Nominal Techniques

Andrew Pitts, Computer Laboratory, University of Cambridge, UK

Programming languages abound with features making use of names in various ways. There is a mathematical foundation for the semantics of such features which uses groups of permutations of names and the notion of the *support* of an object with respect to the action of such a group. The relevance of this kind of mathematics for the semantics of names is perhaps not immediately obvious. That it is relevant and useful has emerged over the last 15 years or so in a body of work that has acquired its own name: *nominal techniques*. At the same time, the application of these techniques has broadened from semantics to computation theory in general. This article introduces the subject and is based upon a tutorial at LICS-ICALP 2015 [Pitts 2015a].

1. INTRODUCTION

What mathematical structures are appropriate for modelling *locality*? This question concerns the syntax and semantics of programming language features for restricting named resources to specific lexical scopes and hiding information about names outside their scope. Here are three examples of such features:

- local variables in Algol-like languages, `new` X `in` $\langle command\rangle$
- generativity + local declarations in ML-like languages, `let` $x =$ `ref`$\langle val\rangle$ `in` $\langle exp\rangle$
- channel-name restriction in the π-calculus, $(\nu a)\langle process\rangle$

What mathematical or logical structures are needed for the semantics of these locality constructs? Is it the same in each case? I am sure the reader can think of many similar questions, because the semantics of locality is of great importance to software science. That this is so follows partly from the need to use compositional methods when constructing large and complicated systems from more manageable parts. If the parts involve names, then one has to have mechanisms for controlling accidental (or even malicious) clashes of names when combining them into a whole. Explaining the semantics of such mechanisms involves answering a more fundamental question: what does it mean for a mathematical structure (used in the semantics of programming languages) to *depend* upon some names, or to be *independent* of some names?

A popular answer to that question involves explaining name-dependence in terms of parameterization: structures that may depend upon a name are represented by functions mapping names to name-free structures. A nice aspect of this approach is that it allows name-dependency to be subsumed by well-understood and widely applied logics of functions, such as the typed λ-calculus. A not-so-nice aspect is that it forces dependency upon names to be completely explicit: if we are dealing with structures that may depend on say 42 distinct names, then we use functions of 42 arguments. Later on we may wish to use such a structure in a context with more distinct names, but to do so we have to explicitly weaken 42-ary functions into functions of greater arity. The book-keeping involved with such explicit weakening can be very irksome.

In contrast, *nominal techniques* provide a setting in which dependence of structures upon names is implicit. They achieve this by representing name independence, rather than dependence, in terms of invariance under symmetry, as I explain next.

2. INDEPENDENCE VIA SYMMETRY INVARIANCE

Let $\mathrm{Perm}\,\mathbb{A}$ denote the set of all finite permutations of a fixed, countably infinite set $\mathbb{A}$. I will call the elements of $\mathbb{A}$ *atomic names*, or just *atoms*, and write them as $a, b, c, \ldots$. The elements of $\mathrm{Perm}\,\mathbb{A}$ are the bijections π from $\mathbb{A}$ to itself for which $\{a \in \mathbb{A} \mid \pi\, a \neq a\}$ is a finite set. The identity function $\mathrm{id}_{\mathbb{A}}$ is in $\mathrm{Perm}\,\mathbb{A}$; if π and π' are in $\mathrm{Perm}\,\mathbb{A}$, then so is their composition $\pi' \circ \pi$; and so is the inverse function π^{-1}. Therefore we get the structure of a group on $\mathrm{Perm}\,\mathbb{A}$. Among the very many things that are known about this group, let me mention that every $\pi \in \mathrm{Perm}\,\mathbb{A}$ can be factored (non-uniquely) as a composition of finitely many *transpositions*. The transposition $(a\ b)$ is the finite permutation of $\mathbb{A}$ that maps a to b, b to a and leaves all other atoms fixed.

If data is constructed in some way using names drawn from $\mathbb{A}$, then it should be possible for finite permutations $\pi \in \mathrm{Perm}\,\mathbb{A}$ to *act* upon the data by replacing occurrences of each atom $a \in \mathbb{A}$ in a datum by the corresponding atom $\pi\, a$, thereby producing another datum. For example, if the data are abstract syntax trees of untyped λ-terms using elements of $\mathbb{A}$ to name variables

$$t ::= a \mid \lambda a.t \mid t\,t \qquad (a \in \mathbb{A}) \tag{1}$$

then the action of $\pi \in \mathrm{Perm}\,\mathbb{A}$ on a tree t produces another such, $\pi \cdot t$, where

$$\begin{aligned} \pi \cdot a &= \pi\, a \\ \pi \cdot (\lambda a.t) &= \lambda(\pi\, a).(\pi \cdot t) \\ \pi \cdot (t\,t') &= (\pi \cdot t)(\pi \cdot t') \end{aligned} \tag{2}$$

As another example, consider the set $\mathrm{P}\,\mathbb{A}$ of all subsets of $\mathbb{A}$. Finite permutations $\pi \in \mathrm{Perm}\,\mathbb{A}$ act upon subsets $A \in \mathrm{P}\,\mathbb{A}$ by

$$\pi \cdot A = \{\pi\, a \mid a \in A\} \tag{3}$$

There are many other examples, from all of which one abstracts the following definition.

Definition 2.1 (*Action*). An *action* of the group $\mathrm{Perm}\,\mathbb{A}$ on a set X is a binary operation mapping each $\pi \in \mathrm{Perm}\,\mathbb{A}$ and $x \in X$ to an element $\pi \cdot x \in X$, satisfying

$$\begin{aligned} \mathrm{id}_{\mathbb{A}} \cdot x &= x \\ (\pi' \circ \pi) \cdot x &= \pi' \cdot (\pi \cdot x) \end{aligned}$$

for all $x \in X$ and $\pi, \pi' \in \mathrm{Perm}\,\mathbb{A}$.

If a set X comes equipped with such an action, then we have a way of expressing the fact that an element $x \in X$ only depends upon atoms in some particular subset $A \subseteq \mathbb{A}$ and no others: for in that case, if a permutation π leaves each $a \in A$ invariant ($\pi\, a = a$), then it should also leave x invariant ($\pi \cdot x = x$). This gives us the key concept upon which nominal techniques depend:

Definition 2.2 (*Support*). For each $A \subseteq \mathbb{A}$, write $\mathrm{Perm}_A\,\mathbb{A}$ for the subgroup of $\mathrm{Perm}\,\mathbb{A}$ consisting of finite permutations π satisfying $(\forall a \in A)\ \pi\, a = a$. Suppose $\mathrm{Perm}\,\mathbb{A}$ acts upon a set X. A set of atoms $A \subseteq \mathbb{A}$ ***supports*** an element $x \in X$ if $\pi \cdot x = x$ holds for all $\pi \in \mathrm{Perm}_A\,\mathbb{A}$. The set X together with the action $\_ \cdot \_$ is called a *nominal set* if for each $x \in X$ there is a finite set of atoms that supports it with respect to the action.

Example 2.3 (*The set Λ of λ-terms is a nominal set*). The set of syntax trees of untyped λ-terms (1) is a nominal set when equipped with the action defined in (2). For it is not to hard to see that each such tree is supported by the finite set of atoms that occur anywhere in the tree (either as variables at its leafs, or as part of a λ-binder at a node of the tree). A related and more interesting example is given by the set Λ of untyped λ-terms themselves, by which I mean α-equivalence classes of abstract syntax trees. (See Section 4 if you are unfamiliar with the notion of α-equivalence.) The action (2) respects α-equivalence and hence induces an action on Λ. One can show that with respect to that action, each λ-term is supported by its finite set of *free* variables (and this is the smallest support set); see [Pitts 2013, Section 4.1].

