedious details left implicit by informal practice become explicit, but rather that many of these details have to be revisited on a case-by-case basis for each object language. The use of parse trees containing de Bruijn indices [7] is more elegant; but this has its own complications and also side-steps the issue of formalising informal practice to do with named bound variables.

[5] It is also masked in the programming language FreshML sketched in [35], which has a richer term language than does Nominal Logic; this is because FreshML features unrestricted fixed point recursion in order to be Turing powerful, and hence naturally contains partially defined expressions.

- *Abstractions as functions from terms to terms.* The desire to take care of the tedious details of α-equivalence and substitution once and for all at the meta-level leads naturally to encodings of object-level syntax in a typed λ-calculus modulo α, β (and η) equivalence. This is the approach of *higher-order abstract syntax* (HOAS) [32], an idea dating back to Church and then Martin-Löf. It is well-supported by existing systems for machine-assisted reasoning based on typed λ-calculus. It does not lend itself very easily to principles of structural recursion and induction for the encoded object-language, but nevertheless such principles have been developed. For example, McDowell and Miller have developed a way of using Hallnäs' notion of partial inductive definitions [20] to enable inductive reasoning about HOAS specifications in an intuitionistic higher-order logic [27]; and Despeyroux, Pfenning and Schürmann have developed a modal typed λ-calculus that allows primitive recursive functions on HOAS-encoded object-language syntax without destroying the adequacy of the encoding [8,36].
- *Abstractions as functions from names to terms.* The *Theory of Contexts* [23] reconciles the elegance of higher-order abstract syntax with the desire to deal with names at the object-level and have relatively simple forms of structural recursion/induction. It does so by axiomatising a suitable type of names within classical higher order logic. The Theory of Contexts involves a 'non-occurrence' predicate and axioms quite similar to those for freshness in FM-set theory [17] and in the Nominal Logic presented here. However, 'non-occurrence' in [23] is dependent upon the object language, whereas our notion of freshness is a purely logical property, independent of any particular object syntax with binders. (The same remark applies to the axiomatisation of α-equivalence of λ-terms in higher order logic in [18]; and to the extension of first-order logic with binders studied in [9].) Furthermore, the use of *total* functions on names to model abstraction means that the Theory of Contexts is incompatible with the Axiom of Unique Choice (cf. Section 8), forcing the theory to have a relational rather than functional feel: see [29]. On the other hand, the Theory of Contexts is able to take advantage of existing machine-assisted infrastructure (namely Coq [6]) quite easily, whereas Gabbay had to work hard to adapt the Isabelle [31] set theory package to produce his Isabelle/FM-sets package: see [13, Chapter III] and [15].

The notion of abstraction that is definable within Nominal Logic (see Section 7) captures something of the first and third approaches mentioned above: atom-abstractions are defined to be pairs in which the name-component has been made anonymous via swapping; but we saw in Proposition 7 that atom-abstractions also behave like functions, albeit partial ones. Whatever the pros and cons of the various views of name abstraction, at least one can say that, being first-order, Nominal Logic gives a more elementary explanation of names and binding than much the work mentioned above; and a more fundamental one, I would claim, because of the independence of the notions of atoms, swapping, freshness and atom-abstraction from any particular object-level syntax involving binders.

10. Conclusion

Nominal Logic gives a first-order axiomatisation of some of the key concepts of FM-set theory – atoms, swapping and freshness – which were used in [17] to model syntax modulo α-equivalence with inductively defined sets whose structural induction/recursion properties remain close to informal practice. We have seen that, being first-order, Nominal Logic does not give a complete axiomatisation of the notion of *finite support* that underlies the notion of freshness in FM-sets. Nevertheless, the first-order properties of a notion of *freshness of names* presented in this paper seem sufficient to develop a useful theory, independent of any particular object-level language involving binders. Indeed, many of the axioms listed in Appendix A arose naturally in Gabbay's implementation of FM-set theory in the Isabelle system [13,15] as the practically useful properties of finite support. Nominal Logic and the theories we can formulate in it, are a vehicle for exhibiting those properties clearly. They are also a necessary precursor for the study of the computational properties of the logic of freshness: work is in progress on a version of first-order logic programming extended with Nominal Logic's primitives of swapping and freshness of atoms (cf. Hamana's logic programming language [21] based on the presheaf semantics of binding in [11]).

