A New Approach to Abstract Syntax Involving Binders

Murdoch Gabbay
Cambridge University
DPMMS
Cambridge CB2 1SB, UK
M.J.Gabbay@cantab.com

Andrew Pitts
Cambridge University
Computer Laboratory
Cambridge CB2 3QG, UK
ap@cl.cam.ac.uk

Abstract

*The Fraenkel-Mostowski permutation model of set theory with atoms (*FM-sets*) can serve as the semantic basis of meta-logics for specifying and reasoning about formal systems involving name binding, α-conversion, capture avoiding substitution, and so on. We show that in FM-set theory one can express statements quantifying over 'fresh' names and we use this to give a novel set-theoretic interpretation of name abstraction. Inductively defined FM-sets involving this name-abstraction set former (together with cartesian product and disjoint union) can correctly encode object-level syntax modulo α-conversion. In this way, the standard theory of algebraic data types can be extended to encompass signatures involving binding operators. In particular, there is an associated notion of structural recursion for defining syntax-manipulating functions (such as capture avoiding substitution, set of free variables, etc) and a notion of proof by structural induction, both of which remain pleasingly close to informal practice.*

1. Introduction

The message of this paper is that a modest change to classical set theory can yield benefits for the meta-theory of formal systems that involve name-binding (which most of them do). But of course such a change is not to be undertaken lightly, so we begin with a critique of the current state of the art. It is oriented towards the particular use of such a meta-theory that most concerns us: the formalisation, quite probably with machine-assistance, of proofs about the operational semantics of programming languages.

Background The theory and practice of specifying and reasoning about syntactical structures that *do not* involve binding constructs is well understood. The theory involves such indispensable concepts as user-declared algebraic data types [10] (inductively defined sets), structural recursion over such data, and proof by structural induction [1]; the practice can be seen in several general-purpose systems for machine-assisted proof (such as [12, 30]). This algebraic, 'no binders' machinery is often applied to syntax that *does* involve binders; but in that case it yields overly-concrete representations in which large numbers of essentially routine constructions and proofs to do with renaming bound variables, capture avoiding substitution, and so on, must be done and re-done for each object-language on a case-by-case basis. If only to make large, machine-checkable proofs feasible, a more sophisticated approach is called for.

One such approach involves representing object-level variables by variables in a meta-language based on typed λ-calculus. This shifts renaming and substitution to the meta-level where their properties are established once and for all. This is the 'higher order abstract syntax' (HOAS) approach—an idea going back to Church [2] and Martin-Löf [23] which has found its way into many of the current logical frameworks and proof assistants. Its big drawback, in its original form at least, is that one looses the ability to define functions on syntax by structural recursion and to prove properties by structural induction—absolutely essential tools for our intended applications to operational semantics. There are recent proposals to overcome this shortcoming [24, 5]. They result in systems which are technically very interesting but force the designer of algorithms and proofs 'modulo-α-conversion' to use forms of expression in our view rather far from familiar informal practice. Indeed, the whole HOAS approach by its very nature disallows a feature that we regard of key practical importance: *the ability to manipulate names of bound variables explicitly* in computation and proof. Of course, one can introduce a type of 'names' in a HOAS signature, as for example is done in [15]; but as the authors of that work say [p 26]

"The main drawback of HOAS is the difficulty of dealing with metatheoretic issues concerning names . . . . As a consequence, some metatheoretic properties involving substitution and freshness of names . . . cannot be proved inside the framework and instead have to be postulated."

It is precisely such problems with names which we claim our approach overcomes in a simple way. Similar criticisms apply to approaches to binding based upon de Bruijn's nameless terms or categorical combinators [4, 3, 7]: these are good for machine implementations, but not, we would argue, for representations intended for machine-assisted *human* reasoning.

Instead of the HOAS approach of moving both α-conversion and substitution to the meta-level, we will just promote the former, leaving notions of substitution to be defined by structural recursion on a case-by-case basis. This does not seem too bad a compromise, since we show that it permits both a nice calculus of bound names and notions of structural recursion and structural induction for variable-binding constructs modulo renaming. We present these in an extensional framework with the expressive power of classical set theory which remains close to informal practice in its forms of expression. This, and the focus on α-conversion, makes our work close in spirit to that of Gordon and Melham [11], who axiomatise a type of untyped λ-terms modulo α-conversion within Church's higher order logic. However, we take a more foundational approach, in that the necessary properties of renaming become part of the underlying set theory (or higher order logic—since we believe one can use that rather than set theory as the basis of the approach described here, but we have yet to develop this formulation). This results in notions of structural recursion and induction that seem rather simpler than those in [11] (cf. Example 5.8 below).

Contributions of this paper We motivate our use of Fraenkel and Mostowski's permutation model of sets (the 'FM-universe' as we call it) by considering the operation of permuting the variables in an expression and its relation to α-conversion (Theorem 2.1). The fundamental notion of 'finite support' is recalled in Section 3 and used to define a quantifier for 'fresh' names (Definition 3.4). These concepts are used in the key definition of the paper: an apparently new *set-theoretic notion of name-abstraction* (Definition 4.2). Its importance is justified by the observation (of which Theorem 5.1 is a specific example) that *data types of syntax modulo α-conversion can be correctly modelled by sets in the FM-universe that are inductively defined by operators built up using this new abstraction set-former in combination with the usual operators for disjoint union and cartesian product*. In this way, the standard initial algebra semantics of algebraic data types can be extended to encompass signatures involving binding operators. The notions of finite support and quantification over fresh names in the FM-universe enable us to formulate versions of structural recursion and induction for such signatures with binders. We give examples of this in Section 5. Section 6 sketches the relationship of our approach to recent work on modelling variable-binding abstract syntax in presheaf categories [7, 14]. Finally, Section 7 mentions some of the many things that remain to be done to develop our set-theoretic modelling of abstract syntax involving binders. One of our main motivations is to produce a meta-logic for specifying and reasoning about syntax and semantics (of programming languages involving binding constructs) that is close to informal practice when it comes to the crucial matter of structural recursion/induction. We claim that the ideas introduced here provide some interesting raw material along those lines; but it should be emphasised that many potentially difficult issues of 'proof engineering' lie between our notion of abstraction in the FM-universe and its application in mechanised proof assistants.

2. Permutative renaming

In the '20s and '30s Fraenkel and Mostowski devised their permutation model in order to prove the independence of the Axiom of Choice (AC) from the other axioms of set theory *with atoms* (ZFA) (and three decades later Cohen proved the harder result of independence of AC from set theory without atoms (ZF), via his celebrated forcing method; see [16, Section 6] for a brief survey of these matters). Our application of their model is rather far from this purpose! To motivate its use let us consider the paradigmatic example, namely the terms of the untyped lambda calculus, which we can take to be elements of the following inductively defined set of syntax trees:

$$\Lambda \overset{\text{def}}{=} \mu X.\, \mathsf{Var}(\mathbb{A}) \mid \mathsf{App}(X \times X) \mid \mathsf{Lam}(\mathbb{A} \times X) \quad (1)$$

where $\mathbb{A}$ is some fixed, countably infinite set whose elements we call *atoms* (or 'names', but that term is too overloaded). Consider the following three versions of the notion of variable-renaming for elements M of Λ, where $a, a' \in \mathbb{A}$:

$[a'/a]M$ capture-avoiding substitution of a' for all free occurrences of a in M;

$\{a'/a\}M$ textual substitution of a' for all free occurrences of a in M;

$(a'\ a) \cdot M$ interchange of *all* occurrences (be they free, bound, or binding) of a and a' in M.

Although the third version is possibly unfamiliar, it is in fact more basic than the other two because: firstly, one does not need to know whether any of the constructors defining Λ are binders in order to define it; secondly, it can nevertheless be used to define α-conversion, as the following result shows (cf. [13, p 36], which uses $\{a'/a\}(\Leftrightarrow)$ in place of $(a'\ a) \cdot (\Leftrightarrow)$ for the same purpose).

