J Hurd and T. Melham (Eds): TPHOLs 2005, LNCS 3603, pp. 17–34, 2005., pp. 17–34, 2005.

Alpha-Structural Recursion and Induction (Extended Abstract)

Andrew M. Pitts

University of Cambridge Computer Laboratory, Cambridge CB3 0FD, UK

Abstract. There is growing evidence for the usefulness of name *permutations* when dealing with syntax involving names and name-binding. In particular they facilitate an attractively simple formalisation of common, but often technically incorrect uses of structural recursion and induction for abstract syntax trees modulo α-equivalence. At the heart of this formalisation is the notion of *finitely supported* mathematical objects. This paper explains the idea in as concrete a way as possible and gives a new derivation within higher-order logic of principles of α-*structural* recursion and induction for α-equivalence classes from the ordinary versions of these principles for abstract syntax trees.

1 Introduction

> "They [previous approaches to operational semantics] do not in general have any great claim to being *syntax-directed* in the sense of defining the semantics of compound phrases in terms of the semantics of their components."
>
> —GD Plotkin, *A Structural Approach to Operational Semantics*, p 21 (Aarhus, 1981; reprinted as [18, p 32])

The above quotation and the title of the work from which it comes indicate the important role played by *structural recursion* and *structural induction* in programming language semantics. These are the forms of recursion and induction that fit the commonly used "algebraic" treatment of syntax. In this approach one specifies the syntax of a language at the level of abstract syntax trees (ASTs) by giving an *algebraic signature*. This consists of a collection of *sorts* s (one for each syntactic category of the language), and a collection of *constructors* K (also called "operators" or "function symbols"). Each such K comes with an *arity* consisting of a finite list $(\mathsf{s}_1, \ldots, \mathsf{s}_n)$ of sorts and with a *result-sort* s. Then the ASTs over the signature can be described by inductively generated terms t: if K has arity $(\mathsf{s}_1, \ldots, \mathsf{s}_n)$ and result sort s, and if t_i is a term of sort s_i for $i = 1..n$, then $K(t_1, \ldots, t_n)$ is a term of sort s. One gets off the ground in this inductive definition with the $n = 0$ instance of the rule for forming terms; this covers the case of *constants*, C, and one usually writes the term $C()$ just as C. Recursive definitions and inductive proofs about programs following the structure of their ASTs are both clearer and less prone to error than ones using non-structural methods. However, this treatment of syntax does not take into account the fact that most languages that one deals with in programming semantics involve *binding* constructors. In the presence of binders many syntax-manipulating operations only make sense, or at least only have

J Hurd and T. Melham (Eds): TPHOLs 2005, LNCS 3603, pp. 17–34, 2005., pp. 17–34, 2005.
© Springer-Verlag Berlin Heidelberg 2005

good properties, when we operate on syntax at a level of abstraction represented not by ASTs themselves, but by α-equivalence classes of ASTs.

It is true that this level of abstraction, which identifies terms differing only in the names of bound entities, can be reconciled with an algebraic treatment of syntax by using de Bruijn indexes [4]. The well-known disadvantage of this device is that it necessitates a calculus of operations on de Bruijn indexes that does not have much to do with our intuitive view of the structure of syntax. As a result there can be a big "coding gap" between statements of results involving binding syntax we would like to make and their de Bruijn versions; and (hence) it is easy to get the latter wrong. For this reason, de Bruijn-style representations of syntax may be more suitable for language implementations than for work on language semantics.

In any case, most of the work on semantics which is produced by humans rather than by computers sticks with ordinary ASTs involving explicit bound names and uses an informal approach to α-equivalence classes.[1] This approach is signalled by a form of words such as "we identify expressions up to α-equivalence" and means that: (a) occurrences of "t" now really mean its α-equivalence class "$[t]_\alpha$"; and (b) if the representative t for the class $[t]_\alpha$ is later used in some context where the particular bound names of t clash in some way with those in the context, then t will be changed to an α-variant whose bound names are fresh (i.e. ones not used in the current context). In other words it is assumed that the "Barendregt variable convention" [1, Appendix C] is maintained dynamically. In the literature, the ability to change bound names "on the fly" is usually justified by the assertion that final results of constructions involving ASTs are independent of choice of bound names. A fully formal treatment has to prove such independence results and in this paper we examine ways, arising from the results of [8, 16], to reduce the burden of such proofs.

However, proving that pre-existing functions respect α-equivalence is only part of the story; in most cases a prior (or simultaneous) problem is to prove the existence of the required functions in the first place. To see why, consider the familiar example of *capture-avoiding substitution* $(x := t)t'$ of a λ-term t for all free occurrences of a variable x in a λ-term t'. In the vernacular of programming semantics, we specify $(x := t)(-)$ by saying that it has the properties

$$(x := t)x_1 = \begin{cases} t & \text{if } x_1 = x \\ x_1 & \text{if } x_1 \neq x \end{cases} \tag{1}$$

$$(x := t)(t_1\, t_2) = (x := t)t_1\ (x := t)t_2 \tag{2}$$

$$(x := t)\lambda x_1.\, t_1 = \lambda x_1.\, (x := t)t_1 \quad \text{if } x_1 \neq x \text{ and } x_1 \text{ is not free in } t \tag{3}$$

where in the last clause there is no need to say what happens when $x_1 = x$ or when x_1 does occur freely in t, since we are working "up to α-equivalence" and can change $\lambda x_1.\, t_1$ to an α-variant satisfying these conditions. To see what this specification really amounts to, let us restore the usually-invisible notation for α-equivalence classes. Writing Λ for the set of λ-terms and $\Lambda/{=_\alpha}$ for its quotient by α-equivalence $=_\alpha$, then capture-avoiding substitution of $e \triangleq [t]_\alpha$ for x is a function $\hat{s}_{x,e} \in \Lambda/{=_\alpha} \to \Lambda/{=_\alpha}$.

[1] This includes the metatheory of "higher-order abstract syntax" [15], where the questions we are addressing are pushed up one meta-level to a single binding-form, λ-abstraction.

Every such function corresponds to a function $s_{x,e} \in \Lambda \to \Lambda/{=_\alpha}$ respecting $=_\alpha$, i.e. satisfying

$$t_1 =_\alpha t_2 \;\Rightarrow\; s_{x,e}(t_1) = s_{x,e}(t_2) \tag{4}$$

(enabling us to define $\hat{s}_{x,e}([t']_\alpha)$ as $[s_{x,e}(t')]_\alpha$). The requirements (1)–(3) mean that we want $s_{x,e}$ to satisfy:

$$s_{x,e}(x_1) = \begin{cases} e & \text{if } x_1 = x \\ [x_1]_\alpha & \text{if } x_1 \neq x \end{cases} \tag{5}$$

$$s_{x,e}(t_1\, t_2) = [t_1'\, t_2']_\alpha \quad \text{where } s_{x,e}(t_i) = [t_i']_\alpha \text{ for } i = 1, 2 \tag{6}$$

$$s_{x,e}(\lambda x_1.t_1) = [\lambda x_1.t_1']_\alpha \quad \begin{array}{l}\text{if } x_1 \neq x \text{ and } x_1 \text{ is not free in } e, \\ \text{and where } s_{x,e}(t_1) = [t_1']_\alpha.\end{array} \tag{7}$$

The problem is not one of proving that a certain well-defined function $s_{x,e}$ respects α-equivalence, but rather of proving that a function exists satisfying (4)—(7). Note that (5)—(7) do not constitute a definition of $s_{x,e}(t')$ by recursion on the structure of the AST t': even if we patch up the conditions in clauses (6) and (7) by using some enumeration of ASTs to make the choices t_i' definite functions of $s_{x,e}(t_i)$, the fact still remains that clause (7) only specifies what to do for certain pairs (x_1, t_1), rather than for all such pairs. Of course it is possible to complicate the specification of $s_{x,e}(\lambda x_1.t_1)$ by saying what to do when x_1 does occur freely in e and arrive at a construction for $s_{x,e}$ (either by giving up structural properties and using a less natural recursion on the height of trees; or by using structural recursion to define a more general operation of simultaneous substitution [21]). An alternative approach, and one that works with the original simple specification, is to construct functions by giving rule-based inductive definitions of their graphs (with the rules encoding the required properties of the function); one then has to prove (using rule-based induction) that the resulting relations are single-valued, total and respect $=_\alpha$. This is in principle a fully formal and widely applicable approach to constructing functions like $s_{x,e}$ (using tools that in any case are part and parcel of structural operational semantics), but one that is extremely tedious to carry out. It would be highly preferable to establish a recursion principle that goes straight from definitions like (1)–(3) to the existence of the function $(x := t)(-) \in \Lambda/{=_\alpha} \to \Lambda/{=_\alpha}$. We provide such a principle here for a general class of signatures in which binding information can be declared. We call it *α-structural recursion* and it comes with a derived induction principle, *α-structural induction*.