However, if one wants a single, expressive foundational theory in which to develop the mathematics of syntax in the style of this paper, one can use FM-set theory (and its automated support within Isabelle); or, as Gabbay argues in [14], a version of higher-order logic incorporating atoms, swapping and freshness.

Finally, even if one does not care about the details of Nominal Logic, I think that two simple, but important ideas underlying it are worth taking on board for the practice of operational semantics (be it with pencil-and-paper, or with machine assistance):

- *Name-swapping* $(a\,b)\cdot(-)$ *has much more convenient logical properties than renaming* $\{b/a\}(-)$.
- *The only assertions about syntax we should deal with are ones whose validity is invariant under swapping bindable names.*

Even if one only takes the naïve view of abstractions as (name, term)-pairs, it seems useful to define α-equivalence and capture-avoiding substitution in terms of name-swapping and to take account of equivariance in inductive arguments. We gave some illustrations of this in Section 2. A further example is provided by the work of Caires and Cardelli on modal logic for the spatial structure of concurrent systems [3,4]; this and the related work [5] make use of the freshness quantifier of Section 6. See also [22] for the use of permutative renaming to treat naming aspects of process calculi.

Acknowledgments

This paper is a revised and expanded version of [34] and has benefitted from the comments of René Vestergaard and the anonymous referees. The work described here draws upon joint work with Jamie Gabbay [16,17,35] and was partly funded by UK EPSRC Grant GR/R07615. I also gratefully acknowledge conversations about FM-sets and Nominal Logic with members of the Cambridge *Logic and Semantics Group*, particularly Michael Norrish, Mark Shinwell and Christian Urban.

Appendix A: Syntax and axioms of nominal logic

A.1 Signatures and sorts

A nominal logic *signature* is specified by the following data.

- A collection of ground sort symbols, partitioned into two kinds: *sorts of atoms* and *sorts of data*.
- A collection of *function symbols*, each equipped with an *arity* consisting of a list of argument sorts and a result sort (where the sorts over the signature are defined below); we write $f : \vec{S} \longrightarrow S$ to indicate that function symbol f has argument sorts given by the list $\vec{S}$ and result sort S.

- A collection of *relation symbols*, each equipped with an *arity* consisting of a list of argument sorts; we write $R <: \vec{S}$ to indicate that relation symbol R has argument sorts given by the list $\vec{S}$.

The *sorts* over a signature are built up by forming atom-abstraction sorts from the ground sort symbols:

$$\begin{array}{lll}\text{Sorts } S ::= & A & \textit{sorts of atoms}\\ & D & \textit{sorts of data}\\ & [A]S & \textit{sorts of atom-abstractions}\end{array}$$

A.2 Terms and formulas

Given a nominal logic signature, we fix mutually disjoint, countably infinite sets of variable symbols of each sort over the signature.

The *terms* over the signature are inductively defined as follows. Each well-formed term has a unique sort; we write $t : S$ to indicate that t is a term of sort S.

- $x : S$, if x is a variable symbol of sort S.
- $f(t_1, \ldots, t_n) : S$, if $f : S_1, \ldots, S_n \longrightarrow S$ and $t_1 : S_1, \ldots, t_n : S_n$.
- $(t_1\, t_2)\cdot t_3 : S$, if $t_1 : A$, $t_2 : A$ and $t_3 : S$, with A a sort of atoms and S any sort.
- $t_1.t_2 : [A]S$, if $t_1 : A$ and $t_2 : S$, with A a sort of atoms.

The *formulas* over the signature are inductively defined as follows.

- $R(t_1, \ldots, t_n)$ is a formula, if $R <: S_1, \ldots, S_n$ and $t_1 : S_1, \ldots, t_n : S_n$.
- $t_1 = t_2$ is a formula, if $t_1 : S$ and $t_2 : S$ for some sort S.
- $t_1 \# t_2$ is a formula, if $t_1 : A$ for some sort of atoms A and $t_2 : S$ for some sort S.
- $\neg\varphi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \Rightarrow \psi$ and $\phi \Leftrightarrow \psi$ are formulas, if φ and ψ are.
- $(\forall x : S)\varphi$ and $(\exists x : S)\varphi$ are formulas, if φ is a formula, where S is any sort and x is a variable symbol of sort S.
- $(\text{И} x : A)\varphi$ is a formula, if φ is a formula, where A is any sort of atoms and x is a variable symbol of sort A.