On the other hand, the powerset $\mathrm{P}\,\mathbb{A}$ equipped with the action defined in (3) is not a nominal set. This is because we can partition $\mathbb{A}$ into two countably infinite subsets, A_0 and A_1 say. Neither subset, as an element of $\mathrm{P}\,\mathbb{A}$, is supported by a finite set of atoms with respect to the action (3). This is because for any finite $A \subseteq \mathbb{A}$, both $A_0 - A$ and $A_1 - A$ are non-empty; so picking $a_0 \in A_0 - A$ and $a_1 \in A_1 - A$, the finite permutation $(a_0\ a_1)$ transposing a_0 and a_1 is in $\mathrm{Perm}_A\,\mathbb{A}$, but satisfies $(a_0\ a_1) \cdot A_i \neq A_i$ and hence A does not support A_i. In fact the finitely supported elements of $\mathrm{P}\,\mathbb{A}$ are precisely the subsets $A \subseteq \mathbb{A}$ that are either finite, or whose complement $\mathbb{A} - A$ is finite; see [Pitts 2013, Proposition 2.9].

Definition 2.4 (*Freshness relation*). If X is a nominal set, then an atom $a \in \mathbb{A}$ *is fresh for* an element $x \in X$ if x is supported by some finite set of atoms $A \subseteq \mathbb{A}$ with $a \notin A$. In this case we write $a \mathrel{\#} x$.

This relation of freshness is the symmetry-based notion of a mathematical structure being independent of a name mentioned in the Introduction. To appreciate its usefulness we have to see some of the theory of nominal sets. As I discuss next, there are two approaches to developing that theory.

3. SET THEORY OR CATEGORY THEORY?

The notion of support (Definition 2.2) goes back as least as far as its use in set theory by Fraenkel and Mostowski to produce a model of ZFA (axioms for sets with atoms) that does not satisfy the Axiom of Choice [Fraenkel 1922; Mostowski 1939]. The model is the universe $\mathcal{U}$ of sets containing the elements of $\mathbb{A}$ as atoms and closed under taking finitely supported subsets. More precisely it is the large nominal set that is the union $\mathcal{U} = \bigcup_{\alpha \in \mathrm{On}} \mathcal{U}_\alpha$ of the ordinal-indexed sequence of nominal sets $\mathcal{U}_\alpha$ defined as follows. The sequence starts with $\mathcal{U}_0 = \emptyset$. At successor ordinals, $\mathcal{U}_{\alpha+1}$ consists of all atoms and all subsets of $\mathcal{U}_\alpha$ that are finitely supported with respect to the action of $\mathrm{Perm}\,\mathbb{A}$ on subsets $S \subseteq \mathcal{U}_\alpha$ given by

$$\pi \cdot S = \{\pi \cdot x \mid x \in S\} \tag{4}$$

At limit ordinals, $\mathcal{U}_\lambda$ consists of all elements of any $\mathcal{U}_\alpha$ with $\alpha < \lambda$. The action of $\mathrm{Perm}\,\mathbb{A}$ on elements of $\mathcal{U}$ is given by (4) if they are sets S and by $\pi \cdot a = \pi\, a$ if they are atoms a. By construction every element has a finite support. This does not stop $\mathcal{U}$ satisfying the ZFA axioms in the same way that the usual Von Neumann cumulative hierarchy (starting from $\mathbb{A}$) satisfies them. However, as Mike Mislove explains in his Preface to this article, it does mean that $\mathcal{U}$ fails to satisfy the Axiom of Choice.

Gabbay and Pitts [2002] introduced the use of Fraenkel and Mostowski's universe of sets to develop a theory of freshness (Definition 2.4) and name abstraction (Section 5) applied to recursion and induction for data modulo α-equivalence of bound names. The survey of these kind of nominal techniques by Gabbay [2011] uses this set-theoretic approach. So does the group in Warsaw [Bojańczyk et al. 2014] who are vigorously applying more general nominal techniques to automata and computation theory (Section 7). However, I think that as well as set theory's *element-oriented* view, it is useful to use the *morphism-oriented* view afforded by category theory [MacLane 1971]. Before explaining why I think that, let me describe what are the morphisms between nominal sets.

Definition 3.1 (*Category of nominal sets*). Nominal sets are the objects of a category $\mathbf{Nom}$ whose morphisms $f : X \to Y$ are functions f from the underlying set of X to the underlying set of Y that preserve the action of finite permutations: $(\forall \pi \in \mathrm{Perm}\,\mathbb{A}, x \in X)\ f(\pi \cdot x) = \pi \cdot (f\,x)$. Such functions are often called *equivariant*. The identity function is equivariant, as is the composition of two such functions. So we get a concrete category with a forgetful functor to the category of sets.

There are at least two reasons why it is useful to develop a category-theoretic approach to some piece of mathematics:

(A) **Equivalence of categories** from different mathematical realms (or even just functors between them) can allow techniques from one realm to be transferred to another.
(B) **Universal properties** (adjoint functors) can be used to characterise a mathematical construction uniquely up to isomorphism and help to filter out its special properties from ones that are true just because of that category-theoretic characterisation.

In the case of nominal sets, regarding point (A), $\mathbf{Nom}$ has several equivalent formulations. Montanari and Pistore [2000] developed a notion of *named set* that has been used for model-checking π-calculus processes using HD-automata; it is reassuring that the category of named sets turns out to be equivalent to $\mathbf{Nom}$ [Gadducci et al. 2006; Staton 2007]. $\mathbf{Nom}$ is also equivalent to a well-understood Grothendieck topos, the *Schanuel topos*; see Pitts [2013, Section 6.3]. The Schanuel topos is characterised among Grothendieck toposes by possessing an object that is a generic model of the geometric theory [Vickers 1989] of an infinite, decidable object. Across the equivalence with $\mathbf{Nom}$, that object is simply the nominal set of names:

Definition 3.2 (*Nominal set of names*). $\mathbb{A}$ equipped with the $\mathrm{Perm}\,\mathbb{A}$ action given by application ($\pi \cdot a = \pi\,a$) is a nominal set. With respect to that action, each atom a is supported by $\{a\}$ and hence the freshness relation for this nominal set is just the *not-equal* relation ($a \# b \Leftrightarrow a \neq b$).

That $\mathbf{Nom}$ and $\mathbb{A}$ serve to classify infinite decidable objects in Grothendieck toposes is an intriguing and suggestive fact that has not, so far, directly impacted the development of nominal techniques. However, topos theory is highly developed [Johnstone 2002] and provides a lot of useful tools connected with higher-order logic and type theory to apply to nominal sets. In particular it tells us that $\mathbf{Nom}$ is a cartesian closed category. Since we need them later, let me describe products and exponentials in $\mathbf{Nom}$.

Binary products in $\mathbf{Nom}$ are created by the forgetful functor to the category of sets. Thus the underlying set of the product of X and Y is their cartesian product, $X \times Y = \{(x, y) \mid x \in X \wedge y \in Y\}$, and this has to be equipped with the action $\pi \cdot (x, y) = (\pi \cdot x, \pi \cdot y)$ in order for the product projection functions to be equivariant; and furthermore $\lambda z \in Z.\ (f\,z, g\,z) : Z \to X \times Y$ is equivariant when f and g are. With respect to this action, $(x, y) \in X \times Y$ is supported by $A \cup B$ if $x \in X$ is supported by A and $y \in Y$ is supported by B; so $X \times Y$ is a nominal set.