These recursion and induction principles for α-equivalence classes of ASTs are simplifications and generalisations of the ones introduced by Gabbay and the author in [8] as part of a new mathematical model of fresh names and name binding. That paper expresses its results in terms of a non-standard axiomatic set theory, based on the classical Fraenkel-Mostowski permutation model of set theory. Experience shows that this formalism impedes the take up within computer science of the new ideas contained in [8]. There is an essentially equivalent, but more concrete description of the model as standard sets equipped with some simple extra structure. These so-called *nominal sets* are introduced in [16] and I will use them here to express α-structural recursion and induction within "ordinary mathematics", or more precisely, within Church's higher-order logic [3].

2 Nominal Syntax

The usual principles of structural recursion and induction are parameterised by an algebraic signature that specifies the allowed constructors for forming ASTs of each sort. In order to state principles of recursion and induction for α-equivalence classes of ASTs, we need to fix a notion of signature that also specifies the forms of binding that occur in the ASTs. As explained in the Introduction, we stick with the usual "nominal" approach in which bound entities are explicitly named. Any generalisation of the notion of algebraic signature to encompass constructors that bind names needs to specify how bound occurrences of names in an AST are associated with a binding site further up the syntax tree. There are a number of such mechanisms in the literature of varying degrees of generality [10, 17, 5, 12, 22]. Here we will use the notion of *nominal signature* from [22]. It has the advantage of dealing with binding and α-equivalence independently of any considerations to do with variables, substitution and β-equivalence; bound names in a nominal signature may be of several different sorts and not just variables that can be substituted for. In common with the other cited approaches, nominal signatures only allow for constructors that bind a fixed number of names (and without loss of much generality, we can take that number to be one). There are certainly forms of binding occurring "in the wild" that do not fit comfortably into this framework.[2] I believe that the notion of α-structural recursion given here can be extended to cover more general forms of statically scoped binding; but for simplicity's sake I will stick with constructors binding a fixed number of names.

2.1 Atoms, Nominal Signatures and Terms

From a logical point of view[3], the names we use for making localised bindings in formal languages only need to be atomic, in the sense that the structure of names (of the same kind) is immaterial compared with the distinctions between names. Therefore we will use the term *atom* for such names. Throughout this paper we fix two sets: the set $\mathbb{A}$ of all **atoms** and the set $\mathbb{AS}$ of all **atom-sorts**. We also fix a function $sort \in \mathbb{A} \to \mathbb{AS}$ assigning sorts to atoms and assume that the sets $\mathbb{AS}$ and $\mathbb{A}_{\mathsf{a}} \triangleq \{a \in \mathbb{A} \mid sort(a) = \mathsf{a}\}$, for each $\mathsf{a} \in \mathbb{AS}$, are all countably infinite.

A **nominal signature** Σ consists of a subset $\Sigma_{\mathrm{A}} \subseteq \mathbb{AS}$ of atom-sorts, a set Σ_{D} of **data-sorts** and a set Σ_{C} of **constructors**. Each constructor $K \in \Sigma_{\mathrm{C}}$ comes with an **arity** σ and a **result sort** $\mathsf{s} \in \Sigma_{\mathrm{D}}$, and we write $K : \sigma \to \mathsf{s}$ to indicate this information. The arities σ of Σ are given as follows; at the same time we define the **terms**[4] t over Σ of each arity, writing $t : \sigma$ to indicate that t has arity σ.

Atoms: every atom-sort $\mathsf{a} \in \Sigma_{\mathrm{A}}$ is an arity. If $a \in \mathbb{A}_{\mathsf{a}}$ is an atom of sort a, then $a : \mathsf{a}$.
Constructed terms: every data-sort $\mathsf{s} \in \Sigma_{\mathrm{D}}$ is an arity. If $K : \sigma \to \mathsf{s}$ is in Σ_{C} and $t : \sigma$, then $K\,t : \mathsf{s}$.

[2] The full version of $F_{<:}$ with records and pattern-matching used in Part 2B of the "POPLMARK challenge" (`www.cis.upenn.edu/group/proj/plclub/mmm/`) is an example.

[3] As opposed to a pragmatic one that also encompasses issues of parsing and pretty-printing.

[4] Compared with [22, Definition 2.3] we only define *ground* terms, since we do not need to consider variables ranging over terms here.

Tuples: if $\sigma_1, \ldots, \sigma_n$ is a finite list of arities, then $\sigma_1 * \cdots * \sigma_n$ is an arity; the $n = 0$ case is called the **unit** arity and written 1. If $t_1 : \sigma_1, \ldots, t_n : \sigma_n$, then $\langle t_1, \ldots, t_n \rangle : \sigma_1 * \cdots * \sigma_n$; in particular, when $n = 0$ we have $\langle\rangle : 1$.

Atom-binding: if $\mathsf{a} \in \Sigma_{\mathrm{A}}$ and σ is an arity, then $\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma$ is an arity. If $a \in \mathbb{A}_{\mathsf{a}}$ and $t : \sigma$, then $\langle\!\langle a \rangle\!\rangle t : \langle\!\langle \mathsf{a} \rangle\!\rangle \sigma$.

We write $Ar(\Sigma)$ for the set of all arities over a nominal signature Σ, $\mathsf{T}(\Sigma)$ for the set of all terms over Σ, and $ar \in \mathsf{T}(\Sigma) \to Ar(\Sigma)$ for the function assigning to each term t the unique arity σ such that $t : \sigma$ holds. For each $\sigma \in Ar(\Sigma)$, we write $\mathsf{T}(\Sigma)_\sigma$ for the subset $\{t \in \mathsf{T}(\Sigma) \mid ar(t) = \sigma\}$ of terms of arity σ.

Example 1. Here is a nominal signature for the version of the Milner-Parrow-Walker π-calculus given in [19, Definition 1.1.1]. (The sort gsum is for processes that are guarded sums, and the sort pre is for prefixed processes.)

atom-sorts	data-sorts	constructors
chan	proc	$Gsum : \mathsf{gsum} \to \mathsf{proc}$
	gsum	$Par : \mathsf{proc} * \mathsf{proc} \to \mathsf{proc}$
	pre	$Res : \langle\!\langle \mathsf{chan} \rangle\!\rangle \mathsf{proc} \to \mathsf{proc}$
		$Rep : \mathsf{proc} \to \mathsf{proc}$
		$Zero : 1 \to \mathsf{gsum}$
		$Pre : \mathsf{pre} \to \mathsf{gsum}$
		$Plus : \mathsf{gsum} * \mathsf{gsum} \to \mathsf{gsum}$
		$Out : \mathsf{chan} * \mathsf{chan} * \mathsf{proc} \to \mathsf{pre}$
		$In : \mathsf{chan} * \langle\!\langle \mathsf{chan} \rangle\!\rangle \mathsf{proc} \to \mathsf{pre}$
		$Tau : 1 \to \mathsf{pre}$
		$Match : \mathsf{chan} * \mathsf{chan} * \mathsf{pre} \to \mathsf{pre}$

This example uses several atom- and data-sorts, but does not illustrate the usefulness of the inter-mixing of the arity-formers for tupling and atom-binding that is allowed in a nominal signature. An example that does do this is given in [22, Example 2.2]; see also the discussion in [10, Sect. 3].