Like $\forall$ and $\exists$, the *freshness quantifier* И is a binder – the free variables of $(\text{И} x : A)\varphi$ are all the free variables of φ except x.

A.3 Axioms

A nominal logic *theory* consists of a signature and a collection of formulas over the signature, called the (non-logical) axioms of the theory. The *theorems* of the theory are all the formulas derivable using the rule of Modus Ponens from the usual axioms of first-order logic with equality augmented by the following axioms specific to nominal logic. In what follows, A and A' range over sorts of atoms, S ranges over arbitrary sorts, and $\vec{S}$ over lists of sorts.

Properties of swapping

$$(\forall a : A)(\forall x : S)\ (a\, a)\cdot x = x \tag{\textbf{S1}}$$

$$(\forall a, a' : A)(\forall x : S)\ (a\, a')\cdot(a\, a')\cdot x = x \tag{\textbf{S2}}$$

$$(\forall a, a' : A)\ (a\,a')\cdot a = a' \tag{S3}$$

Equivariance

$$(\forall a, a' : A)(\forall b, b' : A')(\forall x : S)\ (a\,a')\cdot(b\,b')\cdot x = ((a\,a')\cdot b\ \ (a\,a')\cdot b')\cdot(a\,a')\cdot x \tag{E1}$$

$$(\forall a, a' : A)(\forall b : A')(\forall x : S)\ b \mathbin{\#} x \Rightarrow (a\,a')\cdot b \mathbin{\#} (a\,a')\cdot x \tag{E2}$$

$$(\forall a, a' : A)(\forall \vec{x} : \vec{S})\ (a\,a')\cdot f(\vec{x}) = f((a\,a')\cdot\vec{x}) \tag{E3}$$

where f is a function symbol of arity $\vec{S} \longrightarrow S$ and $(a\,a')\cdot\vec{x}$ indicates the finite list of arguments given by $(a\,a')\cdot x_i$ as x_i ranges over $\vec{x}$.

$$(\forall a, a' : A)(\forall \vec{x} : \vec{S})\ R(\vec{x}) \Rightarrow R((a\,a')\cdot\vec{x}) \tag{E4}$$

where R is a relation symbol of arity $\vec{S}$.

$$(\forall b, b' : A')(\forall a : A)(\forall x : S)\ (b\,b')\cdot(a.x) = ((b\,b')\cdot a).((b\,b')\cdot x) \tag{E5}$$

Properties of freshness

$$(\forall a, a' : A)(\forall x : S)\ a \mathbin{\#} x \wedge a' \mathbin{\#} x \Rightarrow (a\,a')\cdot x = x \tag{F1}$$

$$(\forall a, a' : A)\ a \mathbin{\#} a' \Leftrightarrow \neg a = a' \tag{F2}$$

$$(\forall a : A)(\forall a' : A')\ a \mathbin{\#} a' \tag{F3}$$

where A and A' are different sorts of atoms.

$$(\forall \vec{x} : \vec{S})(\exists a : A)\ a \mathbin{\#} \vec{x} \tag{F4}$$

where $a \mathbin{\#} \vec{x}$ indicates the finite conjunction of the formulas $a \mathbin{\#} x_i$ as x_i ranges over the list $\vec{x}$.

Definition of И

$$((\text{И}a : A)\varphi) \Leftrightarrow (\exists a : A)a \mathbin{\#} \vec{x} \wedge \varphi \tag{Q}$$

where $a, \vec{x}$ is a list of distinct variables containing the free variables of φ.

Properties of atom-abstraction

$$(\forall a, a' : A)(\forall x, x' : S)\ a.x = a'.x' \Leftrightarrow x(a = a' \wedge x = x') \vee (a' \mathbin{\#} x\ \wedge\ x' = (a\,a')\cdot x) \tag{A1}$$

$$(\forall y : [A]S)(\exists a : A)(\exists x : S)\ y = a.x \tag{A2}$$

References

[1] H.P. Barendregt, The Lambda Calculus: Its Syntax and Semantics, revised ed., North-Holland, Amsterdam, 1984.