Turning to exponentials in $\mathbf{Nom}$, if X and Y are nominal sets, then the set Y^X of all functions from X to Y has a $\mathrm{Perm}\,\mathbb{A}$ action given by

$$\pi \cdot f = \lambda x \in X.\ \pi \cdot (f(\pi^{-1} \cdot x)) \qquad (\pi \in \mathrm{Perm}\,\mathbb{A}, f \in Y^X) \tag{5}$$

Not all functions are finitely supported with respect to this action, but if we cut down to the subset

$$X \to_{\mathrm{fs}} Y = \{f \in Y^X \mid f \text{ is finitely supported with respect to the action (5)}\} \tag{6}$$

then, together with the above action, we get a nominal set. The application function $\lambda(f,x) \in Y^X \times X.\, f\,x$ is equivariant and so gives a morphism $(X \to_{\mathrm{fs}} Y) \times X \to Y$ in $\mathbf{Nom}$. It has the universal property required of an exponential for X and Y in $\mathbf{Nom}$, because currying an equivariant function $Z \times X \to Y$ yields an equivariant function $Z \to Y^X$ that factors through the inclusion $X \to_{\mathrm{fs}} Y \subseteq Y^X$.

We know that (6) is a good notion of function space for nominal sets because, together with the application function, it satisfies the universal property of an exponential. That property not only determines it uniquely up to isomorphism, but also allows one to interpret simply typed λ-calculus in $\mathbf{Nom}$ – a standard result that is valid for any cartesian closed category [Lambek and Scott 1986]. This is a small and probably familiar illustration of point (B) above. Another example of point (B), more specific to nominal techniques, is given by the concept of *name abstraction*, described in Section 5.

Before finishing this section let me say that my answer to the question in its title is: use both approaches, as appropriate.

4. PERMUTATION BEFORE SUBSTITUTION

The relation $=_\alpha$ of *α-equivalence* between abstract syntax trees of untyped λ-terms (1) has a very simple characterisation in terms of the action (2) of $\mathrm{Perm}\,\mathbb{A}$ on such trees: it is the binary relation inductively generated by the following three rules:

$$\frac{}{a =_\alpha a} \qquad \frac{t_1 =_\alpha t_1' \qquad t_2 =_\alpha t_2'}{t_1\, t_2 =_\alpha t_1' t_2'} \qquad \frac{(b\ a)\cdot t =_\alpha (b\ a')\cdot t'}{\lambda a.t =_\alpha \lambda a'.t'}\ b \mathrel{\#} (a\, t\, a'\, t') \tag{7}$$

The side-condition on the third rule refers to the freshness relation (Definition 2.4) for the nominal set of abstract syntax trees; concretely it means that b is not equal to either a or a' and does not occur in either t or t'. In that case the tree $(b\ a)\cdot t$ obtained by swapping occurrences of b and a in t is the same as the tree obtained by substituting b for all occurrences of a in t; and up to α-equivalence that is the same as substituting b for all *free* occurrences of a. These observations form the basis of a proof that the above, non-standard definition of $=_\alpha$ coincides with the standard one that is based on name substitution rather than name permutation [Barendregt 1984, Definition 2.1.11]. Given that it is equivalent to the usual definition, why should one care about this formulation in terms of permutations?

One reason for using permutation actions is because in the theory of programming languages one deals with many different formal languages containing syntactic constructs with often quite complicated forms of name-binding and with many notions of substitution. There is a simple and uniform action of $\mathrm{Perm}\,\mathbb{A}$ on well-formed syntax trees for any particular language, one which ignores any subtleties to do with binders. (For the simple example of λ-terms, this is just the fact that a permutation is applied uniformly to all the occurrences of names in a tree, be they free, binding or bound.) So one can always define permutation action before having to worry about possibly tricky notions of α-equivalence and substitution, and then use it to define those notions, much as we did above for $=_\alpha$.

That is quite a syntactical reason for using permutations. An even more important one has to do with semantics. The mathematical structures needed for a programming language semantics are sometimes finitary (for example, an operational semantics based upon finitary inductive definitions), but more often than not involve infinitary constructs such as function spaces or powersets (certainly for denotational semantics, but also for operational semantics involving coinductive definitions). Permutation actions extend very simply to such infinitary structures. Mainly this is because permutations are bijections and hence have inverses, which can be used to simplify many constructions. We have already seen an example of this in (5), where the permutation action on functions makes essential use of inverses of permutations. Exponentials for permutation actions are relatively simple (elements of the exponential are just certain kinds of function) compared with exponentials for substitution structures, which typically involve Kripke-style definitions (world-indexed families of functions) that have their roots in the Yoneda Lemma for presheaf categories [Awodey 2010, Section 8.7].

Experience with nominal techniques over the last 15 years shows that taking name permutations to be fundamental is very fruitful for programming language theory and semantics. Here are some examples, the last one of which is considered in more detail in the following two sections.

- It is natural to give operational semantics of nominal calculi such as the π-calculus [Milner et al. 1992] using inductive definitions of nominal sets [Pitts 2013, Chapter 7]; see Gabbay [2003], Staton [2007] and Cimini et al. [2012], for example.
- Game semantics can be carried out in the category of nominal sets to produce fully abstract, compositional semantics of some of the locality constructs mentioned at the beginning of this article; see Murawski and Tzevelekos [2013], for example. More conventional denotational semantics using partially ordered structures and topology can also usefully incorporate permutation actions; see Section 8. This involves the notion of *orbit finiteness* discussed in Section 7.
- Names and the notion of name abstraction described in Section 5 have been incorporated into ML-style typed functional programming [Shinwell et al. 2003]. The patch of OCaml (caml.inria.fr/ocaml/) by Shinwell [2005] has very nice pattern-matching features for computing with data involving bound names while automatically respecting α-equivalence.
- There is a permutation-base unification theory for first-order terms involving name binding operations, introduced by Urban et al. [2004]. This kind of *nominal unification* underlies extensions of first-order logic programming to treat terms modulo α-equivalence; the αProlog language of Cheney and Urban [2008] is an example.
- Nominal unification is part of a broader topic of equational logic, and rewriting, modulo α-equivalence; see for example the paper on *Nominal Rewriting Systems* by Fernández et al. [2004], which won the PPDP most influential paper 10-year award in 2014.
- Informal practice when it comes to bound names (such as *Barendregt's variable convention* [Barendregt 1984, 2.1.13]) seems to be captured rather well by the formal recursion and induction principles that can be established for nominal inductive datatypes involving name abstraction [Pitts 2013, Chapter 8]. Evidence for this is provided by Urban and Berghofer's *Nominal Package* for Isabelle/HOL (isabelle.in.tum.de/nominal/). See for example the paper on *Nominal techniques in Isabelle/HOL* by Urban and Tasson [2005], which won a CADE Skolem Award in 2015.

5. ABSTRACTING NAMES

The third rule in (7) gives the essence of the notion of α-equivalence. It can be decoupled from the particular, inductive structure of syntax trees to yield a syntax-independent definition that makes sense for any nominal set:

Definition 5.1 (*Nominal set of name abstractions*). Given a nominal set X, there is an equivalence relation $\sim_\alpha$ on $\mathbb{A} \times X$ given by

$$(a, x) \sim_\alpha (a', x') \triangleq (\exists b \mathrel{\#} (a, x, a', x'))\ (b\ a) \cdot x = (b\ a') \cdot x' \tag{8}$$

We write $[\mathbb{A}]X$ for the quotient set $(\mathbb{A} \times X)/{\sim_\alpha}$ and $\langle a \rangle x$ for the $\sim_\alpha$-equivalence class of a pair (a, x). The relation $\sim_\alpha$ is equivariant

$$(a, x) \sim_\alpha (a', x') \Rightarrow (\pi\, a, \pi \cdot x) \sim_\alpha (\pi\, a', \pi \cdot x')$$

and therefore we get a well-defined $\mathrm{Perm}\,\mathbb{A}$ action on $[\mathbb{A}]X$ satisfying $\pi \cdot \langle a \rangle x = \langle \pi\, a \rangle (\pi \cdot x)$. If $A \subseteq \mathbb{A}$ supports x in X, then one can show that $A - \{a\}$ supports the element $\langle a \rangle x$ in $[\mathbb{A}]X$ with respect to this action [Pitts 2013, Proposition 4.5]. Therefore $[\mathbb{A}]X$ is a nominal set and the freshness relation associated with it satisfies

$$b \mathrel{\#} \langle a \rangle x \;\Leftrightarrow\; b = a \;\vee\; b \mathrel{\#} x \tag{9}$$

The mapping $X \mapsto [\mathbb{A}]X$ is the object part of a functor $\mathbf{Nom} \to \mathbf{Nom}$: if $f \in \mathbf{Nom}(X, Y)$, then we get $[\mathbb{A}]f \in \mathbf{Nom}([\mathbb{A}]X, [\mathbb{A}]Y)$ well-defined by

$$([\mathbb{A}]f)\, \langle a \rangle x \triangleq \langle a \rangle (f\, x) \tag{10}$$

and this operation on morphisms preserves composition and identities. This functor has many nice properties. In particular, as explained in the next section, it can be used to extend the usual *initial algebra semantics* of algebraic datatypes [Goguen et al. 1977] to encompass the common situation where the datatype involves constructors that bind names and one wishes to quotient data by an appropriate notion of α-equivalence.