2.2 Ordinary Structural Recursion and Induction

The terms over a nominal signature Σ are just the abstract syntax trees determined by an ordinary signature associated with Σ whose sorts are the arities of Σ, whose constructors are those of Σ, plus constructors for tupling and atom-binding, and with atoms regarded as particular constants. Consequently we can use ordinary structural recursion to define functions on the set $\mathsf{T}(\Sigma)$ of terms over Σ. We state without proof a simple, iterative form of the principle that we will be using later.

Theorem 2. *Let Σ be a nominal signature. Suppose we are given sets S_σ, for each $\sigma \in Ar(\Sigma)$, and functions*

$$\begin{aligned}
g_{\mathsf{a}} &\in \mathbb{A}_{\mathsf{a}} \to S_{\mathsf{a}} && (\mathsf{a} \in \Sigma_{\mathrm{A}}) \\
g_K &\in S_\sigma \to S_{\mathsf{s}} && ((K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}}) \\
g_{\sigma_1 * \cdots * \sigma_n} &\in S_{\sigma_1} \times \cdots \times S_{\sigma_n} \to S_{\sigma_1 * \cdots * \sigma_n} && (\sigma_i \in Ar(\Sigma) \mid i = 1..n) \\
g_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma} &\in \mathbb{A}_{\mathsf{a}} \times S_\sigma \to S_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma} && (\mathsf{a} \in \Sigma_{\mathrm{A}}, \sigma \in Ar(\Sigma)).
\end{aligned}$$

Then there is a unique family of functions $\overline{g}_\sigma \in \mathsf{T}(\Sigma)_\sigma \to S_\sigma$ ($\sigma \in Ar(\Sigma)$) satisfying the following properties

$$\overline{g}\, a = g_{\mathsf{a}}(a) \tag{8}$$
$$\overline{g}(K\, t) = g_K(\overline{g}\, t) \tag{9}$$
$$\overline{g}\langle t_1, \ldots, t_n\rangle = g_{\sigma_1 * \cdots * \sigma_n}\langle \overline{g}\, t_1, \ldots, \overline{g}\, t_n\rangle \tag{10}$$
$$\overline{g}\langle\!\langle a\rangle\!\rangle t = g_{\langle\!\langle \mathsf{a}\rangle\!\rangle \sigma}(a, \overline{g}\, t) \tag{11}$$

where we have abbreviated $\overline{g}_\sigma(t)$ to $\overline{g}\, t$ (since $\sigma = ar(t)$ is determined by t). □

Using the fact that subsets of $\mathsf{T}(\Sigma)$ are in bijection with functions $\mathsf{T}(\Sigma) \to \mathbb{B}$ (where $\mathbb{B} = \{T, F\}$ is the two-element set of boolean values), one can derive the following principle of structural induction for terms over Σ as a corollary of the uniqueness part of Theorem 2.

Corollary 3. *Let Σ be a nominal signature and $S \subseteq \mathsf{T}(\Sigma)$ a set of terms over Σ. To prove that S is the whole of $\mathsf{T}(\Sigma)$ it suffices to show*

$$(\forall \mathsf{a} \in \Sigma_{\mathrm{A}}, a \in \mathbb{A}_{\mathsf{a}})\ a \in S$$
$$(\forall (K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}}, t : \sigma)\ t \in S \ \Rightarrow\ K\, t \in S$$
$$(\forall (\sigma_i \in Ar(\Sigma), t_i : \sigma_i \mid i = 1..n))\ t_1 \in S\ \&\ \cdots\ \&\ t_n \in S \ \Rightarrow\ \langle t_1, \ldots, t_n\rangle \in S$$
$$(\forall \mathsf{a} \in \Sigma_{\mathrm{A}}, a \in \mathbb{A}_{\mathsf{a}}, \sigma \in Ar(\Sigma), t : \sigma)\ t \in S \ \Rightarrow\ \langle\!\langle a\rangle\!\rangle t \in S\ .$$ □

2.3 α-Equivalence and α-Terms

So far we have taken no account of the fact that atom-binder terms $\langle\!\langle a\rangle\!\rangle t$ should be identified up to renaming the bound atom a. Given a nominal signature Σ, the binary relation of **α-equivalence**, $t =_\alpha t' : \sigma$ (where $\sigma \in Ar(\Sigma)$ and $t, t' \in \mathsf{T}(\Sigma)_\sigma$) makes such identifications. It is inductively defined by the following rules. In rule ($=_\alpha$-4), $atm\, t$ indicates the finite set of atoms occurring in t; and $t\{a'/a\}$ indicates the term resulting from replacing all occurrences in t of the atom a by the atom a' (assumed to be of the same sort).

$$(=_\alpha\text{-}1)\ \frac{\mathsf{a} \in \Sigma_{\mathrm{A}} \qquad a \in \mathbb{A}_{\mathsf{a}}}{a =_\alpha a : \mathsf{a}} \qquad (=_\alpha\text{-}2)\ \frac{(K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}} \qquad t =_\alpha t' : \sigma}{K\, t =_\alpha K\, t' : \mathsf{s}}$$

$$(=_\alpha\text{-}3)\ \frac{t_1 =_\alpha t_1' : \sigma_1 \qquad \cdots \qquad t_n =_\alpha t_n' : \sigma_n}{\langle t_1, \ldots, t_n\rangle =_\alpha \langle t_1', \ldots, t_n'\rangle : \sigma_1 * \cdots * \sigma_n}$$

$$(=_\alpha\text{-}4)\ \frac{\mathsf{a} \in \Sigma_{\mathrm{A}} \qquad a, a', a'' \in \mathbb{A}_{\mathsf{a}} \qquad a'' \notin atm\,\langle a, t, a', t'\rangle \qquad t\{a''/a\} =_\alpha t'\{a''/a'\} : \sigma}{\langle\!\langle a\rangle\!\rangle t =_\alpha \langle\!\langle a'\rangle\!\rangle t' : \langle\!\langle \mathsf{a}\rangle\!\rangle \sigma}$$

Here we have generalised to terms over a nominal signature a version of the definition of α-equivalence of λ-terms [11, p. 36] that is conveniently syntax-directed compared

with the classic version [1, Definition 2.1.11]. It is easy to see that $=_\alpha$ is reflexive, symmetric and respects the various term-forming constructions for nominal syntax. Less straightforward is the fact that $=_\alpha$ is transitive. This can be proved in a number of ways. My favourite way makes good use of the techniques we will be using in Sect. 3, based on the action of atom-permutations on terms; see [16, Example 1].

For each $\sigma \in Ar(\Sigma)$, we write $\mathsf{T}_\alpha(\Sigma)_\sigma$ for the quotient of $\mathsf{T}(\Sigma)_\sigma$ by the equivalence relation $(-) =_\alpha (-) : \sigma$. Thus the elements of $\mathsf{T}_\alpha(\Sigma)_\sigma$ are α-equivalence classes of terms of arity σ; we write $[t]_\alpha$ for the class of t and refer to $[t]_\alpha$ as an **α-term** of arity σ over the nominal signature Σ.

3 Finite Support

The crucial ingredient in the formulation of structural recursion and induction for α-terms over a nominal signature is the notion of finite[5] *support*. It gives a well-behaved way, phrased in terms of atom-permutations, of expressing the fact that atoms are fresh for mathematical objects. It turns out to agree with the obvious definition when the objects are finite data such as abstract syntax trees, but allows us to deal with freshness for the not so obvious case of infinite sets and functions.

3.1 Nominal Sets

Let $Perm$ denote the set of all (finite, sort-respecting) **atom-permutations**; by definition, its elements are bijections $\pi : \mathbb{A} \leftrightarrow \mathbb{A}$ such that $\{a \in \mathbb{A} \mid \pi(a) \neq a\}$ is finite and $sort(\pi(a)) = sort(a)$ for all $a \in \mathbb{A}$. The operation of composing bijections gives a binary operation $\pi, \pi' \in Perm \mapsto \pi \circ \pi' \in Perm$ that makes $Perm$ into a group; we write ι for the identity atom-permutation and π^{-1} for the inverse of π. Among the elements of $Perm$ we single out **transpositions** $(a\ a')$ given by a pair of atoms of the same sort; $(a\ a')$ is the atom-permutation mapping a to a', mapping a' to a and leaving all other atoms fixed. It is a basic fact of group theory that every $\pi \in Perm$ is equal to a finite composition of such transpositions.