Theorem 2.1. *Recall that α-conversion, $=_\alpha$, is usually defined as the least congruence on Λ that identifies $\mathsf{Lam}(a,M)$ with $\mathsf{Lam}(a',[a'/a]M)$. Then $=_\alpha$ coincides with the binary relation $\sim$ on Λ inductively generated by the following axioms and rules.*

$$\mathsf{Var}(a) \sim \mathsf{Var}(a)$$

$$\frac{M_1 \sim M_1' \quad M_2 \sim M_2'}{\mathsf{App}(M_1,M_2) \sim \mathsf{App}(M_1',M_2')}$$

$$\frac{(a''\,a)\cdot M \sim (a''\,a')\cdot M'}{\mathsf{Lam}(a,M) \sim \mathsf{Lam}(a',M')} \quad \textit{if } a'' \textit{ does not occur in } M, M'.$$

Proof. It is not hard to see that $(a'\,a)\cdot(\Leftrightarrow)$ preserves $=_\alpha$ and hence that $=_\alpha$ is closed under the axioms and rules defining $\sim$. Therefore $\sim$ is contained in $=_\alpha$. The converse follows by proving that $\sim$ is a congruence relating $\mathsf{Lam}(a,M)$ to $\mathsf{Lam}(a',[a'/a]M)$: this follows from the facts that $(a'\,a)\cdot(\Leftrightarrow)$ preserves $\sim$, and that if a' does not occur in M, then $(a'\,a)\cdot M \sim [a'/a]M$. ☐

This theorem suggests that matters to do with variable binding can be phrased in terms of the operation of variable-transposition $(a'\,a)\cdot(\Leftrightarrow)$, rather than the more familiar operation of variable-substitution (be it textual $\{a'/a\}M$, or capture-avoiding $[a'/a]M$). Note that transposition is an instance of the more general operation of *permuting the atoms in M according to a bijection $\pi : \mathbb{A} \cong \mathbb{A}$*, the result of which we write as $\pi \cdot M$. This 'permutation action' permits one to formalise one essence of the notion of 'variable', namely that *properties of syntax should be sensitive only to* distinctions *between variable names, rather than to the particular names themselves.* Put more formally, this is the *equivariance* property of sentences $\forall\vec{x}\,.\,\phi(\vec{x})$ about syntax

$$\forall\pi,\vec{x}\,.\,(\phi(\vec{x}) \;\Leftrightarrow\; \phi(\pi\cdot\vec{x})) \tag{2}$$

the validity of which of course depends upon the nature of ϕ, but also, crucially, upon the fact that all the free variables of ϕ are listed in $\vec{x}$. Such notions belong to the rich mathematical theory of sets equipped with a permutation action, which we draw upon next. It is important to note that much of that theory would be inapplicable were one to try to base the development upon arbitrary (or even injective) functions from atoms to atoms, rather than upon permutations.

3. FM-sets

Given a group G, recall that a *G-set* is a set X equipped with a *G-action*, which by definition is a function mapping pairs $(\pi,x) \in G \times X$ to elements $\pi\cdot x \in X$ and satisfying for all $x \in X$ and $\pi,\pi' \in G$ that $id\cdot x = x$ and $\pi\cdot(\pi'\cdot x) = \pi\pi'\cdot x$ (where id is the group identity and $(\pi,\pi') \mapsto \pi\pi'$ the group multiplication). We need this notion for the case $G = S_{\mathbb{A}}$, the group of all permutations of the set $\mathbb{A}$. (In Section 7 we need to consider products of such groups.) Note that $\mathbb{A}$ is itself an $S_{\mathbb{A}}$-set if we define $\pi\cdot a$ to be $\pi(a)$; and then the set Λ of lambda terms from the previous section is an $S_{\mathbb{A}}$-set via an action defined recursively from that one:

$$\begin{aligned}
\pi\cdot\mathsf{Var}(a) &\overset{\text{def}}{=} \mathsf{Var}(\pi(a))\\
\pi\cdot\mathsf{App}(M,M') &\overset{\text{def}}{=} \mathsf{App}(\pi\cdot M, \pi\cdot M')\\
\pi\cdot\mathsf{Lam}(a,M) &\overset{\text{def}}{=} \mathsf{Lam}(\pi(a), \pi\cdot M).
\end{aligned}$$

$S_{\mathbb{A}}$-sets like Λ have an important finiteness property: their elements only involve finitely many different atoms. The following key notion expresses this property purely in terms of the action, and hence can be applied to any $S_{\mathbb{A}}$-set, whether or not it is given concretely in terms of syntax trees.

Definition 3.1 (Finite support). Let X be an $S_{\mathbb{A}}$-set. A subset $\omega \subseteq \mathbb{A}$ *supports* $x \in X$ if for all $\pi \in S_{\mathbb{A}}$

$$(\forall a \in \omega\,.\,\pi(a) = a) \Rightarrow \pi\cdot x = x.$$

We say x is *finitely supported* if there is some finite $\omega \subseteq \mathbb{A}$ supporting x. In fact one can prove that if x is finitely supported, then there is a smallest finite subset of $\mathbb{A}$ supporting it: we call this the *support* of x, and denote it by $supp(x)$ (leaving implicit which $S_{\mathbb{A}}$-set X is being referred to). We say that an atom a *is apart from* x, and write $a \# x$, if $a \notin supp(x)$.

Recall the usual von Neumann cumulative hierarchy of sets, $\mathcal{V}$ [35]. We can build the notions of 'permutation action' and 'finite support property' into a set-theoretic universe by replacing $\mathcal{V}$ with the *Fraenkel-Mostowski universe*, $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$, which by definition is the least $S_{\mathbb{A}}$-class (i.e. class with an $S_{\mathbb{A}}$-action) $\mathcal{X}$ satisfying

$$\mathcal{X} = \mathbb{A} + pow_{\mathrm{fs}}(\mathcal{X})$$

where $+$ is disjoint union, and $S \in pow_{\mathrm{fs}}(\mathcal{X})$ iff $S \in \mathcal{V}$, $S \subseteq \mathcal{X}$, and S is finitely supported for the action given by: $\pi\cdot S = \{\pi\cdot x \mid x \in S\}$. The elements of $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ not in $\mathbb{A}$ will be called *FM-sets* (over the set of atoms $\mathbb{A}$).

The FM-universe $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ can be built up as the union of transfinitely many stages, where at each successor ordinal we take all finitely supported subsets of the previous stage and a copy of $\mathbb{A}$. Each stage, and hence $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ itself, comes equipped with an $S_{\mathbb{A}}$-action making all of its elements finitely supported. Note that an FM-set x is not itself closed under the permutation action unless $supp(x) = \emptyset$. The notion of support of an FM-set is quite subtle. Note in particular that $supp(x)$ may differ from the set of atoms in $TC(x)$, the $\in$-transitive closure of x. For example $supp(\mathbb{A}) = \emptyset$, but $TC(\mathbb{A}) = \mathbb{A}$. $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ contains a copy of the ZF universe $\mathcal{V}$, namely those FM-sets x whose $\in$-transitive closure $TC(x)$ is disjoint from $\mathbb{A}$. We call such an x a *pure* FM-set.

The usual constructions of ZF can be carried out within FM to build various sets. In particular we will make use of the set $\mathbb{N}$ of natural numbers, and the usual constructions of cartesian products, disjoint unions, power- and function-sets.

Axiomatic FM-set theory To develop the properties of the FM-universe further, it is convenient to work in a setting where all set-theoretic constructions are guaranteed to preserve the finite support property and hence keep us within $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$. One can achieve that with a suitable theory of atoms and FM-sets within classical first-order logic with equality. This theory is based upon ZFA—ZF set theory with Atoms (see [8], for example). This has a signature containing not only a binary relation symbol '$\in$' for membership, but also a constant '$\mathbb{A}$' for the set of atoms. ZFA has an axiom expressing the fact that only non-atoms can have elements; its other axioms are like those of ZF set theory, except that certain quantifications $\mathsf{Q}x\,.\,(\Leftrightarrow)$ (for $\mathsf{Q} = \forall, \exists$) have to be restricted to $\mathsf{Q}x \notin \mathbb{A}\,.\,(\Leftrightarrow)$ when x must range just over sets rather than over sets and atoms. The axioms of ZFA are given in an Appendix to this paper. They capture the basic, set-theoretic properties of $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ without saying anything very specific about properties of the set of atoms itself, or of the permutation action. The properties relevant to our intended application depend upon the fact that $\mathbb{A}$ is (countably) infinite and that every element of $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ is finitely supported in the sense of Definition 3.1. We can get both properties by adding the following axiom to ZFA.