It turns out that the nominal set $[\mathbb{A}]X$ can be characterised category-theoretically as a kind of affine linear function space from $\mathbb{A}$ to X in $\mathbf{Nom}$: it is the value at X of the right adjoint to a functor $\_ * \mathbb{A} : \mathbf{Nom} \to \mathbf{Nom}$ given by the *separated tensor product*

$$X * \mathbb{A} \triangleq \{(x, a) \in X \times \mathbb{A} \mid a \mathrel{\#} x\} \tag{11}$$

which is a nominal set when endowed with the $\mathrm{Perm}\,\mathbb{A}$ action $\pi \cdot (x, a) = (\pi \cdot x, \pi\, a)$. In other words there is a natural bijection between sets of morphisms

$$\frac{X * \mathbb{A} \to Y}{\overline{\overline{X \to [\mathbb{A}]Y}}} \tag{12}$$

mediated by the morphism

$$\_ \mathbin{@} \_ : ([\mathbb{A}]Y) * \mathbb{A} \to Y \tag{13}$$

that *concretes* a name abstraction $\langle a \rangle y$ at any atom b satisfying $b \mathrel{\#} \langle a \rangle y$ to yield the element $(\langle a \rangle y) \mathbin{@} b \triangleq (a\ b) \cdot y$. See Pitts [2013, Theorem 4.12] for a proof of this adjointness property.

The atomic nature of the nominal set $\mathbb{A}$ of names manifests itself in the fact that the right adjoint $[\mathbb{A}]\_ : \mathbf{Nom} \to \mathbf{Nom}$ itself has a right adjoint:

$$\frac{([\mathbb{A}]X) \to Y}{\overline{\overline{X \to R\,Y}}} \tag{14}$$

where

$$R\,Y \triangleq \{f \in \mathbb{A} \to_{\mathrm{fs}} Y \mid (\forall a \in \mathbb{A})\; a \mathrel{\#} f\, a\} \tag{15}$$

(see Pitts [2013, Theorem 4.13]). The adjointness property (14) gives a characterisation of functions out of nominal sets of name abstractions that is very useful for recursion and induction principles for data involving binding operations, as I explain next.

6. NOMINAL ALGEBRAIC DATATYPES

Given a category $\mathbf{C}$ and a functor $T : \mathbf{C} \to \mathbf{C}$, recall that a *$T$-algebra* is an object $D \in \mathbf{C}$ equipped with a morphism $i \in \mathbf{C}(T\,D, D)$. It is *initial* if

for any T-algebra $f \in \mathbf{C}(T\,X, X)$,
there is a unique morphism $\hat{f} \in \mathbf{C}(D, X)$
satisfying $f \circ T\,\hat{f} = \hat{f} \circ i$

$$\begin{tikzcd} T\,D \arrow[r, "T\,\hat{f}"] \arrow[d, "i"'] & T\,X \arrow[d, "f"] \\ D \arrow[r, "\hat{f}"] & X \end{tikzcd} \tag{16}$$

This universal property determines (D, i) uniquely up to isomorphism and also implies that $i : T\,D \to D$ is itself an isomorphism. Furthermore, from the universal property one can derive recursion principles for constructing morphisms out of D and induction principles for proving properties of them. We have here another illustration of point (B) in Section 3: initiality characterises D up to isomorphism; and for particular categories $\mathbf{C}$ and functors T, one may be able to develop more easily applicable forms of recursion and induction for the initial algebra.

Ordinary algebraic signatures give rise to functors T on $\mathbf{C} = \mathbf{Set}$ that are sums of products: one summand for each operation symbol in the signature and the number of factors in a product equal to the number of arguments taken by the corresponding operation symbol. For the nominal version, one replaces the category of sets by the category $\mathbf{Nom}$ and consider functors $T : \mathbf{Nom} \to \mathbf{Nom}$. *Nominal algebraic signatures* [Pitts 2013, Chapter 8] allow operation symbols that bind names in some of their argument positions; and the functor $T : \mathbf{Nom} \to \mathbf{Nom}$ corresponding to such a signature uses the name abstraction functor $[\mathbb{A}]_-$ where there is such a binding argument. For example, the nominal algebraic signature corresponding to (1) yields the functor

$$T(\_) = \mathbb{A} + (\_ \times \_) + ([\mathbb{A}]\_) : \mathbf{Nom} \to \mathbf{Nom} \tag{17}$$

For a proof of the following theorem, see Pitts [2013, Theorem 8.15].

THEOREM 6.1. *Functors $T : \mathbf{Nom} \to \mathbf{Nom}$ derived from nominal algebraic signatures always have initial algebras; and those initial algebras can be constructed as sets of abstract syntax trees for the signature quotiented by a form of α-equivalence automatically generated from the signature.* $\square$

For example, the initial algebra for (17) is the nominal set Λ of λ-terms from Example 2.3. In this case the universal property (16) gives an *iteration* principle for constructing equivariant functions $\Lambda \to X$: given $f_1 \in \mathbf{Nom}(\mathbb{A}, X)$, $f_2 \in \mathbf{Nom}(X \times X, X)$ and $f_3 \in \mathbf{Nom}([\mathbb{A}]X, X)$, there is a unique $\hat{f} \in \mathbf{Nom}(\Lambda, X)$ satisfying for all $a \in \mathbb{A}$ and $t, t' \in \Lambda$

$$\begin{aligned} \hat{f}\,a &= f_1\,a \\ \hat{f}(t\,t') &= f_2(\hat{f}\,t, \hat{f}\,t') \\ \hat{f}(\lambda a.t) &= f_3(\langle a \rangle(\hat{f}\,t)) \end{aligned}$$

Instead of supplying a function f_3 on name abstractions, in practice it is more convenient to use a function of (name,value)-pairs subject to a side-condition, the *freshness condition on binders* (18), that comes from the right adjointness property (14). At the same time one can derive a form of primitive recursion rather than iteration; and also allow implicit dependence upon parameters by moving from equivariant to finitely supported functions, that is, elements of exponential nominal sets (6). Combining all these refinements, one arrives at the following principle of *α-structural recursion for λ-terms*; see Pitts [2013, Theorem 8.17] for a proof.