An **action** of $Perm$ on a set X is a function $Perm \times X \to X$, whose effect on $(\pi, x) \in Perm \times X$ we write as $\pi \cdot x$ (with X understood), and which is required to have the properties: $\iota \cdot x = x$ and $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$, for all $x \in X$ and $\pi, \pi' \in Perm$. Given such an action and an element $x \in X$, we say that a set $A \subseteq \mathbb{A}$ of atoms **supports** x if $(a\ a') \cdot x = x$ holds for all atoms a and a' (of the same sort) that are *not* in A. Then a **nominal set** is by definition a set X equipped with an action of $Perm$ such that every element $x \in X$ is supported by some *finite* set of atoms. If A_1 and A_2 are both finite sets of atoms supporting $x \in X$, then one can show that $A_1 \cap A_2$ also supports x. It follows that in a nominal set X, each element $x \in X$ possesses a *smallest* finite support, which we write as $supp_X(x)$ (or just $supp(x)$, if X is clear from the context) and call **the support of** x in X.

[5] Both Gabbay [7] and Cheney [2] develop more general notions of "small" supports. As Cheney's work shows, such a generalisation is necessary for some techniques of classical model theory to be applied; but finite supports are sufficient here.

Example 4. (i) Each set $\mathbb{A}_{\mathsf{a}}$ of atoms of a particular sort a is a nominal set once we endow it with the atom-permutation action given by $\pi \cdot a = \pi(a)$; as one might expect, $supp(a) = \{a\}$. It is not hard to see that the disjoint union of nominal sets is again a nominal set. So since the set of all atoms is the disjoint union of $\mathbb{A}_{\mathsf{a}}$ as a ranges over atom-sorts, $\mathbb{A}$ is a nominal set with atom-permutation action and support sets as for each individual $\mathbb{A}_{\mathsf{a}}$.

(ii) Let Σ be a nominal signature. Using Theorem 2 we can define an atom-permutation action on the sets $\mathsf{T}(\Sigma)_\sigma$ of terms over Σ of each arity $\sigma \in Ar(\Sigma)$:

$$\begin{array}{cc} \pi \cdot a = \pi(a) & \pi \cdot \langle t_1, \ldots, t_n \rangle = \langle \pi \cdot t_1, \ldots, \pi \cdot t_n \rangle \\ \pi \cdot K\, t = K(\pi \cdot t) & \pi \cdot \langle\!\langle a \rangle\!\rangle t = \langle\!\langle \pi \cdot a \rangle\!\rangle (\pi \cdot t)\ . \end{array}$$

Using Corollary 3 one can prove that this has the properties required of an atom-permutation action, that $a, a' \notin atm\, t \Rightarrow (a\ a') \cdot t = t$, and that $a \in atm\, t\ \&\ (a\ a') \cdot t = t \Rightarrow a = a'$. From these facts it follows that each $\mathsf{T}(\Sigma)_\sigma$ is a nominal set, with $supp(t) = atm\, t$, the finite set of atoms occurring in t.

(iii) Turning next to α-terms over Σ (Sect. 2.3), first note that the action of atom-permutations on terms preserves α-equivalence.[6] Therefore we get a well-defined action on α-terms by defining: $\pi \cdot [t]_\alpha = [\pi \cdot t]_\alpha$. For this action one finds that $\mathsf{T}_\alpha(\Sigma)_\sigma$ is a nominal set with $supp([t]_\alpha) = fa(t)$, the finite set of **free atoms** of any representative t of the class $[t]_\alpha$, defined (using Theorem 2) by:

$$\begin{array}{cc} fa(a) = \{a\} & fa(\langle t_1, \ldots, t_n \rangle) = fa(t_1) \cup \cdots \cup fa(t_n) \\ fa(K\, t) = fa(t) & fa(\langle\!\langle a \rangle\!\rangle t) = fa(t) - \{a\}\ . \end{array}$$

(iv) Each set S becomes a nominal set, called the **discrete nominal set** on S, if we endow it with the **trivial action** of atom-permutations, given by $\pi \cdot s = s$ for each $\pi \in Perm$ and $s \in S$; in this case the support of each element is empty. In particular, we will regard the set of booleans $\mathbb{B} = \{T, F\}$ and the set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ as nominal sets in this way.

3.2 Products and Functions

If $X_1, \ldots, X_n$ are nominal sets, then we get an action of atom-permutations on their cartesian product $X_1 \times \cdots \times X_n$ by defining $\pi \cdot (x_1, \ldots, x_n)$ to be $(\pi \cdot x_1, \ldots, \pi \cdot x_n)$, for each $(x_1, \ldots, x_n) \in X_1 \times \cdots \times X_n$. If A_i supports $x_i \in X_i$ for $i = 1..n$, then it is not hard to see that $A_1 \cup \cdots \cup A_n$ supports $(x_1, \ldots, x_n) \in X_1 \times \cdots \times X_n$; indeed, one can prove that $supp((x_1, \ldots, x_n)) = supp(x_1) \cup \cdots \cup supp(x_n)$. Thus $X_1 \times \cdots \times X_n$ is also a nominal set.

If X and Y are nominal sets, then we get an action of atom-permutations on the set $X \to Y$ of all functions from X to Y by defining $\pi \cdot f$ to be the function mapping each $x \in X$ to $\pi \cdot (f(\pi^{-1} \cdot x)) \in Y$. If you have not seen this definition before, it may look

[6] As noted in [16, p 169], this fact has nothing much to do with the particular nature of $=_\alpha$ and everything to do with the fact that it is inductively defined by a collection of schematic rules with the property that the action of any atom-permutation takes any instance of the rules to another instance.

more complicated than expected; however, it is forced by the important requirement that function application be respected by atom-permutations:[7]

$$\pi \cdot (f(x)) = (\pi \cdot f)(\pi \cdot x) \; . \tag{12}$$

Unlike the situation for cartesian product, not every element $f \in X \to Y$ is necessarily finitely supported with respect to this action (see Example 6 below). However, note that if f is supported by a finite set of atoms A, then $\pi \cdot f$ is supported by $\{\pi(a) \mid a \in A\}$. Therefore

$$X \to_{\mathrm{fs}} Y \triangleq \{f \in X \to Y \mid (\exists \,\text{finite}\, A \subseteq \mathbb{A})\; A \,\text{supports}\, f\}$$

is closed under the atom-permutation action and is a nominal set.

Example 5. Recall that the elements of $Perm$ are bijections from $\mathbb{A}$ to itself that respect sorts and leave fixed all but finitely many atoms. So each $\pi \in Perm$ is in particular a function $\mathbb{A} \to \mathbb{A}$. Regarding $\mathbb{A}$ as a nominal set as in Example 4(i), the action of atom-permutations on π *qua* function turns out to be the operation of *conjugation*: $\pi' \cdot \pi = \pi' \circ \pi \circ (\pi')^{-1}$. Hence the action of atom-permutations on $\mathbb{A} \to \mathbb{A}$ restricts to an action on $Perm$. One can prove that the finite set $\{a \in \mathbb{A} \mid \pi(a) \neq a\}$ supports π with respect to this action (and is in fact the smallest such set); so $Perm$ is a nominal set.

Example 6. Not every function between nominal sets is finitely supported. For example, since the set $\mathbb{A}$ of atoms is countable, there are surjective functions in $\mathbb{N} \to \mathbb{A}$; but it is not hard to see that any $f \in \mathbb{N} \to_{\mathrm{fs}} A$ must have a finite image (which is in fact the support of f). A more subtle example of a non-finitely-supported function is any **choice function** for the set $\mathbb{A}$ of atoms, i.e. any function $choose \in (\mathbb{A} \to_{\mathrm{fs}} \mathbb{B}) \to \mathbb{A}$ (where $\mathbb{B} = \{T, F\}$) satisfying $f(a) = T \;\Rightarrow\; f(choose(f)) = T$, for all $f \in \mathbb{A} \to_{\mathrm{fs}} \mathbb{B}$ and $a \in \mathbb{A}$.[8]

3.3 Freshness

Given an element of a nominal set, most of the time we are interested not so much in its support as in the (infinite) set of atoms that are *not* in its support. More generally, if $x \in X$ and $y \in Y$ are elements of nominal sets, we write $x \# y$ when $supp_X(x) \cap supp_Y(y) = \emptyset$ and say that x is **fresh** for y. Of the many properties of this notion of "freshness" developed in [8, 16] we single out the following one that we need below; it provides a very general criterion for when a construction that "picks a fresh atom" is independent of which fresh atom is chosen. (We omit the proof in this abstract.)