[2] M.J. Beeson, Foundations of Constructive Mathematics, Ergebnisse der Mathematik und ihrer Grenzgebeite, Springer, Berlin, 1985.

[3] L. Caires, L. Cardelli, A spatial logic for concurrency (part I), in: N. Kobayashi, B.C. Pierce (Eds.), Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001, Sendai, Japan, October 29–31, 2001, Proceedings, Lecture Notes in Computer Science, vol. 2215, Springer, Berlin, 2001, pp. 1–38.

[4] L. Caires, L. Cardelli, A spatial logic for concurrency (part II), in: 13th International Conference on Concurrency Theory, CONCUR 2002, Proceedings, Lecture Notes in Computer Science, Springer, Berlin, 2002.

[5] L. Cardelli, A.D. Gordon, Logical properties of name restriction, in: S. Abramsky (Ed.), Typed Lambda Calculus and Applications, 5th International Conference, Lecture Notes in Computer Science, vol. 2044, Springer, Berlin, 2001.

[6] The Coq proof assistant, version 7.2, January 2002. Institut National de Recherche en Informatique et en Automatique, France. Available from <coq.inria.fr>.

[7] N.G. de Bruijn, Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem, Indag. Math. 34 (1972) 381–392.

[8] J. Despeyroux, F. Pfenning, C. Schürmann, Primitive recursion for higher-order abstract syntax, in: Typed Lambda Calculus and Applications, 3rd International Conference, Lecture Notes in Computer Science, vol. 1210, Springer, Berlin, 1997, pp. 147–163.

[9] G. Dowek, T. Hardin, C. Kirchner. A completeness theorem for an extension of first-order logic with binders, in: S. Ambler, R. Crole, A. Momigliano (Eds.), Mechanized Reasoning about Languages with Variable Binding (MERLIN 20001), Proceedings of a Workshop held in conjunction with the International Joint Conference on Automated Reasoning, IJCAR 2001, Siena, June 2001, Department of Computer Science Technical Report 2001/26, University of Leicester, 2001, pp. 49–63.

[10] G. Dowek, T. Hardin, C. Kirchner, HOL-$\lambda\sigma$: An intentional first-order expression of higher-order logic, Mathematical Structures in Computer Science 11 (1) (2001) 21–45.

[11] M.P. Fiore, G.D. Plotkin, D. Turi, Abstract Syntax and Variable Binding, 14th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press, Washington, 1999, pp. 193–202.

[12] P.J. Freyd, The axiom of choice, Journal of Pure and Applied Algebra 19 (1980) 103–125.

[13] M.J. Gabbay, A theory of inductive definitions with α-equivalence: semantics, implementation, programming language, Ph.D. thesis, University of Cambridge, 2000.

[14] M.J. Gabbay, FM-HOL, a higher-order theory of names, in: Thirty Five years of Automath, Heriot-Watt University, Edinburgh, April 2002, Informal Proceedings, 2002.

[15] M.J. Gabbay, FM techniques in Isabelle, Preprint, March 2002.

[16] M.J. Gabbay, A.M. Pitts, A new approach to abstract syntax involving binders, 14th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press, Washington, 1999, pp. 214–224.

[17] M.J. Gabbay, A.M. Pitts, A new approach to abstract syntax with variable binding, Formal Aspects of Computing 13 (2002) 341–363.

[18] A.D. Gordon, T. Melham, Five axioms of alpha-conversion, in: Theorem Proving in Higher Order Logics: 9th Interational Conference, TPHOLs'96, Lecture Notes in Computer Science, vol. 1125, Springer, Berlin, 1996, pp. 173–191.

[19] C.A. Gunter, Semantics of Programming Languages: Structures and Techniques, Foundations of Computing, MIT Press, Cambridge, 1992.

[20] L. Hallnäs, Partial inductive definitions, Theoretical Computer Science 87 (1991) 115–142.