THEOREM 6.2 (α-STRUCTURAL PRIMITIVE RECURSION FOR λ-TERMS). *Let Λ denote the nominal set of λ-terms. For any nominal set X and finitely supported functions $f_1 \in \mathbb{A} \to_{\mathrm{fs}} X$, $f_2 \in \Lambda \times \Lambda \times X \times X \to_{\mathrm{fs}} X$ and $f_3 \in \mathbb{A} \times \Lambda \times X \to_{\mathrm{fs}} X$ satisfying*

$$(\forall a \in \mathbb{A})\ a \mathrel{\#} (f_1, f_2, f_3) \;\Rightarrow\; (\forall t \in \Lambda, x \in X)\ a \mathrel{\#} f_3(a, t, x) \tag{18}$$

there is a unique finitely supported function $\hat{f} \in \Lambda \to_{\mathrm{fs}} X$ satisfying for all $a \in \mathbb{A}$ and $t, t' \in \Lambda$

$$\hat{f}\,a = f_1\,a \tag{19}$$

$$\hat{f}(t\,t') = f_2(t, t', \hat{f}\,t, \hat{f}\,t') \tag{20}$$

$$a \mathrel{\#} (f_1, f_2, f_3) \Rightarrow \hat{f}(\lambda a.t) = f_3(a, t, \hat{f}\,t) \tag{21}$$

□

For example, given $a' \in \mathbb{A}$ and $t' \in \Lambda$, the capture-avoiding substitution function $(\_)[t'/a'] : \Lambda \to \Lambda$ is the $\hat{f}$ for

$$\begin{aligned} f_1\,a &\triangleq \text{if } a = a' \text{ then } t' \text{ else } a \\ f_2(t_1, t_2, t_1', t_2') &\triangleq t_1'\,t_2' \\ f_3(a, t_1, t_1') &\triangleq \lambda a.t_1' \end{aligned}$$

for which condition (18) holds, because $a \mathrel{\#} \lambda a.t_1'$. See Pitts [2006] for further examples. That paper shows that the above notion of α-structural primitive recursion generalizes smoothly from λ-terms to any nominal algebraic signature, giving a version of Gödel's System T for nominal data types. Urban and Berghofer's Nominal package for Isabelle/HOL (isabelle.in.tum.de/nominal/) implements this, and more. It seems to be a convenient formalisation of informal computational and proving practice to do with identifying terms up to renaming of bound names.

Finally, I should mention that nominal techniques can usefully apply not only to algebraic datatypes, but dually to coinductive datatypes; see Kurz et al. [2013], for example.

7. FINITENESS MODULO SYMMETRY

Using symmetries to reduce the size of finite state spaces is an important technique for software verification; see for example E. M. Clarke and Peled [2000, chapter 14]. For infinite state spaces, quotienting by symmetry may allow one to regain finiteness. When the symmetries are given by name permutations, one has the following notion of finiteness modulo symmetry:

Definition 7.1 (*Orbit-finiteness*). For each set X equipped with a $\mathrm{Perm}\,\mathbb{A}$-action (Defintion 2.1), consider the equivalence relation $\sim$ on the set X given by

$$x \sim y \;\triangleq\; (\exists \pi \in \mathrm{Perm}\,\mathbb{A})\ \pi \cdot x = y \tag{22}$$

The $\sim$-equivalence classes are called the *orbits* of X and X is said to be *orbit-finite* if there are only finitely many of them. We extend the definition to subsets and say that a subset $S \subseteq X$ is orbit-finite if it is both finitely supported (with respect to the $\mathrm{Perm}\,\mathbb{A}$-action (4) on subsets) and contained in a finite union of orbits of X. In this way one can consider sets in Fraenkel and Mostowski's cumulative hierarchy $\mathcal{U}$ (section 3) that are orbit-finite in this sense. This notion of finiteness in $\mathcal{U}$ plays a central role in the Warsaw group's development of nominal computation theory [Bojańczyk et al. 2012; Bojańczyk et al. 2013; Bojańczyk et al. 2014].

For example, for each $n \in \mathbb{N}$, the nominal set $\mathbb{A}^n$ of ordered n-tuples of names is infinite, but is orbit-finite; there are as many orbits as there are equivalence relations on the finite ordinal $\{0, \ldots, n-1\}$. On the other hand the nominal set $\mathbb{A}^*$ of finite lists of names is not orbit-finite, since the elements of any orbit must be lists of equal length.

Orbit-finiteness underlies the applications of nominal techniques to automata over infinite alphabets. The work of Montanari and Pistore [2000] on HD-automata and model-checking for π-calculus processes pioneered the idea using the formalism of named sets, which turns out to be equivalent to nominal sets [Gadducci et al. 2006; Staton 2007], but better adapted for giving finite presentations. Orbit-finiteness is implicit in some of the work of Tzevelekos [2011] on *fresh register automata*, an automata-theoretic model of computation with fresh-name generation (which includes the finite-memory automata of Kaminski and Francez [1994]). Studying languages of traces for that form of computation led Gabbay and Ciancia [2011] to consider Kleene algebra in this context; and this has recently been extended with a coalgebraic perspective [Kozen et al. 2015]. As I mentioned above, orbit-finiteness is central to the work of the Warsaw group, which investigates what happens if one does computation theory inside the Fraenkel-Mostowski universe replacing finite-state notions by orbit-finite ones. (In fact they do more, by considering different notions of symmetry, as described in Section 9.) This approach to computation theory is possible because orbit-finite subsets can be presented in terms of finite information:

THEOREM 7.2. *Given a nominal set X, every orbit-finite subset of X is equal to*

$$\mathrm{hull}_A\, F \;\triangleq\; \{\pi \cdot x \mid \pi \in \mathrm{Perm}_A\, \mathbb{A} \wedge x \in F\} \tag{23}$$

for some finite subsets $A \subseteq \mathbb{A}$ and $F \subseteq X$. (Recall from Definition 2.2 that $\mathrm{Perm}_A\, \mathbb{A}$ denotes the subgroup of $\mathrm{Perm}\, \mathbb{A}$ consisting of permutations π satisfying $\pi\, a = a$ for all $a \in A$.) $\square$

This presentation, which enables calculation with orbit-finite subsets, was discovered independently by Turner [2009], Gabbay [2009] and Bojańczyk et al. [2012] (in the last case, in the greater generality considered in Section 9); for a proof see Pitts [2013, Proposition 5.25].

Orbit-finite subsets generally have good closure properties, the notable exception being lack of closure under powerset, which leads to a divergence between deterministic and non-deterministic forms of nominal computation compared with the classical situation [Bojańczyk et al. 2013].

8. DOMAIN THEORY

Orbit-finite subsets also play an important role in the nominal version of Scott-Plotkin style domains and denotational semantics. Turner and Winskel [2009] note that a poset $(X, \sqsubseteq)$ in $\mathbf{Nom}$ has joins for all finitely supported chains iff it is *uniform-directed complete*, which by definition means that it has joins for all directed subsets all of whose elements are supported by a common finite subset of $\mathbb{A}$. Lösch and Pitts [2014] prove that the subsets which are compact with respect to uniform-directed joins are precisely the orbit-finite ones, thus explaining the central position of orbit-finiteness in this nominal domain theory.

There is a well-behaved domain theory based upon uniform-directed complete nominal posets, including recursively defined domains constructed along traditional lines [Abramsky and Jung 1994]; see Pitts [2013, Chapter 11]. In this setting one has an operation on domains corresponding to name abstraction (Definition 5.1) and this can be used to give denotational semantics for languages with features involving local names. Turner and Winskel [2009] apply this to nominal process calculi. The earlier work of Shinwell and Pitts [2005] gives a denotational semantics for FreshML [Shinwell et al. 2003]. One tricky aspect of the name-abstraction domain construct is that the name concretion operation (13) can be discontinuous; see Pitts [2013, Section 11.2]. However, it is continuous if the domain comes equipped with a notion of name restriction [Pitts 2013, Chapter 9]; this is automatically the case for domains of continuations, which accounts for the continuation-passing style of denotational semantics employed by Shinwell and Pitts [2005]. Domains with name restriction feature in the work of Lösch [2014] and Lösch and Pitts [2014], whose main result is a full abstraction theorem in the style of Plotkin [1977] for the PCF language extended with a form of local scoping for names due to Odersky [1994].