[7] More precisely, the definition of the action on functions is forced by the requirement that $X \to Y$ together with the usual application function be the *exponential* of X and Y in the cartesian closed category of sets equipped with an atom-permutation action.

[8] It was this lack of finite support for choice functions that motivated the original construction of the permutation model of set theory by Fraenkel and Mostowski.

Lemma 7 (Freshness Lemma). *Given an atom-sort* $\mathsf{a} \in \mathbb{AS}$ *and a nominal set* X*, if a finitely supported function* $h \in \mathbb{A}_\mathsf{a} \to_{\mathrm{fs}} X$ *satisfies*

$$(\exists a \in \mathbb{A}_\mathsf{a})\ a \# h\ \ \&\ \ a \# h(a) \tag{13}$$

then there is a unique element $\mathit{fresh}(h) \in X$ *satisfying*

$$(\forall a \in \mathbb{A}_\mathsf{a})\ a \# h\ \Rightarrow\ h(a) = \mathit{fresh}(h)\ . \tag{14}$$

□

4 Recursion and Induction for α-Terms

4.1 The Structure of α-Terms

Recall that $\mathsf{T}_\alpha(\Sigma)_\sigma$ denotes the set of α-terms of arity σ over a nominal signature Σ; by definition these are α-equivalence classes $[t]_\alpha$ of terms $t : \sigma$. Elementary properties of the relation $=_\alpha$ of α-equivalence yield the following structural properties of α-terms; at the same time we introduce some concrete syntax for α-terms mirroring the informal notation for α-equivalence classes mentioned in the Introduction.

Atoms: if $\mathsf{a} \in \Sigma_\mathrm{A}$ and $e \in \mathsf{T}_\alpha(\Sigma)_\mathsf{a}$, then there is a unique $a \in \mathbb{A}_\mathsf{a}$ such that $e = [a]_\alpha$. *In this case we write* e *just as* a.

Constructed α-terms: if $\mathsf{s} \in \Sigma_\mathrm{D}$ and $e \in \mathsf{T}_\alpha(\Sigma)_\mathsf{s}$, then there are unique $(K : \sigma \to \mathsf{s}) \in \Sigma_\mathrm{C}$ and $e' \in \mathsf{T}_\alpha(\Sigma)_\sigma$ such that there exists t' with $e' = [t']_\alpha$ and $e = [K\,t']_\alpha$. *In this case we write* e *as* $K\,e'$.

Tuples: if $\sigma_1, \ldots, \sigma_n \in Ar(\Sigma)$ and $e \in \mathsf{T}_\alpha(\Sigma)_{\sigma_1 * \cdots * \sigma_n}$, then there are unique $e_i \in \mathsf{T}_\alpha(\Sigma)_{\sigma_i}$ for $i = 1..n$ such that there exist t_i with $e_i = [t_i]_\alpha$ $(i = 1..n)$ and $e = [\langle t_1, \ldots, t_n \rangle]_\alpha$. *In this case we write* e *as* $(e_1, \ldots, e_n)$.

Atom-binding: if $\mathsf{a} \in \Sigma_\mathrm{A}$, $\sigma \in Ar(\Sigma)$ and $e \in \mathsf{T}_\alpha(\Sigma)_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}$, then for each $a \in \mathbb{A}_\mathsf{a}$ with $a \# e$ (i.e. with a not a free atom of e—cf. Example 4(iii)), there is a unique $e' \in \mathsf{T}_\alpha(\Sigma)_\sigma$ such that there exists t' with $e' = [t']_\alpha$ and $e = [\langle\!\langle a \rangle\!\rangle t']_\alpha$. *In this case we write* e *as* $a.\,e'$.

4.2 α-Structural Recursion and Induction

We can now state and prove the main result of this paper, a principle of structural recursion for α-terms over a nominal signature. Compared with Theorem 2, the principle uses nominal sets rather than ordinary sets, and requires a common finite support for the collection of functions in its hypothesis. Furthermore, the function supplied for each binding arity must satisfy a *freshness condition for binders* (FCB) saying, roughly, that for *some* sufficiently fresh choice of the atom being bound, the result of the function can never contain that atom in its support. These conditions ensure that there is a unique (finitely supported) arity-indexed family of functions that is well-defined on α-equivalence classes and satisfies the required recursion equations—for *all* sufficiently fresh bound atoms, in the case of the recursion equation for binders. The "*some*/*any*" aspect of the principle is a characteristic of the treatment of fresh names from [8].

Theorem 8 (α-Structural recursion). *Let Σ be a nominal signature. Suppose we are given an arity-indexed family of nominal sets X_σ ($\sigma \in Ar(\Sigma)$) and functions*

$$\begin{aligned}
f_{\mathsf{a}} &\in \mathbb{A}_{\mathsf{a}} \to_{\mathrm{fs}} X_{\mathsf{a}} && (\mathsf{a} \in \Sigma_{\mathrm{A}}) \\
f_K &\in X_\sigma \to_{\mathrm{fs}} X_{\mathsf{s}} && ((K : \sigma \to_{\mathrm{fs}} \mathsf{s}) \in \Sigma_{\mathrm{C}}) \\
f_{\sigma_1 * \cdots * \sigma_n} &\in X_{\sigma_1} \times \cdots \times X_{\sigma_n} \to_{\mathrm{fs}} X_{\sigma_1 * \cdots * \sigma_n} && (\sigma_i \in Ar(\Sigma) \mid i = 1..n) \\
f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma} &\in \mathbb{A}_{\mathsf{a}} \times X_\sigma \to_{\mathrm{fs}} X_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma} && (\mathsf{a} \in \Sigma_{\mathrm{A}}, \sigma \in Ar(\Sigma))
\end{aligned}$$

all of which are supported by a finite set of atoms A and satisfy the

> **Freshness Condition for Binders (FCB)**: *for each atom-binding arity $\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma \in Ar(\Sigma)$, the function $f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}$ satisfies $(\exists a' \in \mathbb{A}_{\mathsf{a}} - A)(\forall x \in X_\sigma)\ a' \# f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}(a', x)$.*

Then there is a unique family of finitely supported functions $\overline{f}_\sigma \in \mathsf{T}_\alpha(\Sigma)_\sigma \to_{\mathrm{fs}} X_\sigma$ ($\sigma \in Ar(\Sigma)$) with $supp(\overline{f}_\sigma) \subseteq A$ and satisfying the following properties for all $a, e, e_1, \ldots, e_n$ of suitable arity:

$$\overline{f} a = f_{\mathsf{a}}(a) \tag{15}$$

$$\overline{f}(K\, e) = f_K(\overline{f} e) \tag{16}$$

$$\overline{f}(e_1, \ldots, e_n) = f_{\sigma_1 * \cdots * \sigma_n}(\overline{f} e_1, \ldots, \overline{f} e_n) \tag{17}$$

$$a \notin A \;\Rightarrow\; \overline{f}(a.\, e) = f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}(a, \overline{f} e) \tag{18}$$

where we have abbreviated $\overline{f}_\sigma(e)$ to $\overline{f} e$ and used the notation for α-terms from Sect. 4.1.