Definition 3.2 (The theory FM). Define FM to be the first-order theory obtained from ZFA by adding the axioms

($\mathbb{A}$ Not Finite) $\mathbb{A} \notin pow_{\mathrm{fin}}(\mathbb{A})$

(Fresh) $\forall x\,.\,\exists a \in \mathbb{A}\,.\,a \# x$.

Here and elsewhere we write '$x \in pow_{\mathrm{fin}}(y)$' to indicate a suitable formula in the language of ZFA expressing that x is a finite subset of y. Similarly, '$a \# x$' stands for a suitable formula expressing the notion of apartness given in Definition 3.1: see the Appendix (noting that the theory given there generalises FM to many sorts A of atoms). This in turn requires us to express in the language of ZFA the permutation action—or at least to express the result $(a'\,a) \cdot x$ of transposing atoms a and a' in x, which can be done by $\in$-recursion: once again, see the Appendix for details. $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ is a model of the theory FM just because $\mathbb{A}$ is an infinite set and every element of $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ is finitely supported.

Remark 3.3 (FM $\Rightarrow \neg$AC). Careful formulations of the definition of capture-avoiding substitution quite often make use of a choice function for picking out fresh variables: see [39, Section 2], for example. The vague feeling that such concrete choices should be irrelevant crystallises here into the fact that such choice functions are inconsistent with FM, because it contradicts the Axiom of Choice (AC). For example, the axiom ($\mathbb{A}$ Not Finite) implies that the set of cofinite subsets of $\mathbb{A}$ is a set of non-empty sets; but there is no choice function in $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$—a diagonalisation argument (using the fact that every graph of a function $pow_{\mathrm{fin}}(\mathbb{A}) \to \mathbb{A}$ must be finitely supported) shows this. Proof assistants based on set theory or higher order logic often include Hilbert's choice operator, $\varepsilon x\,.\,\phi$, to provide anonymous notations for terms defined by formulas (see [21, Section 2.1]). Since the ε-operator can be used to prove AC, we cannot add it to FM without inconsistency. However, it would be both consistent and useful to augment the language of FM with a notation for terms that are *uniquely* defined by a formula.

In what follows we make *implicit* use of the theory FM: everything we do can be reduced to its rather spare language and axioms, but we avoid this in order not to obscure the ideas.

The И-quantifier Many consequences of the finite support property of FM-sets are neatly expressed in terms of the following quantifier for 'newness' of atoms.

Definition 3.4. Let $Cof(x)$ denote the FM-set of *cofinite* subsets of an FM-set x, i.e. those $s \subseteq x$ for which $x \setminus s$ is finite. For each formula ϕ of the language of FM, we write

$$\text{И}a \in \mathbb{A}\,.\,\phi$$

for the formula expressing that $\{a \in \mathbb{A} \mid \phi\}$ is in $Cof(\mathbb{A})$.

So this И-quantifier means "for all but finitely many atoms a, ... ". However, the nature of the set of atoms in $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ endows the quantifier with very special properties. For one thing, since every FM-subset s of $\mathbb{A}$ has finite support, it is not hard to see that s is either finite or cofinite. So $Cof(\mathbb{A})$ is an ultrafilter and $\text{И}a \in \mathbb{A}\,.\,(\Leftrightarrow)$ commutes with conjunction, disjunction *and* negation. More is true:

Lemma 3.5. *For any formula ϕ and list of distinct variables $\vec{x}$ in the language of FM, consider the following formulas.*

$$\forall a \in \mathbb{A}\,.\,a \# \vec{x} \Rightarrow \phi \quad (3)$$

$$\text{И}a \in \mathbb{A}\,.\,\phi \quad (4)$$

$$\exists a \in \mathbb{A}\,.\,a \# \vec{x} \wedge \phi \quad (5)$$

(where $a \# \vec{x}$ is a conjunction of apartness formulas, one for each variable in the list). Then in FM, (3) $\Rightarrow$ (4) $\Rightarrow$ (5)*; and if the free variables of ϕ are contained in $\{\vec{x}, a\}$, then also* (5) $\Rightarrow$ (3) *and hence in this case the three formulas are provably equivalent in FM.*

Proof. The proof makes use of the finite support property and in particular the fact that $\{a \in \mathbb{A} \mid a \# \vec{x}\} \in \mathit{Cof}(\mathbb{A})$. For the implication (5) $\Rightarrow$ (3) we also need that FM satisfies the equivariance property (2). $\square$

Remark 3.6 (Proof rules for И). One can extract introduction and elimination rules for the И-quantifier from the above lemma, provided one uses sequents tagged with sets of possibly-free variables (a common practice in categorical logic [20])—e.g. sequents of the form $\Gamma \vdash_{\vec{x}} \phi$, where Γ is a finite set of formulas, ϕ a formula, and $\vec{x}$ a finite set of variables containing those occurring freely in Γ and ϕ. Then we can derive an introduction rule for И of the form

$$\frac{\Gamma, a \# \vec{x} \vdash_{a,\vec{x}} \phi}{\Gamma \vdash_{\vec{x}} \text{И} a \in \mathbb{A} \,.\, \phi}$$

(cf. the usual rule for $\forall$-introduction) and an elimination rule of the form

$$\frac{\Gamma \vdash_{\vec{x}} \text{И} a \in \mathbb{A} \,.\, \phi \quad \Gamma, \phi, a \# \vec{x} \vdash_{a,\vec{x}} \psi}{\Gamma \vdash_{\vec{x}} \psi}$$

(cf. $\exists$-elimination). In these rules, $a, \vec{x}$ means the finite set properly extending $\vec{x}$ with a variable $a \notin \vec{x}$.

In view of the lemma, we are justified in reading $\text{И} a \in \mathbb{A} \,.\, \phi$ as 'for some/any new atom a, it is the case that ϕ'. This simultaneous $\exists$-$\forall$ flavour of the И-quantifier seems to exactly fit many situations where a statement about 'freshness' of variables is required: we choose *some* fresh variable with a particular property, but later on may need the fact that *any* such variable will do. We see this in subsequent sections, where we put the И-quantifier to work.

4. Abstracting atoms

In Section 2 we saw that α-conversion can be formulated in terms of the two notions of permuting variables and of the non-occurrence, or 'apartness', predicate. In Section 3 these two notions were lifted from the particular data type Λ of λ-terms to an enveloping universe of sets. So now we can consider what 'α-conversion of sets' means, arriving at a new, set-theoretic notion of abstraction. By analogy with the relation $\sim$ used in Theorem 2.1 to characterise α-conversion of λ-terms, consider the following binary relation on $\mathbb{A} \times \mathcal{V}_{\mathrm{FM}}(\mathbb{A})$:

$$(a, x) \sim_{\mathbb{A}} (a', x') \overset{\text{def}}{\Leftrightarrow} \text{И} a'' \in \mathbb{A} \,.\, (a''\, a) \cdot x = (a''\, a') \cdot x'.$$

It is not hard to see that $\sim_{\mathbb{A}}$ is an equivalence relation. We denote the $\sim_{\mathbb{A}}$-equivalence class of a pair (a, x) by $[a]x$ and call it the $\mathbb{A}$-*abstraction* determined by $a \in \mathbb{A}$ and $x \in \mathcal{V}_{\mathrm{FM}}(\mathbb{A})$. (We will see below that $[a]x \in \mathcal{V}_{\mathrm{FM}}(\mathbb{A})$.) This is a form of 'abstraction as information hiding' (like that for abstract data types [28]), since $[a]x$ turns out to behave like a pair (a, x) in which the identity of a is hidden. However, and quite remarkably, $\mathbb{A}$-abstractions also embody a notion of 'abstraction as function' (analogous to that occurring in higher order abstract syntax), as the following lemma shows. We write $\mathcal{F}un_{\mathrm{FM}}(\mathbb{A})$ for the subclass of $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ consisting of unary partial functions: i.e. $f \in \mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ is in $\mathcal{F}un_{\mathrm{FM}}(\mathbb{A})$ if and only if it satisfies

$$\begin{aligned} &f \notin \mathbb{A} \;\wedge\; \forall x \in f \,.\, \exists y, z \,.\, x = (y, z) \;\wedge \\ &\forall x, y, z \,.\, (x, y) \in f \wedge (x, z) \in f \;\Rightarrow\; y = z. \end{aligned}$$

Using the easily verified fact that if $(a, x) \sim_{\mathbb{A}} (a, x')$, then $x = x'$, together with the the ZFA axiom of Collection, one obtains the following.