This nominal domain theory should support a version of Scott's information systems [Scott 1982] and a *domain theory in logical form* [Abramsky 1991], but as far as I know this has not been investigated. In this respect the work of Gabbay et al. [2011] on Stone duality for nominal Boolean algebras with a freshness quantifier should be relevant.

9. DIFFERENT SYMMETRIES

Recall from Definition 2.2 the central concept of nominal techniques: the notion of *support*. The concept still makes sense, and is very useful, if one replaces the group $\mathrm{Perm}\,\mathbb{A}$ by some subgroup G. In other words, one considers a more restrictive notion of symmetry than that given by arbitrary (finite) permutations of names. Bojańczyk et al. [2012] consider three interesting examples:

- **Equality**: G is the whole of $\mathrm{Perm}\,\mathbb{A}$. This is the case we have considered so far. It is called the *equality* symmetry because the only relations preserved by all permutations of $\mathbb{A}$ are equality and its complement. A variation on this symmetry is given by partitioning $\mathbb{A}$ into countably many countably infinite subsets and taking G to be the subgroup of $\mathrm{Perm}\,\mathbb{A}$ consisting of permutations that map each subset back into itself. This gives a version of nominal sets with many sorts of names, which is useful for applications to programming language semantics, where one usually has to consider several different syntactic categories of names.
- **Linear order**: identify the countably infinite set of names $\mathbb{A}$ with the set of rational numbers and take G to be all order-preserving permutations.
- **Graphs**: take $\mathbb{A}$ to be the vertices of the Rado graph [Rado 1964] and G to be the group of automorphisms of that graph.

The general pattern, of which these three are instances, is to take $\mathbb{A}$ to be the carrier of the universal homogeneous structure (Fraïssé limit) for a finite relational signature and G to be the group of automorphisms with respect to the signature. The category of finitely supported G-sets and equivariant functions (or if you prefer, the universe of hereditarily finitely supported sets with respect to this symmetry) appears to have many of the good properties enjoyed by $\mathbf{Nom}$ when it comes to nominal automata theory [Bojańczyk et al. 2014]. The extent to which it has forms of name abstraction like those considered in Section 5 and Section 6 (and hence can be used to model locality constructs when computing modulo G-symmetry) remains to be investigated.

10. CONSTRUCTIVE TYPE THEORY

Urban and Berghofer's *Nominal Package* (isabelle.in.tum.de/nominal/) depends, in part, upon a formalization of some of the mathematics mentioned in Sections 5 and 6 within the Isabelle/HOL implementation of Church's typed higher-order logic. It would be useful to provide similar facilities for theorem-proving systems such as Coq (coq.inria.fr) and Agda (wiki.portal.chalmers.se/agda) that are based upon Martin-Löf's constructive type theory and enhancements of it such as the Calculus of Inductive

Constructions. To do so requires a suitably constructive version of the theory of nominal sets. At least one way in which the material in Pitts [2013] is not constructive is that it makes extensive use of the support function supp_X that assigns to each element x of a nominal set X the *smallest* finite set $\mathrm{supp}_X\, x$ of atoms that supports it. The definition of nominal set (Definition 2.2) asks that every $x \in X$ possesses some finite set of atoms. From the existence of some such finite set one can prove the existence of a smallest one; see [Pitts 2013, (2.4)]. However, as Swan [2014, Section 1.2.1] points out, the validity of that result in a constructive setting relies upon X having decidable equality and it may fail when that is not the case. It seems that much of the theory of nominal sets can be developed without relying upon the existence of smallest supports (with the possible exception of the generalised form of name abstraction [Pitts 2013, Section 4.6] that has proved useful in Fresh OCaml [Shinwell 2003]). There have been some experiments with constructive versions of nominal sets within Coq [Aydemir et al. 2007] and Agda [www.cl.cam.ac.uk/~amp12/agda/choudhury/html/], but the analogue of the Isabelle/HOL *Nominal Package* is still lacking.

Instead of developing the theory of nominal sets within an existing constructive type theory, one can consider enhancing the type theory with features that are, more or less, inspired by properties of dependent families of nominal sets. Like any topos, the category of nominal sets models the extensional version of Martin-Löf's constructive type theory. Furthermore, its characteristically nominal features (such as name abstraction and the freshness quantifier [Pitts 2013, Section 3.2]) have interesting, dependently-typed versions. This was first investigated by Schöpp and Stark [Schöpp and Stark 2004; Schöpp 2006] and more recently using the formalism of *categories with families* [Dybjer 1996] by Pitts et al. [2015]. There are several proposals for nominal dependent type theories in the literature. Some are closely linked to properties of families in **Nom** [Cheney 2012; Pitts et al. 2015], some loosely so [Fairweather et al. 2015] and some are more operationally motivated [Westbrook et al. 2009]. It remains to be seen which, if any, will be useful in practice.

Finally, nominal techniques have recently been found useful in the study of Homotopy Type Theory [Univalent Foundations Program 2013] in connection with the search for a model of Voevodsky's *axiom of univalence* which has computational content. Bezem et al. [2014] introduce such a model, based on a category of presheaves $[\mathbf{C}^{\mathrm{op}}, \mathbf{Set}]$, that is, Set-valued functors on the opposite of a certain category $\mathbf{C}$. The objects of $\mathbf{C}$ are finite subsets A of $\mathbb{A}$, where one thinks of the elements of A as a finite number of named cartesian directions (axes); the morphisms $A \to B$ between two such finite subsets are functions $f : B \to A \uplus \{0,1\}$ satisfying $(\forall b, b' \in B)\ f\,b = f\,b' \notin \{0,1\} \Rightarrow b = b'$; such morphisms compose in an evident fashion and there are identity morphisms for that composition. The category $[\mathbf{C}^{\mathrm{op}}, \mathbf{Set}]$ contains a distinguished object I, given by the representable presheaf on any one-element subset of $\mathbb{A}$, which acts like an interval. The representable presheaf $\mathbf{C}(\_, A)$ on an n-element subset $A \in \mathbf{C}$ is isomorphic to I^n and hence for each $X \in [\mathbf{C}^{\mathrm{op}}, \mathbf{Set}]$, the set $X(A)$ is in bijection with $[\mathbf{C}^{\mathrm{op}}, \mathbf{Set}](I^n, X)$, the n-cubes of X.

At first the nominal aspect of this description – using finite sets of atoms instead of finite ordinals for the objects of $\mathbf{C}$ – might seem trivial. However, the nominal approach allows the path space for an object $X \in [\mathbf{C}^{\mathrm{op}}, \mathbf{Set}]$, that is the exponential X^I, to be described by a construction very like name abstraction (Definition 5.1) and to inherit its good properties. Such path spaces are the (total spaces of) intensional identity types once one restricts attention to Kan-fibrant objects in $[\mathbf{C}^{\mathrm{op}}, \mathbf{Set}]$; see Bezem et al. [2014, Section 8.2]. In fact path spaces are not just very like name-abstraction nominal sets, they actually coincide with them over an equivalence of categories between $[\mathbf{C}^{\mathrm{op}}, \mathbf{Set}]$ and a category of nominal sets equipped with a notion of substitution of 0 or 1 for names. This equivalence was first noted by Staton in his study of equivalences between (pre)sheaf categories and nominal sets equipped with substitution structures [Staton 2007; 2010] and provides yet another illustration of point (A) in Section 3. In fact Pitts [2015b] shows that $[\mathbf{C}^{\mathrm{op}}, \mathbf{Set}]$ is also equivalent to a category of finitely supported M-sets for a certain monoid of substitution operations. The point of this way of describing cubical sets is two-fold. First, a lot of the theory of group actions to do with support, freshness and name abstraction extends to monoid-actions. Secondly, by choosing M suitably one gets, up to equivalence, the latest and (so far) greatest cubical sets model of homotopy type theory [Cohen et al. 2015].