Proof (sketch). We can reduce the proof of the theorem to an application of Theorem 2, taking advantage of the fact that we are working (informally) in higher-order logic.[9] From the $Ar(\Sigma)$-indexed family of nominal sets X_σ we define another such family: $S_\sigma \triangleq Perm \to_{\mathrm{fs}} X_\sigma$ (regarding $Perm$ as a nominal set as in Example 5 and using the $\to_{\mathrm{fs}}$ construct from Sect.3.2). Now define functions g_{a}, g_K, $g_{\sigma_1 * \cdots * \sigma_n}$ and $g_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}$ (with domains and codomains as in the statement of Theorem 2) as follows.

$$\begin{aligned}
g_{\mathsf{a}}\, a &\triangleq \lambda \pi \in Perm.\ f_{\mathsf{a}}(\pi(a)) \\
g_K\, s &\triangleq \lambda \pi \in Perm.\ f_K(s(\pi)) \\
g_{\sigma_1 * \cdots * \sigma_n}(s_1, \ldots, s_n) &\triangleq \lambda \pi \in Perm.\ f_{\sigma_1 * \cdots * \sigma_n}(s_1(\pi), \ldots, s_n(\pi)) \\
g_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}(a, s) &\triangleq \lambda \pi \in Perm.\ fresh(\lambda a' \in \mathbb{A}_{\mathsf{a}}.\, f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}(a', s(\pi \circ (a\ a'))))
\end{aligned}$$

The crucial clause in this definition is the last one, where we are using the *fresh* functional from Lemma 7 applied to the function $h \triangleq \lambda a' \in \mathbb{A}_{\mathsf{a}}.\, f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}(a', s(\pi \circ (a\ a')))$; in this abstract we omit the proof that the condition (13) needed to apply the lemma is satisfied in this case. Applying Theorem 2 with this data, we get a family of functions $\overline{g}_\sigma \in \mathsf{T}(\Sigma)_\sigma \to (Perm \to_{\mathrm{fs}} X_\sigma)$ satisfying the recursion equations (8)–(11) of that theorem. Next one proves that these functions respect α-equivalence; this is done by induction over the derivation of $t_1 =_\alpha t_2 : \sigma$ from the rules in Sect.2.3. So the functions $\overline{g}_\sigma$ induce functions $\overline{f}_\sigma \in \mathsf{T}_\alpha(\Sigma)_\sigma \to X_\sigma$ given by $\overline{f}_\sigma[t]_\alpha \triangleq \overline{g}_\sigma\, t\, \iota$ for any $t : \sigma$

[9] In other words the theorem is reducible to primitive recursion at higher types.

(recalling that ι stands for the identity permutation). One proves that these functions $\overline{f}_\sigma$ are all supported by A by first proving that the functions $\overline{g}_\sigma$ are so supported. Then the fact that the $\overline{f}_\sigma$ satisfy the required recursion equations (15)–(18) follows from the recursion equations (8)–(11) satisfied by the $\overline{g}_\sigma$. That concludes the existence part of the proof of Theorem 8.

For the uniqueness part, suppose functions $f'_\sigma \in \mathsf{T}_\alpha(\Sigma)_\sigma \to_{\mathrm{fs}} X_\sigma$ are all supported by A and satisfy the recursion equations (15)–(18) for $\overline{f}_\sigma$. Define $g'_\sigma \in \mathsf{T}(\Sigma)_\sigma \to S_\sigma$ by $g'_\sigma\, t\, \pi \triangleq f'_\sigma[\pi \cdot t]_\alpha$ ($\sigma \in Ar(\Sigma), t : \sigma, \pi \in Perm$). One can show that the g'_σ satisfy the same recursion equations (8)–(11) from Theorem 2 as the functions $\overline{g}_\sigma$; so by the uniqueness part of that theorem, $g'_\sigma = \overline{g}_\sigma$. Therefore for all $t : \sigma$, $f'_\sigma[t]_\alpha = f'_\sigma[\iota \cdot t]_\alpha \triangleq g'_\sigma\, t\, \iota = \overline{g}_\sigma\, t\, \iota \triangleq \overline{f}_\sigma[t]_\alpha$; hence $f'_\sigma = \overline{f}_\sigma$. □

Remark 9 ("Some/any" property). In Theorem 8 we gave the FCB as an existential statement. It is in fact equivalent to the universal statement $(\forall a' \in \mathbb{A}_{\mathsf{a}} - A)(\forall x \in X_\sigma)\ a' \# f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}(a', x)$. This is an instance of the characteristic "some/any" property of fresh names noted in [8, Proposition 4.10] and [16, Proposition 4].

Given a nominal set X, we can use the usual bijection between subsets of X and functions in $X \to \mathbb{B}$ (where $\mathbb{B} = \{T, F\}$) to transfer the action of atom-permutations on $X \to \mathbb{B}$ to one on subsets of X. This action sends $\pi \in Perm$ and $S \subseteq X$ to the subset $\pi \cdot S = \{\pi \cdot x \mid x \in S\}$. The nominal set $P_{\mathrm{fs}}(X)$ of **finitely supported subsets** of the nominal set X consists of all those subsets $S \subseteq X$ that are finitely supported with respect to this action. Thus $P_{\mathrm{fs}}(X)$ is isomorphic to $X \to_{\mathrm{fs}} \mathbb{B}$. Using this isomorphism, as a corollary of the uniqueness part of Theorem 8 we obtain the following principle of structural induction for α-terms.

Corollary 10 (α-Structural induction). *Let Σ be a nominal signature. Suppose we are given a finitely supported set $S \in P_{\mathrm{fs}}(\mathsf{T}_\alpha(\Sigma))$ of α-terms over Σ. To prove that S is the whole of $\mathsf{T}_\alpha(\Sigma)$ it suffices to show*

$$
\begin{array}{l}
(\forall \mathsf{a} \in \Sigma_{\mathrm{A}}, a \in \mathbb{A}_{\mathsf{a}})\ a \in S \\
(\forall (K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}}, e \in \mathsf{T}_\alpha(\Sigma)_\sigma)\ e \in S \ \Rightarrow\ K\, e \in S \\
(\forall (\sigma_i \in Ar(\Sigma), e_i \in \mathsf{T}_\alpha(\Sigma)_{\sigma_i} \mid i = 1..n))\ e_1 \in S \mathbin{\&} \cdots \mathbin{\&} e_n \in S \ \Rightarrow \\
\qquad\qquad (e_1, \ldots, e_n) \in S \\
(\forall \mathsf{a} \in \Sigma_{\mathrm{A}}, \sigma \in Ar(\Sigma))(\exists a \in \mathbb{A}_{\mathsf{a}} - \mathit{supp}(S))(\forall e \in \mathsf{T}_\alpha(\Sigma)_\sigma)\ e \in S \ \Rightarrow\ a.\, e \in S\ .
\end{array}
$$

□

Remark 11 (Primitive recursion). Theorem 8 gives a simple "iterative" form of structural induction for α-terms, rather than a more complicated "primitive recursive" form with recursion equations

$$
\begin{array}{cc}
\overline{f}\, a = f_{\mathsf{a}}(a) & \overline{f}(e_1, \ldots, e_n) = f_{\sigma_1 * \cdots * \sigma_n}(e_1, \ldots, e_n, \overline{f} e_1, \ldots, \overline{f} e_n) \\
\overline{f}(K\, e) = f_K(e, \overline{f} e) & \mathit{atm} \notin A \Rightarrow \overline{f}(a.\, e) = f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}(a, e, \overline{f} e).
\end{array}
$$

In fact this more general form can be deduced from the simple one given in the theorem.

4.3 "Sort-Directed" Recursion Principle

Theorem 8 is an "arity-directed" recursion principle for α-terms: one has to specify nominal sets X_σ for each arity σ, and give functions $f_{(\_)}$ for tuple and atom-binding binding arities in addition to ones for atoms and constructors. It is possible to derive a "sort-directed" version of the principle in which one only has to give X_σ when σ is a data-sort, and only has to give the functions $f_{(\_)}$ for constructors; the FCB has to be replaced by a more complicated family of conditions, indexed by the argument arities of constructors. In this extended abstract I will not formulate this version of the principle for an arbitrary nominal signature, but instead just give it for the particular case of untyped λ-calculus, for which the FCB is quite simple to state.