Lemma 4.1 ($\mathbb{A}$-Abstractions are functions). *Each $\mathbb{A}$-abstraction $[a]x$ is an element of $\mathcal{F}un_{\mathrm{FM}}(\mathbb{A})$ and has support $\mathit{supp}([a]x) = \mathit{supp}(x) \setminus \{a\}$.* $\square$

Write $\mathcal{A}bs(\mathbb{A})$ for the subclass of $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ consisting of all $\mathbb{A}$-abstractions $[a]x$, as a ranges over $\mathbb{A}$ and x over $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$. By the lemma, each $f \in \mathcal{A}bs(\mathbb{A})$ is a function; and by construction its domain of definition $\mathit{dom}(f)$ is a subset of $\mathbb{A}$. In fact if $f = [a]x$, one can show that $\mathit{dom}(f)$ is the cofinite set $\{a\} \cup (\mathbb{A} \setminus \mathit{supp}(x))$, which by the lemma is $\mathbb{A} \setminus \mathit{supp}(f)$. Thus if $f \in \mathcal{A}bs(\mathbb{A})$ we can apply the function f to any atom $a \in \mathbb{A}$ satisfying $a \# f$ to obtain an element $f(a) \in \mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ that we call the *concretion of the $\mathbb{A}$-abstraction f at a*. An $\mathbb{A}$-abstraction is in fact uniquely determined by some/any of its concretions, since one can prove in FM that for all $f, f' \in \mathcal{A}bs(\mathbb{A})$

$$(\text{И} a \in \mathbb{A} \,.\, f(a) = f'(a)) \;\Rightarrow\; f = f'.$$

Definition 4.2 (Abstraction set-former). For any FM-set X, the FM-set of $\mathbb{A}$-*abstractions of elements of X* is:

$$[\mathbb{A}]X \overset{\text{def}}{=} \{f \in \mathcal{A}bs(\mathbb{A}) \mid \text{И} a \in \mathbb{A} \,.\, f(a) \in X\}.$$

(An application of the ZFA axiom of Collection is needed to see that this is a set rather than a proper class; moreover it is finitely supported, with $\mathit{supp}([\mathbb{A}]X) = \mathit{supp}(X)$.)

Thus the elements of $[\mathbb{A}]X$ are $\mathbb{A}$-abstractions $[a]x$ satisfying $a \# X$ and $x \in X$. In the rest of this paper we will only be concerned with the case when $\mathit{supp}(X) = \emptyset$, in which case the condition $a \# X$ is vacuously satisfied. The use of the same notation $\Leftrightarrow[\Leftrightarrow]$ for the abstraction set former and for its elements is not ambiguous because $\mathbb{A}$ is not an atom. Using the $[\mathbb{A}](\Leftrightarrow)$ construct in combination with cartesian product and disjoint union, we can form inductively defined FM-sets that allow us to view sets of syntax *modulo α-conversion* as algebraic data types in $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ of a kind that is very close to the 'classical' theory in $\mathcal{V}$ for syntax without binders. We give the paradigmatic example of this, untyped λ-terms modulo $=_\alpha$, in the next section.

5. Example: $\Lambda/{=_\alpha}$ as an inductive FM-set

First, let us recall a little of the theory of inductively defined sets (for the simple case of *finitary* set operators). Given a function F mapping FM-sets to FM-sets which is definable by a formula in FM, monotone for $\subseteq$, and which preserves unions of countable ascending chains, then the least fixed point of F exists; we call it the *inductively defined FM-set determined by* F and denote it by $\mu X \,.\, F(X)$. It can be constructed by the familiar Tarski formula: $\mu X \,.\, F(X) = \bigcup_{n\in\mathbb{N}} F^n(\emptyset)$.

Now it follows from Definition 4.2 that $[\mathbb{A}](\Leftrightarrow)$ is monotone and preserves unions of countable ascending chains of FM-sets. Therefore we can use it in combination with other such functions, such as cartesian product ($\times$) and disjoint union ($+$), to form inductively defined FM-sets. For example, consider

$$\Lambda_\alpha \stackrel{\text{def}}{=} \mu X \,.\, \mathsf{Var}_\alpha(\mathbb{A}) \mid \mathsf{App}_\alpha(X \times X) \mid \mathsf{Lam}_\alpha([\mathbb{A}]X). \qquad (6)$$

As in Section 2, here we are using a notation for disjoint union in which the injection functions are named explicitly—by Var_α, App_α, and Lam_α in this case.

Theorem 5.1 ($\Lambda/{=_\alpha}$ as an inductive FM-set). *Consider the FM-set Λ of untyped λ-terms, inductively defined as in* (1)*. Then $\Lambda/{=_\alpha}$, the FM-set of equivalence classes modulo the equivalence relation of α-conversion, is in bijection with the inductively defined FM-set Λ_α.*

Proof. Combine the proof of Theorem 2.1 with the definition of $[\mathbb{A}](\Leftrightarrow)$. $\square$

Remark 5.2 (Free variables). Under the bijection of Theorem 5.1, the set of (names of) free variables of an α-equivalence class of λ-terms is identified with the support (in the sense of Definition 3.1) of the corresponding element of Λ_α: see Example 5.9 below. In particular, $\{t \in \Lambda_\alpha \mid supp(t) = \emptyset\}$ corresponds to the subset of *closed* λ-terms modulo $=_\alpha$.

Initial algebra semantics Like cartesian product and disjoint union, the abstraction set-former $[\mathbb{A}](\Leftrightarrow)$ is the object part of a *functor* on FM-sets and functions: its action on a function $g \in X \to Y$, is the function $[\mathbb{A}]g \in [\mathbb{A}]X \to [\mathbb{A}]Y$ that maps each $f \in [\mathbb{A}]X$ to the unique $\mathbb{A}$-abstraction $([\mathbb{A}]g)(f)$ satisfying $\text{И}a \in \mathbb{A} \,.\, ([\mathbb{A}]g)(f) = [a]g(f(a))$. Thus each function F on FM-sets built up using $(\Leftrightarrow) \times (\Leftrightarrow)$, $(\Leftrightarrow) + (\Leftrightarrow)$, and $[\mathbb{A}](\Leftrightarrow)$ is functorial; and a standard argument shows that the associated inductively defined FM-set $\mu X \,.\, F(X)$ is an *initial algebra* for this functor. In other words, for every $f \in F(X) \to X$, there is a unique $\overline{f} \in (\mu X \,.\, F(X)) \to X$ such that

$$\begin{tikzcd}
F(\mu X \,.\, F(X)) \arrow[r, equal] \arrow[d, "F(\overline{f})"'] & \mu X \,.\, F(X) \arrow[d, "\overline{f}"] \\
F(X) \arrow[r, "f"'] & X
\end{tikzcd}$$

commutes. In particular we have:

Corollary 5.3. *The FM-set $\Lambda/{=_\alpha}$ of λ-terms modulo $alpha$-conversion is an initial algebra for the functor $\mathbb{A} + (\Leftrightarrow \times \Leftrightarrow) + [\mathbb{A}](\Leftrightarrow)$.* $\square$

The usefulness of the initial algebra property of inductively defined FM-sets can be increased by analysing the nature of functions out of abstraction sets $[\mathbb{A}]X$. For example, specifying an algebra $F(X) \to X$ for $F(\Leftrightarrow) = \mathbb{A} + (\Leftrightarrow \times \Leftrightarrow) + [\mathbb{A}](\Leftrightarrow)$ amounts to giving functions

$$f \in \mathbb{A} \to X \qquad g \in X \times X \to X \qquad h \in [\mathbb{A}]X \to X$$

and one would like to know when h is induced via the quotient mapping $(a, x) \mapsto [a]x$ from a function $\mathbb{A} \times X \to X$. As the following lemma shows, the concepts of finite support and И-quantifier from Section 3 provide an answer.