11. CONCLUSION

I write this article from the UK. It is the first country to allocate research funding based on its societal impact (www.ref.ac.uk). Leaving aside the question of whether this is a Good or a Bad Thing (or somewhere in between), I think we can all agree that it can take a long (possibly infinite) time for some great mathematical ideas to have impact outside their own area of specialism. For example, the foundational work of Abraham Fraenkel and Andrzej Mostowski on symmetric models of set theory took place in Germany and Poland in the 1920s and 1930s. The nominal techniques in computer science that I have described in this article depend in part upon their work and emerged about 15 years ago. To begin with, the emphasis was on program semantics and understanding locality of resources. More recently a broader agenda of *symmetry-aware* computation has emerged [Bojanczyk et al. 2014]. I hope this article has convinced you that work on nominal techniques is interesting and useful. But maybe it is still too early to tell what will be the lasting impact of this work on computer science.

REFERENCES

S. Abramsky. 1991. Domain Theory In Logical Form. *Annals of Pure and Applied Logic* 51 (1991), 1–77.

S. Abramsky and A. Jung. 1994. Domain Theory. In *Handbook of Logic in Computer Science, Volume 3. Semantic Structures*, S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum (Eds.). Oxford University Press, Chapter 1, 1–168. (Corrected and expanded version available via the second author's web pages.).

S. Awodey. 2010. *Category Theory* (2nd ed.). Oxford Logic Guides, Vol. 52. Oxford University Press.

B. Aydemir, A. Bohannon, and S. Weirich. 2007. Nominal Reasoning Techniques in Coq: (Extended Abstract). *Electronic Notes in Theoretical Computer Science* 174, 5 (2007), 69–77. DOI:http://dx.doi.org/10.1016/j.entcs.2007.01.028 Proceedings of the First International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2006).

H. P. Barendregt. 1984. *The Lambda Calculus: Its Syntax and Semantics* (revised ed.). North-Holland.

M. Bezem, T. Coquand, and S. Huber. 2014. A Model of Type Theory in Cubical Sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013) (Leibniz International Proceedings in Informatics (LIPIcs))*, R. Matthes and A. Schubert (Eds.), Vol. 26. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 107–128. DOI:http://dx.doi.org/10.4230/LIPIcs.TYPES.2013.107

M. Bojańczyk, L. Braud, B. Klin, and S. Lasota. 2012. Towards Nominal Computation. In *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2012)*, M. Hicks (Ed.). ACM Press, New York, NY, USA, 401–412.

M. Bojanczyk, B. Klin, A. Kurz, and A. M. Pitts. 2014. Nominal Computation Theory (Dagstuhl Seminar 13422). *Dagstuhl Reports* 3, 10 (2014), 58–71. DOI:http://dx.doi.org/10.4230/DagRep.3.10.58

M. Bojańczyk, B. Klin, and S. Lasota. 2014. Automata Theory in Nominal Sets. *Logical Methods in Computer Science* 10, 3 (Aug. 2014), paper 4.

M. Bojańczyk, B. Klin, S. Lasota, and S. Toruńczyk. 2013. Turing Machines with Atoms. In *Logic in Computer Science (LICS), 2013 28th Annual IEEE/ACM Symposium on*. 183–192. DOI:http://dx.doi.org/10.1109/LICS.2013.24

J. Cheney. 2012. A Dependent Nominal Type Theory. *Logical Methods in Computer Science* 8 (2012), (1:08). DOI:http://dx.doi.org/10.2168/LMCS-8(1:8)2012

J. Cheney and C. Urban. 2008. Nominal Logic Programming. *Transactions on Programming Languages and Systems* 30, 5 (2008), 1–47. DOI:http://dx.doi.org/10.1145/1387673.1387675

M. Cimini, M. R. Mousavi, M. A. Reniers, and M. J. Gabbay. 2012. Nominal SOS. *Electronic Notes in Theoretical Computer Science* 286 (2012), 103–116. DOI:http://dx.doi.org/10.1016/j.entcs.2012.08.008 Proceedings of the 28th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVIII).

C. Cohen, T. Coquand, S. Huber, and A. Mörtberg. 2015. Cubical Type Theory: a Constructive Interpretation of the Univalence Axiom. (Dec. 2015). Preprint.

Peter Dybjer. 1996. Internal type theory. In *Types for Proofs and Programs*, S. Berardi and M. Coppo (Eds.). Lecture Notes in Computer Science, Vol. 1158. Springer Berlin Heidelberg, 120–134. DOI:http://dx.doi.org/10.1007/3-540-61780-9_66

O. Grumberg E. M. Clarke and D. A. Peled. 2000. *Model Checking*. MIT Press.

E. Fairweather, M. Fernández, N. Szasz, and A. Tasistro. 2015. Dependent Types for Nominal Terms with Atom Substitutions. In *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015) (Leibniz International Proceedings in Informatics (LIPIcs))*, Thorsten Altenkirch (Ed.), Vol. 38. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 180–195. DOI:http://dx.doi.org/10.4230/LIPIcs.TLCA.2015.180

M. Fernández, M.J. Gabbay, and I. Mackie. 2004. Nominal Rewriting Systems. In *Proc. 6th ACM-SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'04)*. ACM Press, 108–119.

A. A. Fraenkel. 1922. Der Begriff 'definit' und die Unabhängigkeit des Auswahlsaxioms. *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse* (1922), 253–257.

M. J. Gabbay. 2003. The π-calculus in FM. In *Thirty-Five Years of Automating Mathematics*, Fairouz Kamareddine (Ed.). Applied Logic, Vol. 28. Kluwer, 71–123.

M. J. Gabbay. 2009. A Study of Substitution, Using Nominal Techniques and Fraenkel-Mostowski Sets. *Theoretical Computer Science* 410, 12-13 (March 2009), 1159–1189. DOI:http://dx.doi.org/10.1016/j.tcs.2008.11.013

M. J. Gabbay. 2011. Foundations of Nominal Techniques: Logic and Semantics of Variables in Abstract Syntax. *Bulletin of Symbolic Logic* 17, 2 (2011), 161–229.

M. J. Gabbay and V. Ciancia. 2011. Freshness and Name-Restriction in Sets of Traces with Names. In *Foundations of Software Science and Computation Structures, 14th International Conference (FOSSACS 2011) (Lecture Notes in Computer Science)*, Vol. 6604. Springer-Verlag, 365–380.

M. J. Gabbay, T. Litak, and D. L. Petrişan. 2011. Stone Duality for Nominal Boolean Algebras with И. In *Algebra and Coalgebra in Computer Science*, A. Corradini, B. Klin, and C. Cîrstea (Eds.). Lecture Notes in Computer Science, Vol. 6859. Springer Berlin / Heidelberg, 192–207. http://dx.doi.org/10.1007/978-3-642-22944-2_14

M. J. Gabbay and A. M. Pitts. 2002. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing* 13 (2002), 341–363.

F. Gadducci, M. Miculan, and U. Montanari. 2006. About Permutation Algebras, (Pre)sheaves and Named Sets. *Higher-Order and Symbolic Computation* 19 (2006), 283–304. DOI:http://dx.doi.org/10.1007/s10990-006-8749-3

J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. 1977. Initial Algebra Semantics and Continuous Algebras. *J. ACM* 24 (1977), 68–95.

P. T. Johnstone. 2002. *Sketches of an Elephant, A Topos Theory Compendium, Volumes 1 and 2*. Number 43–44 in Oxford Logic Guides. Oxford University Press.

M. Kaminski and N. Francez. 1994. Finite-Memory Automata. *Theoretical Computer Science* 134, 2 (1994), 329–363.

D. Kozen, K. Mamouras, D. L. Petrişan, and A. Silva. 2015. Nominal Kleene Coalgebra. In *Automata, Languages, and Programming*. Lecture Notes in Computer Science, Vol. 9135. Springer Berlin Heidelberg, 286–298. DOI:http://dx.doi.org/10.1007/978-3-662-47666-6_23

A. Kurz, D. L. Petrişan, P. Severi, and F.-J. de Vries. 2013. Nominal Coalgebraic Data Types with Applications to Lambda Calculus. *Logical Methods in Computer Science* 9, 4:20 (Dec. 2013).