Let Σ^λ be the nominal signature with a single atom-sort v (for variables), a single data-sort t (for λ-terms), and constructors $Var : \mathsf{v} \to \mathsf{t}$, $App : \mathsf{t} * \mathsf{t} \to \mathsf{t}$ and $Lam : \langle\!\langle \mathsf{v} \rangle\!\rangle \mathsf{t} \to \mathsf{t}$. Thus $\mathsf{T}(\Sigma^\lambda)_\mathsf{t}$ is the usual set Λ of abstract syntax trees of λ-terms and $\mathsf{T}_\alpha(\Sigma^\lambda)_\mathsf{t}$ is the quotient $\Lambda/{=_\alpha}$ of that by the usual notion of α-equivalence—in other words $\mathsf{T}_\alpha(\Sigma^\lambda)_\mathsf{t}$ is what is normally meant by the set of all (open or closed) untyped λ-terms. The following "sort-directed" version of α-structural recursion for Σ^λ can be deduced by suitably instantiating $X_{(\_)}$ and $f_{(\_)}$ in Theorem 8.

Corollary 12 (α-Structural recursion for λ-terms). *Given a nominal set X and functions $h_{Var} \in \mathbb{A}_\mathsf{v} \to_{\mathrm{fs}} X$, $h_{App} \in X \times X \to_{\mathrm{fs}} X$ and $h_{Lam} \in \mathbb{A}_\mathsf{v} \times X \to_{\mathrm{fs}} X$, all supported by a finite set of atoms A and with h_{Lam} satisfying*

$$(\exists a' \in \mathbb{A}_\mathsf{v} - A)(\forall x \in X)\ a' \# h_{Lam}(a', x) \qquad \text{(FCB}')$$

then there is a unique finitely supported function $\overline{h} \in \Lambda/{=_\alpha} \to_{\mathrm{fs}} X$ with $supp(\overline{h}) \subseteq A$ and satisfying

$$\overline{h}(\mathit{Var}\, a) = h_{Var}(a) \tag{19}$$

$$\overline{h}(\mathit{App}\,(e_1, e_2)) = h_{App}(\overline{h}\, e_1, \overline{h}\, e_2) \tag{20}$$

$$a \notin A \;\Rightarrow\; \overline{h}(\mathit{Lam}\, a.\, e) = h_{Lam}(a, \overline{h}\, e)\ . \tag{21}$$

□

4.4 Applying the Principles

How do we use Theorem 8 in practice? Suppose that some language of interest has been specified as the α-terms for a particular nominal signature. Suppose that we wish to define a function on those α-terms specified by an instance of the recursion scheme (15)–(18) and we have identified suitable functions f_a, f_K, $f_{\sigma_1 * \cdots * \sigma_n}$ and $f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}$. Then there are three tasks involved in applying the theorem to this data:

(I) Show that the sets X_σ that we are mapping into have the structure of nominal sets.
(II) Show that the functions f_a, f_K, $f_{\sigma_1 * \cdots * \sigma_n}$ and $f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}$ are all supported by a single finite set of atoms A.
(III) Show that the functions $f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma}$ for atom-binding arities satisfy the FCB.

It is possible to dispose of tasks (I) and (II) by applying a single metatheorem about the notion of support, based on the fact that nominal sets form a model of higher-order logic (without choice functions—see Example 6). In the author's opinion, the best way of explaining this model is to use *topos theory* (see [13], for example). Call a function $f \in X \to Y$ between two nominal sets **equivariant** if it is supported by the empty set; in view of (12), this means that $\pi \cdot (f(x)) = f(\pi \cdot x)$, for all $\pi \in Perm$ and $x \in X$. Nominal sets and equivariant functions form a category that has the structure of a boolean topos with natural number object: products and exponentials are given by the operations $(-) \times (-)$ and $(-) \to_{\mathrm{fs}} (-)$ considered in Sect.3.2; the terminal object, subobject classifier and the natural number object are just the discrete nominal sets 1, $\mathbb{B}$ and $\mathbb{N}$ respectively (cf. Example 4(iv)). As for any such category, there is a sound interpretation of classical higher-order logic with arithmetic in this category. However, in this particular case the interpretation is easy to describe concretely: so long as we interpret function variables as ranging over only finitely supported functions, the usual set-theoretic interpretation of higher-order logic always yields finitely supported elements. If we remain within pure higher-order logic over ground types for numbers and booleans, then we only get elements with empty support. However, if we add a ground type for the set $\mathbb{A}$ of atoms, a constant for the function $sort \in \mathbb{A} \to \mathbb{AS}$ (taking $\mathbb{AS}$ to be a copy of $\mathbb{N}$) and constants for each atom, then the terms and formulas of higher-order logic describe functions and subsets which may have non-empty, finite support; such a "higher-order logic with atoms" has been developed by Gabbay [7]. Note that nominal sets of abstract syntax trees $\mathsf{T}(\Sigma)$ and their quotients by α-equivalence $\mathsf{T}_\alpha(\Sigma)$ are constructible within such a setting. As far as tasks (I) and (II) are concerned, we can sum things up thus: *if we use nominal sets and finitely supported functions in constructions definable in classical higher-order logic with arithmetic but without choice, the result will again be nominal sets and finitely supported functions.*

5 Examples

Here are some examples of Theorem 8 and Corollary 12 in action. In view of the above remarks, in each case we pass quickly over tasks (I) and (II) and concentrate on task (III).

Example 13 (Capture-avoiding substitution). The example mentioned in the Introduction of capture-avoiding substitution of λ-terms, $\hat{s}_{x,e} \in \Lambda/{=_\alpha} \to \Lambda/{=_\alpha}$, is obtained from Corollary 12 (using the nominal signature Σ^λ) by taking X to be the nominal set $\Lambda/{=_\alpha}$, i.e. $\mathsf{T}_\alpha(\Sigma^\lambda)_\mathsf{t}$. Given $x \in \mathbb{A}_\mathsf{v}$ and $e \in X$, then $\hat{s}_{x,e}$ is given by $\overline{h}$ where

$$h_{Var} \triangleq \lambda a \in \mathbb{A}_\mathsf{v}.\ \mathit{if}\ a = x\ \mathit{then}\ e\ \mathit{else}\ \mathit{Var}\, a \qquad h_{Lam} \triangleq \lambda(a,e) \in \mathbb{A}_\mathsf{v} \times X.\ \mathit{Lam}\, a.\, e$$
$$h_{App} \triangleq \lambda(e_1,e_2) \in X \times X.\ \mathit{App}(e_1,e_2) \qquad A \triangleq \mathit{supp}(x,e).$$

(FCB′) is satisfied because, as noted in Example 4(iii), for each $e \in X = \mathsf{T}_\alpha(\Sigma)_\mathsf{t}$, $\mathit{supp}(e)$ is the finite set of free atoms of e; in particular $a \# \mathit{Lam}\, a.\, e = h_{Lam}(a,e)$, because a is not free in (any representative of the α-equivalence class) $\mathit{Lam}\, a.\, e$. Note that the common finite support A of the $h_{(\_)}$ functions consists of x and the finite set of free variables of e. Therefore the restriction "$a \notin A$" in the recursion equation (21) corresponds precisely to the side-condition "$x_1 \neq x$ and x_1 is not free in e" in (7).