Lemma 5.4. *For all functions $h \in \mathbb{A} \times X \to Y$ in $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$, there is a (necessarily unique) $h' \in [\mathbb{A}]X \to Y$ satisfying $\text{И}a \in \mathbb{A} \,.\, \forall x \in X \,.\, h'([a]x) = h(a, x)$ iff h satisfies the condition*

$$\text{И}a \in \mathbb{A} \,.\, \forall x \in X \,.\, a \,\#\, h(a, x). \qquad (7)$$

$\square$

The condition (7) captures the idea that the value of the induced function $h' \in ([\mathbb{A}]X) \to Y$ at some $[a]x$ should be independent of the choice of a. The lemma specifies what h' does just when a is 'fresh', but this is enough to tell us the effect of h' on any element of $[\mathbb{A}]X$, because of the way the latter is defined.

Combining the initial algebra property of $\Lambda_\alpha = \mu X \,.\, \mathbb{A} + X \times X + [\mathbb{A}]X$ with the above lemma, we obtain the following principle of structural recursion for λ-terms modulo α-conversion. (Compare it with the Recursion Scheme of [11, Section 3.1].) A similar principle can be derived for other inductively defined FM-sets involving the abstraction set-former.

Theorem 5.5 (Λ_α structural recursion). *Given functions $f \in \mathbb{A} \to X$, $g \in X \times X \times \Lambda_\alpha \times \Lambda_\alpha \to X$, and $h \in \mathbb{A} \times X \times \Lambda_\alpha \to X$ in $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ with h satisfying*

$$\text{И}a \in \mathbb{A}\,.\,\forall x \in X\,.\,\forall t \in \Lambda_\alpha\,.\,a \# h(a,x,t) \qquad (8)$$

then there is a unique $k \in \Lambda_\alpha \to X$ such that

$$\begin{aligned}
&\forall a \in \mathbb{A}\,.\,k(\mathsf{Var}_\alpha(a)) = f(a)\\
&\forall t,t' \in \Lambda_\alpha\,.\,k(\mathsf{App}_\alpha(t,t')) = g(k(t),k(t'),t,t')\\
&\text{И}a \in \mathbb{A}\,.\,\forall t \in \Lambda_\alpha\,.\,k(\mathsf{Lam}_\alpha([a]t)) = h(a,k(t),t).
\end{aligned}$$

Moreover, the support of k is contained in $supp(X) \cup supp(f) \cup supp(g) \cup \sup(h)$. □

Corollary 5.6 (Λ_α structural induction). *Given a subset $S \subseteq \Lambda_\alpha$ in $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$, to prove that S is the whole of Λ_α it suffices to show*

$$\begin{aligned}
&\forall a \in \mathbb{A}\,.\,\mathsf{Var}_\alpha(a) \in S\\
&\forall t,t' \in S\,.\,\mathsf{App}_\alpha(t,t') \in S\\
&\text{И}a \in \mathbb{A}\,.\,\forall t \in S\,.\,\mathsf{Lam}_\alpha([a]t) \in S.
\end{aligned}$$

□

This structural induction principle seems to correspond very closely to informal inductive arguments about α-equivalence classes of λ-terms that proceed by picking representatives and applying structural induction at the level of abstract syntax trees, leaving mute the tedious proofs that such choices do not affect the argument. In effect, by restricting to equivariant properties (cf. (2)), FM-set theory ensures that all those choices of representatives are made in a way that does not affect meaning. In the rest of this section we give some simple examples.

Example 5.7 (Capture-avoiding substitution). Given $t \in \Lambda_\alpha$ and $a \in \mathbb{A}$, if in Theorem 5.5 we take $X = \Lambda_\alpha$ and make suitable choices for the functions f, g, and h, we can deduce that there is a function $sub(t,a)$ that is the unique element $k \in \Lambda_\alpha \to \Lambda_\alpha$ satisfying

$$\begin{aligned}
&\forall a' \in \mathbb{A}\,.\,k(\mathsf{Var}_\alpha(a')) = (\mathsf{if}\ a' = a\ \mathsf{then}\ t\ \mathsf{else}\ \mathsf{Var}_\alpha(a'))\\
&\forall t',t'' \in \Lambda_\alpha\,.\,k(\mathsf{App}_\alpha(t',t'')) = \mathsf{App}_\alpha(k(t'),k(t''))\\
&\text{И}a' \in \mathbb{A}\,.\,\forall t' \in \Lambda_\alpha\,.\,k(\mathsf{Lam}_\alpha([a']t')) = \mathsf{Lam}_\alpha([a']k(t')).
\end{aligned}$$

Condition (8) is satisfied in this case because by Lemma 4.1, $a' \# [a']t'$ holds for any a' and t'. From the above properties it follows that under the bijection of Theorem 5.1 between elements t of Λ_α and α-equivalence classes of λ-terms M, $sub(t,a)$ corresponds to the capture-avoiding substitution function $[M/a](\Leftrightarrow)$. One property of capture-free substitution of λ-terms is that the only free occurrences of a in $[M/a]M'$ are due to free occurrences of a in M. The analogue for Λ_α of this property is the statement that if $a \# t$, then $a \# sub(t,a)(t')$ holds for all t'. This can be proved by structural induction on t' by taking $S = \{t' \in \Lambda_\alpha \mid a \# sub(t,a)(t')\}$ in Corollary 5.6. Hence (by Lemma 3.5) $\text{И}a \in \mathbb{A}\,.\,\forall t' \in \Lambda_\alpha\,.\,a \# sub(t,a)(t')$ holds for any $t \in \Lambda_\alpha$. Thus by Lemma 5.4, $(a,t') \mapsto sub(t,a)(t')$ induces a function $[\mathbb{A}]\Lambda_\alpha \to \Lambda_\alpha$ for each $t \in \Lambda_\alpha$. In this way we get a substitution function $\sigma \in [\mathbb{A}]\Lambda_\alpha \times \Lambda_\alpha \to \Lambda_\alpha$ satisfying for all $(f,t) \in [\mathbb{A}]\Lambda_\alpha \times \Lambda_\alpha$ that $\text{И}a \in \mathbb{A}\,.\,\sigma(f,t) = sub(t,a)(f(a))$. (Cf. the substitution function $\sigma : \delta\Lambda \times \Lambda \to \Lambda$ in [7, Section 3].)

Example 5.8 (A size function). In Theorem 5.5, taking $X = \mathbb{N}$ and suitable choices for the functions f, g, h, we can deduce that there is a unique function $k \in \Lambda_\alpha \to \mathbb{N}$ satisfying

$$\begin{aligned}
&\forall a \in \mathbb{A}\,.\,k(\mathsf{Var}_\alpha(a)) = 1\\
&\forall t,t' \in \Lambda_\alpha\,.\,k(\mathsf{App}_\alpha(t,t')) = k(t) + k(t')\\
&\text{И}a \in \mathbb{A}\,.\,\forall t \in \Lambda_\alpha\,.\,k(\mathsf{Lam}_\alpha([a]t)) = k(t) + 1.
\end{aligned}$$

Condition (8) is satisfied in this case for the same reason as in the previous example. By Lemma 3.5, the last property of k above is equivalent to

$$\forall a \in \mathbb{A}\,.\,a \# k \Rightarrow \forall t \in \Lambda_\alpha\,.\,k(\mathsf{Lam}_\alpha([a]t)) = k(t) + 1\,;$$

but from the last part of Theorem 5.5 we have that $supp(k) = \emptyset$ (since the particular X, f, g, h that determine k all have empty support). Therefore we can strengthen this last defining clause for k to:

$$\forall a \in \mathbb{A}\,.\,\forall t \in \Lambda_\alpha\,.\,k(\mathsf{Lam}_\alpha([a]t)) = k(t) + 1.$$

So the formalism allows us to express very easily the properties we expect a size function to have on α-equivalence classes of λ-terms. (Compare this example with the complications encountered by Gordon and Melham defining a similar function by recursion using their axiomatisation of α-conversion [11, Section 3.3].)