[21] M. Hamana, A logic programming language based on binding algebras, in: N. Kobayashi, B.C. Pierce (Eds.), Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001, Sendai, Japan, October 29–31, 2001, Proceedings, Lecture Notes in Computer Science, vol. 2215, Springer, Berlin, 2001, pp. 243–262.

[22] K. Honda, Elementary structures in process theory (1): sets with renaming, Mathematical Structures in Computer Science 10 (2000) 617–663.

[23] F. Honsell, M. Miculan, I. Scagnetto, An axiomatic approach to metareasoning on nominal algebras in hoas, in: F. Orejas, P.G. Spirakis, J. van Leeuwen (Eds.), 28th International Colloquium on Automata, Languages and Programming, ICALP 2001, Crete, Greece, July 2001, Proceedings, Lecture Notes in Computer Science, vol. 2076, Springer, Heidelberg, 2001, pp. 963–978.

[24] B. Huppert, in: Endliche Gruppen I, Grundlehren Math. Wiss., vol. 134, Springer, Berlin, 1967.

[25] T.J. Jech, About the axiom of choice, in: J. Barwise (Ed.), Handbook of Mathematical Logic, North-Holland, Amsterdam, 1977, pp. 345–370.

[26] M. Makkai, G.E. Reyes, Lecture Notes in Mathematics, Springer, Berlin, 1977.

[27] R.C. McDowell, D.A. Miller, Reasoning with higher-order abstract syntax in a logical framework, ACM Transactions on Computational Logic 3 (1) (2002) 80–136.
[28] J. McKinna, R. Pollack, Some type theory and lambda calculus formalised, Journal of Automated Reasoning 23 (1999) 373–409.
[29] M. Miculan. Developing (meta)theory of lambda-calculus in the theory of contexts, in: S. Ambler, R. Crole, A. Momigliano (Eds.), Mechanized Reasoning about Languages with Variable Binding (MERLIN 20001), Proceedings of a Workshop held in conjunction with the International Joint Conference on Automated Reasoning, IJCAR 2001, Siena, June 2001, Department of Computer Science Technical Report 2001/26, University of Leicester, 2001, pp. 65–81.
[30] D.A. Miller, A compact representation of proofs, Studia Logica 46 (4) (1987) 347–370.
[31] L.C. Paulson, in: Isabelle: A Generic Theorem Prover, Lecture Notes in Computer Science, vol. 828, Springer, Berlin, 1994.
[32] F. Pfenning, C. Elliott, Higher-order abstract syntax, Proceedings ACM-SIGPLAN Conference on Programming Language Design and Implementation, ACM Press, New York, 1988, pp. 199–208.
[33] A.M. Pitts, Categorical logic, in: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science, vol. 5. Algebraic and Logical Structures, Oxford University Press, 2000 (Chapter 2).
[34] A.M. Pitts, Nominal logic: A first order theory of names and binding, in: N. Kobayashi, B.C. Pierce (Eds.), Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001, Sendai, Japan, October 29–31, 2001, Proceedings, Lecture Notes in Computer Science, vol. 2215, Springer, Berlin, 2001, pp. 219–242.
[35] A.M. Pitts, M.J. Gabbay, A metalanguage for programming with bound names modulo renaming, in: R. Backhouse, J.N. Oliveira (Eds.), Mathematics of Program Construction, 5th International Conference, MPC2000, Ponte de Lima, Portugal, July 2000, Proceedings, Lecture Notes in Computer Science, vol. 1837, Springer, Heidelberg, 2000, pp. 230–255.
[36] C. Schürmann, Automating the meta-theory of deductive systems, Ph.D. thesis, Carnegie-Mellon University, 2000.
[37] D.S. Scott, Identity and existence in intuitionistic logic, in: M.P. Fourman, C.J. Mulvey, D.S. Scott (Eds.), Applications of Sheaves, Proceedings, Durham 1977, Lecture Notes in Mathematics, vol. 753, Springer, Berlin, 1979, pp. 660–696.
[38] R. Vestergaard, J. Brotherson, A formalised first-order confluence proof for the λ-calculus using one-sorted variable names, in: A. Middeldorp (Ed.), Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Utrecht, The Netherlands, May 2001, Proceedings, Lecture Notes in Computer Science, vol. 2051, Springer, Heidelberg, 2001, pp. 306–321.