J. Lambek and P. J. Scott. 1986. *Introduction to Higher Order Categorical Logic*. Cambridge University Press.

S. Lösch. 2014. *Program Equivalence in Functional Metaprogramming via Nominal Scott Domains*. Ph.D. Dissertation. University of Cambridge.

S. Lösch and A. M. Pitts. 2014. Denotational Semantics with Nominal Scott Domains. *Journal of the ACM* 61, 4 (July 2014), 27:1–27:46. DOI:http://dx.doi.org/10.1145/2629529

S. MacLane. 1971. *Categories for the Working Mathematician*. Springer.

R. Milner, J. Parrow, and D. Walker. 1992. A Calculus of Mobile Processes (Parts I and II). *Information and Computation* 100 (1992), 1–77.

U. Montanari and M. Pistore. 2000. π-Calculus, Structured Coalgebras and Minimal HD-Automata. In *25th International Symposium on Mathematical Foundations of Computer Science, Bratislava, Slovak Republic, (Lecture Notes in Computer Science)*, Vol. 1893. Springer-Verlag, 569–578.

A. Mostowski. 1939. Uber die Unabhängigkeit des Wohlordnungssatzes vom Ordnungsprinzip. *Fundamenta Mathematicae* (1939), 201–252.

A. Murawski and N. Tzevelekos. 2013. Towards Nominal Abramsky. In *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky*, B. Coecke, L. Ong, and P. Panangaden (Eds.). Lecture Notes in Computer Science, Vol. 7860. Springer Berlin Heidelberg, 246–263. DOI:http://dx.doi.org/10.1007/978-3-642-38164-5_17

M. Odersky. 1994. A Functional Theory of Local Names. In *Conference Record of the 21st Annual ACM Symposium on Principles of Programming Languages*. ACM Press, 48–59.

A.M. Pitts. 2015a. Names and Symmetry in Computer Science (Invited Tutorial). In *Logic in Computer Science (LICS), 2015 30th Annual ACM/IEEE Symposium on*. IEEE Computer Society Press, 21–22. DOI:http://dx.doi.org/10.1109/LICS.2015.12

A. M. Pitts. 2006. Alpha-Structural Recursion and Induction. *Journal of the ACM* 53 (2006), 459–506. DOI:http://dx.doi.org/10.1145/1147954.1147961

A. M. Pitts. 2013. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science, Vol. 57. Cambridge University Press.

A. M. Pitts. 2015b. Nominal Presentations of the Cubical Sets Model of Type Theory. In *20th International Conference on Types for Proofs and Programs (TYPES 2014) (Leibniz International Proceedings in Informatics (LIPIcs))*, H. Herbelin, P. Letouzey, and M. Sozeau (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, ?–? to appear.

A. M. Pitts, J. Matthiesen, and J. Derikx. 2015. A Dependent Type Theory with Abstractable Names. In *Proceedings of the 9th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2014) (Electronic Notes in Theoretical Computer Science)*, I. Mackie and M. Ayala-Rincon (Eds.), Vol. 312. Elsevier, 19–50. DOI:http://dx.doi.org/10.1016/j.entcs.2015.04.003

G. D. Plotkin. 1977. LCF Considered as a Programming Language. *Theoretical Computer Science* 5 (1977), 223–255.

R. Rado. 1964. Universal Graphs and Universal Functions. *Acta Arithmetica* 9 (1964), 331–340.

U. Schöpp. 2006. *Names and Binding in Type Theory*. Ph.D. Dissertation. University of Edinburgh.

U. Schöpp and I. D. B. Stark. 2004. A Dependent Type Theory with Names and Binding. In *Computer Science Logic, CSL04, Karpacz, Poland (Lecture Notes in Computer Science)*, Vol. 3210. Springer-Verlag, 235–249.

D. S. Scott. 1982. Domains for Denotational Semantics. In *Automata, Languages and Programming, Proceedings 1982 (Lecture Notes in Computer Science)*, M. Nielson and E. M. Schmidt (Eds.), Vol. 140. Springer-Verlag, Berlin, 577–610.

M. R. Shinwell. 2003. Swapping the Atom: Programming with Binders in Fresh O'Caml. In *Second ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding, MERLIN'03, Uppsala, Sweden, August 2003*. ACM Press.

M. R. Shinwell. 2005. Fresh O'Caml: Nominal Abstract Syntax for the Masses. In *2005 ACM SIGPLAN Workshop on ML (ML 2005), Tallinn, Estonia (Electronic Notes in Theoretical Computer Science)*, P. N. Benton and X. Leroy (Eds.). Elsevier, 53–76.

M. R. Shinwell and A. M. Pitts. 2005. On a Monadic Semantics for Freshness. *Theoretical Computer Science* 342 (2005), 28–55.

M. R. Shinwell, A. M. Pitts, and M. J. Gabbay. 2003. FreshML: Programming with Binders Made Simple. In *Eighth ACM SIGPLAN International Conference on Functional Programming (ICFP 2003), Uppsala, Sweden*. ACM Press, 263–274.

S. Staton. 2007. *Name-Passing Process Calculi: Operational Models and Structural Operational Semantics*. Ph.D. Dissertation. University of Cambridge. Available as University of Cambridge Computer Laboratory Technical Report Number UCAM-CL-TR-688.

S. Staton. 2010. Completeness for Algebraic Theories of Local State. In *Foundations of Software Science and Computational Structures*, L. Ong (Ed.). Lecture Notes in Computer Science, Vol. 6014. Springer Berlin Heidelberg, 48–63. DOI:http://dx.doi.org/10.1007/978-3-642-12032-9_5

A. Swan. 2014. An Algebraic Weak Factorisation System on 01-Substitution Sets: A Constructive Proof. *ArXiv e-prints* arXiv:1409.1829 (Sept. 2014). http://arxiv.org/abs/1409.1829

D. C. Turner. 2009. *Nominal Domain Theory for Concurrency*. Ph.D. Dissertation. University of Cambridge. Available as University of Cambridge Computer Laboratory Technical Report UCAM-CL-TR-751.

D. C. Turner and G. Winskel. 2009. Nominal Domain Theory for Concurrency. In *Computer Science Logic*, E. Grädel and R. Kahle (Eds.). Lecture Notes in Computer Science, Vol. 5771. Springer-Verlag, 546–560. http://dx.doi.org/10.1007/978-3-642-04027-6_39

N. Tzevelekos. 2011. Fresh-Register Automata. In *38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2011)*. ACM, New York, NY, USA, 295–306. DOI:http://dx.doi.org/10.1145/1926385.1926420

The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations for Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study.

C. Urban, A. M. Pitts, and M. J. Gabbay. 2004. Nominal Unification. *Theoretical Computer Science* 323 (2004), 473–497.

C. Urban and C. Tasson. 2005. Nominal Techniques in Isabelle/HOL. In *20th International Conference on Automated Deduction, CADE-20, Tallinn, Estonia, July 2005 (Lecture Notes in Computer Science)*, Vol. 3632. Springer-Verlag, 38–53.

S. Vickers. 1989. *Topology via Logic*. Cambridge Tracts in Theoretical Computer Science, Vol. 5. Cambridge University Press.

E. Westbrook, A. Stump, and E. Austin. 2009. The Calculus of Nominal Inductive Constructions: an Intensional Approach to Encoding Name-Bindings. In *Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-languages: Theory and Practice (LFMTP 2009), Montreal, Canada (ACM International Conference Proceeding Series)*. ACM Press, 74–83.