The principle of structural recursion embodied in Theorem 5.5 requires us not only to specify some functions of the correct type, but also to verify the condition (8) for one of them. In the previous two examples, this condition is satisfied simply because a is never in the support of $[a]t$. Here is a different kind of example.

Example 5.9 (Set of free atoms). It is not hard to see that the support of a finite set of atoms $S \in pow_{\mathrm{fin}}(\mathbb{A})$ is just S itself. Hence a is not in the support of $S \setminus \{a\}$. So in Theorem 5.5, taking $X = pow_{\mathrm{fin}}(\mathbb{A})$ and making suitable choices for the functions f, g, h, it follows that there is a unique function $k \in \Lambda_\alpha \to pow_{\mathrm{fin}}(\mathbb{A})$ satisfying

$$\begin{aligned}
&\forall a \in \mathbb{A}\,.\,k(\mathsf{Var}_\alpha(a)) = \{a\}\\
&\forall t,t' \in \Lambda_\alpha\,.\,k(\mathsf{App}_\alpha(t,t')) = k(t) \cup k(t')\\
&\forall a \in \mathbb{A}\,.\,\forall t \in \Lambda_\alpha\,.\,k(\mathsf{Lam}_\alpha([a]t)) = k(t) \setminus \{a\}
\end{aligned}$$

where in the last clause we have strengthened $\mathsf{И}a \in \mathbb{A} \dots$ to $\forall a \in \mathbb{A} \dots$ using the same argument as in the previous example. This function k gives a structurally recursive definition of the 'finite set of free atoms' for elements of Λ_α. Using structural induction (Corollary 5.6) one can show for all $t \in \Lambda_\alpha$ that in fact $k(t) = supp(t)$ (cf. Remark 5.2).

Example 5.10 (Bound atoms). There is no function $ba \in \Lambda_\alpha \to pow_{\text{fin}}(\mathbb{A})$ in $\mathcal{V}_{\text{FM}}(\mathbb{A})$ picking out the 'finite set of bound atoms' of elements of Λ_α, in the sense that it satisfies

$$\begin{aligned}
&\forall a \in \mathbb{A} \,.\, ba(\mathsf{Var}_\alpha(a)) = \emptyset \\
&\forall t, t' \in \Lambda_\alpha \,.\, ba(\mathsf{App}_\alpha(t, t')) = ba(t) \cup ba(t') \\
&\forall a \in \mathbb{A} \,.\, \forall t \in \Lambda_\alpha \,.\, ba(\mathsf{Lam}_\alpha([a]t)) = \{a\} \cup ba(t).
\end{aligned}$$

We cannot use Theorem 5.5 to define such a function because in this case condition (8) would require $a \# \{a\} \cup S$ (any $S \in pow_{\text{fin}}(\mathbb{A})$), which is certainly false. Indeed, we can argue by contradiction to see that no such function exists in $\mathcal{V}_{\text{FM}}(\mathbb{A})$: if it did, it would necessarily have finite support, so picking any atoms $a \neq a'$ not in its support, we would have $(a' \, a) \cdot ba = ba$; then $t = \mathsf{Lam}_\alpha([a]\mathsf{Var}_\alpha(a))$ satisfies $ba(t) = \{a\}$ and so

$$\begin{aligned}
\{a'\} &= (a' \, a) \cdot \{a\} \\
&= (a' \, a) \cdot ba(t) \\
&= ba(t) \qquad \text{since } a, a' \notin supp(ba) \cup supp(t) \\
&= \{a\}
\end{aligned}$$

i.e. $a' = a$, contradicting the choice of a and a'.

The results of this section can be extended to deal with signatures of n-ary operators each of whose arguments is an m-ary abstraction (for $n, m \geq 0$)—the 'binding signatures' considered in [7, Section 2]. An FM-set inductively defined using a set operator given by a suitable sum of products of $\mathbb{A}$ and iterations of the $[\mathbb{A}](\Leftrightarrow)$ operator is in bijection with the set of terms modulo α-conversion over such a signature. This FM-set is an initial algebra for the functor associated with the set operator and from the shape of the latter can be read off principles of structural recursion and induction like those above. Many-sorted signatures can be dealt with using FM-sets mutually inductively defined by several such operators.

Our approach seems rather well adapted to expressing the 'usual' forms of recursion/induction for abstract syntax, while at the same time dealing with freshness of variables and variable renaming systematically, at the meta-level. Of course much more needs to be done to establish the utility of these FM versions of structural recursion and induction principles (we consider some possibilities in Section 7). However, we regard the sheer simplicity of the above examples (compared with analogous examples in other formalisms) as a good sign!

6. Relation to presheaf models

One origin of the work presented here lies in the 'ν-calculus', a calculus of higher order functions and dynamically created names introduced by the second author and Stark [32, 36] (see also [17]). In [37], Stark studies a model of the ν-calculus based on one of Moggi's 'dynamic allocation' monads [29] in the presheaf category $\mathbf{Set}^{\mathcal{I}}$, where $\mathcal{I}$ is the category of finite ordinals and injective functions between them. Crucial ingredients of the dynamic allocation monad used there are the 'object of names', given by the inclusion functor $\mathcal{I} \hookrightarrow \mathbf{Set}$, and the shift functor $\delta : \mathbf{Set}^{\mathcal{I}} \to \mathbf{Set}^{\mathcal{I}}$, given by $\delta X(n) = X(n+1)$. These ingredients also occur in the work on modelling π-calculus names in $\mathbf{Set}^{\mathcal{I}}$ [6, 38] and the recent work on modelling variable-binding abstract syntax [7, 14], where other presheaf categories besides $\mathbf{Set}^{\mathcal{I}}$ are considered.

Now, a somewhat overlooked model of the ν-calculus, mentioned in [33, Examples 4.3], is the full subcategory of $\mathbf{Set}^{\mathcal{I}}$ whose objects are the pullback preserving functors. This is equivalent to a well-known topos, sometimes called the *Schanuel topos*—the category of continuous G-sets for the topological group $G = S_{\mathbb{A}}$ of permutations of a countably infinite set $\mathbb{A}$ topologised as a subspace of Baire space: see [18, Lemma 1.8] and [22, Section III.9]. Put more concretely, and this is the point, the objects of the Schanuel topos are $S_{\mathbb{A}}$-sets in which every element has finite support (Definition 3.1), and its morphisms are the $S_{\mathbb{A}}$-equivariant functions. Thus on the one hand, the Schanuel topos relates to the FM-universe $\mathcal{V}_{\text{FM}}(\mathbb{A})$ much as the usual cumulative hierarchy $\mathcal{V}$ relates to the topos of sets (see [19] for more on the category theory of universes of sets); on the other hand, the Schanuel topos is a sheaf subtopos of the presheaf category $\mathbf{Set}^{\mathcal{I}}$, with the inclusion sending the FM-set of atoms $\mathbb{A}$ to the object of names $\mathcal{I} \hookrightarrow \mathbf{Set}$ and the abstraction operator $[\mathbb{A}](\Leftrightarrow)$ to the shift functor $\delta(\Leftrightarrow)$ mentioned above.

Both the presheaf toposes used in [7, 14] and the Schanuel topos (and indeed many other categories equipped with a faithful functor to $\mathbf{Set}^{\mathbb{N}}$) support an initial algebra semantics for signatures with binding. So does the Schanuel topos, and its associated FM-set theory, have any advantage over presheaf toposes? It is well known that toposes correspond to theories in extensional, higher-order, intuitionistic logic [20]. Unlike presheaf toposes in general, the Schanuel topos models *classical* rather than intuitionistic higher order logic; furthermore, its higher order structure (function and power objects) is rather easy to calculate with, compared to presheaf categories. Thus if one is looking for a single, general-purpose setting for modelling variable-binding syntax, the logic of the Schanuel topos is both a bit more powerful and familiar. One can view [7] as establishing, amongst other things, a very nice categorical algebra for the de Buijn view of variable-binding and substitution. Here however, we are motivated more by the desire for a useful logic of explicit bound names—mainly to formalise existing common practice, but the logic can also serve as a basis for proving that de Bruijn-like formulations are correct with respect to more concrete representations. Crucial to all this is the notion of 'finite support' (leading to the N-quantifier and our FM-set-theoretic notion of abstraction), which is present the Schanuel toposes, but not in the presheaf toposes. We have chosen to use a set-theoretic rather than a topos-theoretic presentation here, because we think it is more accessible; but the same fundamental ideas underlie both $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ and the Schanuel topos.

7. Further directions

Splitting the set of atoms To support syntax involving (finitely many) *different sorts of variables* one can use the following mild generalisation of the FM-universe. Fix an infinite set $\mathcal{A}$ of disjoint, countably infinite sets (of atoms), and form a universe $\mathcal{V}_{\mathrm{FMS}}(\mathcal{A})$ much as in Section 3

$$\mathcal{V}_{\mathrm{FMS}}(\mathcal{A}) \overset{\mathrm{def}}{=} \mu\mathcal{X} \,.\, (\bigcup\mathcal{A}) + pow_{\mathrm{fs}}(\mathcal{X})$$

except that we work with G-sets (and -classes) for G the subgroup of $S_{\bigcup\mathcal{A}}$ consisting of permutations that respect the partition $\mathcal{A}$; so G is isomorphic to the product group $\prod_{A\in\mathcal{A}} S_A$. Clearly, this affects the meaning of the apartness relation $a \mathrel{\#} x$; and we have to replace $\mathsf{N}a \in \mathbb{A}$ and $[\mathbb{A}](\Leftrightarrow)$ by $\mathsf{N}a \in A$ and $[A](\Leftrightarrow)$ respectively, where $A \in \mathcal{A}$. A corresponding generalisation of the first order theory FM to a theory of *Fraenkel-Mostowski set theory with many Sorts of atoms* (*FMS*) is given in an Appendix to this paper. As an example of the use of FMS, here are mutually inductively defined sets in $\mathcal{V}_{\mathrm{FMS}}(\mathcal{A})$ for the types (*Type*) and terms (*Term*) of Girard's system F [9] modulo renaming of bound type variables and bound variables. The definition is parameterised by sets of atoms $T, V \in \mathcal{A}$ for type variables and variables respectively.

$$\begin{array}{lcl@{\qquad}lcl}
\mathit{Type} & = & \mathsf{TyVar}(T) & \mathit{Term} & = & \mathsf{Var}(V)\\
 & \mid & \mathsf{Fun}(\mathit{Type} \times \mathit{Type}) & & \mid & \mathsf{Lam}(\mathit{Type} \times [V]\,\mathit{Term})\\
 & \mid & \mathsf{All}([T]\,\mathit{Type}) & & \mid & \mathsf{App}(\mathit{Term} \times \mathit{Term})\\
 & & & & \mid & \mathsf{Gen}([T]\,\mathit{Term})\\
 & & & & \mid & \mathsf{Spec}(\mathit{Term} \times \mathit{Type}).
\end{array}$$

Equivariant SOS We believe that FM-set theory will be a useful setting for developing programming language semantics based on structural operational semantics [34]. Syntax-directed, rule-based inductive definitions of relations quite often contain side-conditions to do with freshness of variables, and the hope is that these can be assimilated and manipulated conveniently via the N-quantifier we have introduced here. (Indeed, logic programming in the style of Miller and Nadathur [26] involving this quantifier may be possible.) It is also possible that the approach presented here will help extend the 'functorial' operational semantics of [40] to encompass languages involving binders.

FM type theory An extensional set theory is by no means the only setting in which to consider the notions of permutation action and finite support (although we believe it is the simplest place to start). We have begun to investigate what our approach might look like in the setting of constructive type theory [23, 30] and meta-programming [31]. For one thing, it is evident that Definition 4.2 generalises to a *dependently typed* version of abstraction

$$[a \in \mathbb{A}]X(a) \overset{\mathrm{def}}{=} \{f \in \mathcal{A}bs(\mathbb{A}) \mid \mathsf{N}a \in \mathbb{A} \,.\, f(a) \in X(a)\}$$

which bears the same relationship to the N-quantifier as dependent function types (Π) bear to universal quantification ($\forall$). Another interesting possibility is to build information about the apartness relation $a \mathrel{\#} x$ into types whose set-theoretic interpretation is

$$X^{\# a} \overset{\mathrm{def}}{=} \{x \in X \mid a \mathrel{\#} x\}.$$

This opens up the possibility of giving a decidable approximation to the apartness relation as part of a type system. It also allows us to introduce a term-former generalising the N-quantifier:

$$\frac{a : \mathbb{A}, \Gamma^{\# a} \vdash t : X^{\# a}}{\Gamma \vdash (\mathsf{new}\ a\ \mathsf{in}\ t) : X}$$

where $\Gamma^{\# a} = x_1 : X_1^{\# a}, \ldots, x_n : X_n^{\# a}$ if Γ is the type environment $x_1 : X_1, \ldots, x_n : X_n$. Thus, rather in the spirit (though not the letter) of [25], one goal we are aiming for is an SML-like language [27] for meta-programming that combines user-declared data types with primitives for atomic types A (i.e. sorts of atoms), abstraction types $[A]X$, and apartness $X^{\# a}$. A practically important part of such a language will be pattern-matching definitions of (recursively defined) functions out of such data types, using 'binding patterns' $[a]p$ for values of abstraction types and in which 'freshness conditions' like (8) are enforced statically by the type system. For example, given declarations

$$\begin{array}{lcl}
\mathsf{atomictype}\ A; & & \\
\mathsf{datatype}\ \mathit{Alist} & = & \mathit{Nil}\\
 & \mid & \mathit{Cons}\ \mathsf{of}\ A * \mathit{Alist};\\
\mathsf{datatype}\ \mathit{term} & = & \mathit{Var}\ \mathsf{of}\ A\\
 & \mid & \mathit{App}\ \mathsf{of}\ \mathit{term} * \mathit{term}\\
 & \mid & \mathit{Lam}\ \mathsf{of}\ [A]\mathit{term};
\end{array}$$

in this as yet non-existent language, and given the usual declaration for a list append function $\mathit{append} : \mathit{Alist} \times \mathit{Alist} \rightarrow$

$Alist$, we would want the following declarations of functions rem (for anonymously removing an atom from a list of atoms) and fv (for the free variables of a term)

```
fun $rem =$
  $\{\ [a]Nil \to Nil$
  $\mid [a]Cons(a,x) \to rem([a]x)$
  $\mid [a]Cons(a',x)\{a \# a'\} \to Cons(a', rem([a]x))$
  $\}\ ;$
fun $fv =$
  $\{\ Var(a) \to Cons(a, Nil)$
  $\mid App(x,y) \to append(fv\, x)(fv\, y)$
  $\mid Lam([a]x) \to rem([a](fv\, x))$
  $\}\ ;$
```

to type check, with types $rem : [A]Alist \to Alist$ and $fv : term \to Alist$ (cf. Example 5.9). Note the use of repeated variables of atomic type in patterns (equality of atoms is decidable!) and the use of 'qualified patterns' like $[a]Cons(a',x)\{a \# a'\}$ to enforce that a and a' are distinct. On the other hand, the declaration

```
fun $bv =$
  $\{\ Var(a) \to Nil$
  $\mid App(x,y) \to append(bv\, x)(bv\, y)$
  $\mid Lam([a]x) \to Cons(a, bv\, x)$
  $\}\ ;$
```

should *not* type check, for the reasons given in Example 5.10.

8. Conclusion

Fraenkel and Mostowski's permutation model of ZFA is over sixty years old, yet the use to which we have put it here seems new. The idea that one might want to treat syntax up to permutative renamings of variables is hardly original, but the theory really takes off when one combines that idea with the subtle notion of 'finite support' inherent in the Fraenkel-Mostowski model. Using it, we introduced a useful quantifier И for fresh names and a new set-forming operation for name-abstraction, $[\mathbb{A}](\Leftrightarrow)$, whose properties seem better than the function space $\mathbb{A} \to (\Leftrightarrow)$ which is sometimes used to model name-abstraction. Amongst other things, we saw that the theory of inductively defined FM-sets using this notion of abstraction can correctly model α-equivalence classes of variable-binding syntax, while remaining pleasantly close to the familiar theory of first-order algebraic data types.

References

[1] R. M. Burstall. Proving properties of programs by structural induction. *The Computer Journal*, 12:41–48, 1969.

[2] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[3] P.-L. Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming*. Birkhäuser, 1993.

[4] 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.*, 34:381–392, 1972.

[5] J. Despeyroux, F. Pfenning, and C. Schürmann. Primitive recursion for higher-order abstract syntax. In *TLCA'97*, LNCS vol. 1210, pages 147–163. Springer-Verlag, 1997.

[6] M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully abstract model for the π-calculus (extended abstract). In *11th Annual Symposium on Logic in Computer Science*, pages 43–54. IEEE Computer Society Press, Washington, 1996.

[7] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In this volume.

[8] M. P. Fourman. Sheaf models for set theory. *Journal of Pure and Applied Algebra*, 19:91–101, 1980.

[9] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmetique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972. Thèse de doctorat d'état.

[10] J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *JACM*, 24:68–95, 1977.

[11] A. D. Gordon and T. Melham. Five axioms of alpha-conversion. In *TPHOLs'96*, LNCS vol. 1125, pages 173–191. Springer-Verlag, 1996.

[12] M. J. C. Gordon and T. F. Melham. *Introduction to HOL*. Cambridge University Press, 1993.

[13] C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. MIT Press, 1992.

[14] M. Hofmann. Semantical analysis of higher-order abstract syntax. In this volume.

[15] F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (co)inductive type theory. Technical report, Dipartimento di Matematica e Informatica, Università degli Studi di Udine, 1998.

[16] T. J. Jech. About the axiom of choice. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 345–370. North-Holland, 1977.

[17] A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In this volume.

[18] P. T. Johnstone. Quotients of decidable objects in a topos. *Math. Proc. Cambridge Philosophical Society*, 93:409–419, 1983.

[19] A. Joyal and I. Moerdijk. *Algebraic Set Theory*. Cambridge University Press, 1995.

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

[21] L. Lamport and L. C. Paulson. Should your specification language be typed? Technical Report 147, Digital SRC, 1998.

[22] S. MacLane and I. Moerdijk. *Sheaves in Geometry and Logic. A First Introduction to Topos Theory*. Springer-Verlag, 1992.

[23] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

[24] R. McDowell and D. Miller. A logic for reasoning with higher-order abstract syntax. In *12th Annual Symposium on Logic in Computer Science*, pages 434–445. IEEE Computer Society Press, Washington, 1997.

[25] D. Miller. An extension to ML to handle bound variables in data structures: Preliminary report. In *Proceedings of the Logical Frameworks BRA Workshop*, 1990.

[26] D. Miller and G. Nadathur. A logic programming approach to manipulating formulas and programs. In *4th Annual Symposium on Logic in Computer Science*, pages 379–388. IEEE Computer Society Press, Washington, 1987.

[27] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

[28] J. C. Mitchell and G. D. Plotkin. Abtract types have existential types. *ACM Transactions on Programming Languages and Systems*, 10:470–502, 1988.

[29] E. Moggi. An abstract view of programming languages. Lecture Notes, 46 pp, 1989.

[30] C. Paulin-Mohring. Inductive definitions in the system Coq; rules and properties. In M. Bezem and J. F. Groote, editors, *TLCA'93*, LNCS vol. 664, pages 328–345. Springer-Verlag, 1993.

[31] F. Pfenning. Elf: A language for logic definition and verified metaprogramming. In *4th Annual Symposium on Logic in Computer Science*, pages 313–321. IEEE Computer Society Press, Washington, 1987.

[32] A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *MFCS'93*, LNCS vol. 711, pages 122–141. Springer-Verlag, 1993.

[33] A. M. Pitts and I. D. B. Stark. On the observable properties of higher order functions that dynamically create local names (preliminary report). In *Workshop on State in Programming Languages, Copenhagen, 1993*, pages 31–45. ACM SIGPLAN, 1993. Yale Univ. Dept. Computer Science Technical Report YALEU/DCS/RR-968.

[34] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[35] J. R. Shoenfield. Axioms of set theory. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 321–344. North-Holland, 1977.

[36] I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, 1995.

[37] I. D. B. Stark. Categorical models for local names. *Lisp and Symbolic Computation*, 9(1):77–107, 1996.

[38] I. D. B. Stark. A fully abstract domain model for the π-calculus. In *11th Annual Symposium on Logic in Computer Science*, pages 36–42. IEEE Computer Society Press, Washington, 1996.

[39] A. Stoughton. Substitution revisited. *Theoretical Computer Science*, 59:317–325, 1988.

[40] D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *12th Annual Symposium on Logic in Computer Science*, pages 280–291. IEEE Computer Society Press, Washington, 1997.

Appendix: axiomatic FMS set theory

FMS is a single-sorted theory in first-order predicate calculus with equality whose signature consists of a binary relation $\in$ and constants $\mathbb{A}, \mathcal{A}$, and whose axioms are as follows.

Axioms of ZFA

(Sets) $\forall x, y \,.\, x \in y \Rightarrow y \notin \mathbb{A}$.

(Extensionality) $\forall x, y \notin \mathbb{A} \,.\, (\forall z \,.\, z \in x \Leftrightarrow z \in y) \Rightarrow x = y$.

(Separation) $\forall x \,.\, \exists y \notin \mathbb{A} \,.\, \forall z \,.\, z \in y \Leftrightarrow (z \in x \wedge \phi)$ (where y not free in ϕ).

($\in$-Induction) $(\forall x \,.\, (\forall y \in x \,.\, [y/x]\phi) \Rightarrow \phi) \Rightarrow \forall x \,.\, \phi$.

(Collection) $\forall x \,.\, (\forall y \in x \,.\, \phi) \Rightarrow \exists z \,.\, \forall y \in x \,.\, \exists w \in z \,.\, \phi$.

(Pairing) $\forall x, y \,.\, \exists z \,.\, x \in z \wedge y \in z$.

(Union) $\forall x \,.\, \exists y \,.\, \forall z \,.\, z \in y \Leftrightarrow (\exists w \in x \,.\, z \in w)$.

(Powerset) $\forall x \,.\, \exists y \,.\, \forall z \,.\, z \in y \Leftrightarrow \forall w \in z \,.\, w \in x$.

(Infinity) $\exists x \,.\, \exists y \,.\, y \in x \wedge \forall y \in x \,.\, \exists w \in x \,.\, y \in w$.

Structure of the set of atoms

($\mathbb{A} = \bigcup \mathcal{A}$) $\forall a \,.\, a \in \mathbb{A} \Leftrightarrow \exists A \in \mathcal{A} \,.\, a \in A$.

(Disjointness) $\forall A, A' \in \mathcal{A} \,.\, \forall a \,.\, (a \in A \wedge a \in A') \Rightarrow A = A'$

($\mathbb{A}$ Not Finite) $\mathbb{A} \notin pow_{\text{fin}}(\mathbb{A})$

'Freshness' property

(Fresh) $\forall x \,.\, \forall A \in \mathcal{A} \,.\, \exists a \in A \,.\, a \#_A x$.

Here $a \#_A x$ stands for

$$a \in A \;\wedge\; \exists S \in pow_{\text{fin}}(A) \,.\, \forall a' \in A \,.\, a' \notin S \;\Rightarrow\; (a'\,a) \cdot_A x = x$$

where the term $pow_{\text{fin}}(\mathbb{A})$ denotes the set of finite subsets of $\mathbb{A}$ and has a standard set-theoretic definition. This formula also uses terms $(a'\,a) \cdot_A x$ for the transposition of A-atoms $a, a' \in A$ in x, which can be defined by $\in$-recursion:

$$(a'\,a) \cdot_A x \;=\; \begin{cases} a' & \text{if } x = a \\ a & \text{if } x = a' \\ x & \text{if } x \in \mathbb{A} \setminus \{a, a'\} \\ \{(a'\,a) \cdot_A y \mid y \in x\} & \text{if } x \notin \mathbb{A}. \end{cases}$$