Example 14 (Length of an α-term). In [9, Sect. 3.3] Gordon and Melham give the usual recursion scheme for defining the length of a λ-term, remark that it is not a direct instance of the scheme developed in that paper (their Axiom 4) and embark on a detour via simultaneous substitutions to define the length function. This difficulty is analysed by Norrish [14, Sect. 3] on the way to his improved version of Gordon and Melham's recursion scheme (discussed further in Sect. 6). Pleasingly, the usual recursive definition of the length of a λ-term, or more generally of an α-term over any nominal signature, is a very simple application of α-structural recursion.[10] Thus in Theorem 8 we take X_σ to be the discrete nominal set $\mathbb{N}$ of natural numbers and

$$f_{\mathsf{a}} \triangleq \lambda a \in \mathbb{A}_{\mathsf{a}}.\ 1 \qquad f_{\sigma_1 * \cdots * \sigma_n} \triangleq \lambda (k_1, \ldots, k_n) \in \mathbb{N}^n.\ k_1 + \cdots + k_n$$
$$f_K \triangleq \lambda k \in \mathbb{N}.\ k + 1 \qquad f_{\langle\!\langle \mathsf{a} \rangle\!\rangle \sigma} \triangleq \lambda (a, k) \in \mathbb{A}_{\mathsf{a}} \times \mathbb{N}.\ k + 1$$

These functions are all supported by $A = \emptyset$ and the FCB holds trivially, because $a \# k$ holds for any $a \in \mathbb{A}$ and $k \in \mathbb{N}$. So the theorem gives us functions $\overline{f}_\sigma \in \mathsf{T}_\alpha(\Sigma)_\sigma \to_{\mathrm{fs}} \mathbb{N}$. Writing $\mathit{length}\, e$ for $\overline{f}_\sigma\, e$, we have the expected properties of a length function on α-terms:

$$\mathit{length}\, a = 1 \qquad \mathit{length}(e_1, \ldots, e_n) = \mathit{length}\, e_1 + \cdots + \mathit{length}\, e_n$$
$$\mathit{length}(K\, e) = \mathit{length}\, e + 1 \qquad \mathit{length}(a.\, e) = \mathit{length}\, e + 1\ .$$

Note that the last clause holds for all a, because in (18) the condition "$a \notin A$" is vacuously true (since $A = \emptyset$).

Example 15 (Recursion with "varying parameters"). Norrish [14, p 245] considers a variant $\mathtt{sub}'$ of capture-avoiding substitution whose definition involves recursion with varying parameters; it motivates the parametrised recursion principle he presents in that paper. The α-structural recursion principles we have given here do not involve parameters, let alone varying ones; nevertheless it is possible to derive parameterised versions from them. One can derive parameterised versions of ordinary structural recursion by currying parameters and defining maps into function sets using Theorem 2. In the presence of binders, one has to do something slightly more complicated, involving the Freshness Lemma 7, to derive the parameterised FCB from the unparameterised version of the condition.

Let us see how this works for Norrish's example. Using the nominal signature Σ^λ from Sect.4.3 (for which $\mathsf{T}_\alpha(\Sigma)_{\mathsf{t}}$ coincides with the nominal set $\Lambda/{=_\alpha}$ of α-equivalence classes of λ-terms) his sub' function can be expressed as follows. Fixing atoms $a_1, a_2 \in \mathbb{A}_{\mathsf{v}}$, we seek a function $s \in (\Lambda/{=_\alpha}) \to_{\mathrm{fs}} (\Lambda/{=_\alpha}) \to_{\mathrm{fs}} (\Lambda/{=_\alpha})$ satisfying:

$$s\,(\mathit{Var}\, a)\, e = \mathit{if}\ a = a_1\ \mathit{then}\ e\ \mathit{else}\ \mathit{Var}\, a \tag{22}$$
$$s\,(\mathit{App}(e_1, e_2))\, e = \mathit{App}(s\, e_1\, e, s\, e_2\, e) \tag{23}$$
$$a \# (a_1, a_2, e)\ \Rightarrow\ s\,(\mathit{Lam}\, a.\, e_1)\, e = \mathit{Lam}\, a.\, s\, e_1\, (\mathit{App}(\mathit{Var}\, a_2, e))\ . \tag{24}$$

[10] The same goes for Norrish's `stripc` function, used to illustrate the limitations of Gordon and Melham's workaround for the *length* function [14, p. 247].

If can be obtained from Corollary 12 as $s = \overline{h}$ if we take X to be the nominal set $(\Lambda/{=_\alpha}) \to_{\mathrm{fs}} (\Lambda/{=_\alpha})$ and use the functions

$$h_{Var} \triangleq \lambda a \in \mathbb{A}_{\mathsf{v}}.\lambda e \in (\Lambda/{=_\alpha}).\ \mathit{if}\ a = a_1\ \mathit{then}\ e\ \mathit{else}\ \mathit{Var}\, a$$
$$h_{App} \triangleq \lambda (x_1, x_2) \in X \times X.\ \lambda e \in (\Lambda/{=_\alpha}).\ \mathit{App}(x_1\, e, x_2\, e)$$
$$h_{Lam} \triangleq \lambda (a, x) \in \mathbb{A}_{\mathsf{v}} \times X.\lambda e \in (\Lambda/{=_\alpha}).\ \mathit{fresh}(h(a, x, e))$$

where the last clause uses Lemma 7 applied to $h(a, x, e) \triangleq \lambda a' \in \mathbb{A}_{\mathsf{v}}.\ \mathit{Lam}\, a'.((a\ a') \cdot x)(\mathit{App}(\mathit{Var}\, a_2, e)) \in \mathbb{A}_{\mathsf{v}} \to_{\mathrm{fs}} (\Lambda/{=_\alpha})$, which is easily seen to satisfy the property (13) needed for to apply lemma. Properties (19) and (20) of $\overline{h}$ give (22) and (23) respectively. When $a \neq a_1, a_2$, property (21) gives us $\overline{h}(\mathit{Lam}\, a.\, e_1) = h_{Lam}(a, \overline{h}\, e_1) = \mathit{fresh}(h(a, \overline{h}\, e_1, e))$. So if $a \# (a_1, a_2, e)$, picking any $a' \# (a_1, a_2, e, e_1, h)$, then by Lemma 7 we have $\mathit{fresh}(h(a, \overline{h}\, e_1, e)) = h(a, \overline{h}\, e_1, e)\, a' \triangleq$
$\mathit{Lam}\, a'.((a\ a') \cdot (\overline{h}\, e_1))(\mathit{App}(\mathit{Var}\, a_2, e)) = \mathit{Lam}\, a'.(a\ a') \cdot (\overline{h}\, e_1\, (\mathit{App}(\mathit{Var}\, a_2, e)))$. Hence by definition of $=_\alpha$, $\overline{h}(\mathit{Lam}\, a.\, e_1) = \mathit{Lam}\, a.\, \overline{h}\, e_1\, (\mathit{App}(\mathit{Var}\, a_2, e))$, as required for (24).

6 Assessment

Mathematical Perspective. The results of this paper are directly inspired by my joint work with Gabbay on "FM-set" theory [8] and by his PhD thesis [6]; in particular those works contain structural recursion and induction principles for an inductively defined FM-set isomorphic to λ-terms modulo α-equivalence. Here I have taken an approach that is both a bit more general and more concrete: more general, because the particular signature for λ-terms has been replaced by an arbitrary nominal signature (a notion which comes from joint work with Urban and Gabbay [22] and is developed further in Cheney's thesis [2]); and more concrete in two respects. First, the key notion of (finite) *support* has been developed using nominal sets within the framework of ordinary higher-order logic, rather than being axiomatised within FM-set theory; see Cheney [2, Chapter 3] for a more leisurely and generalised account of the theory of nominal sets. Secondly, rather than using an inductively defined nominal set that is isomorphic to the set of α-terms, the recursion and induction principles refer directly to α-terms, i.e. standard α-equivalence classes of abstract syntax trees. This is also the approach taken by Norrish [14], building on Gordon and Melham's five axioms for α-equivalence [9]; and also by Urban and Tasson [23]. Norrish's recursion principle [14, Fig. 1] has side-conditions requiring that the function being defined be well-behaved with respect to variable-permutations and with respect to fresh name generation. In effect these side-conditions build in just enough of the theory of nominal sets to yield a well-defined and total function, while only having to specify how binders with fresh names are mapped by the function. Along with Urban and Tasson [23], I prefer to develop the theory of nominal sets in its own right and then give a simple-looking (compare the statements of Theorems 2 and 8) recursion principle within that theory. One advantage of such an approach is that it makes it easier to identify and use properties of name *freshness*, such as Lemma 7, independently of the recursion principle. We used Lemma 7 in the reduction of Theorem 8 to Theorem 2 and in the reduction of "varying parameters" to

"no parameters" (Example 15); another good example of its use occurs (implicitly) in the denotational semantics of FreshML's `fresh` expression [20, Sect. 3].

Automated Theorem-Proving Perspective. How easy is it to apply these principles of α-structural recursion and induction? Just as for the work of Gordon-Melham, Norrish and Urban-Tasson, to use them one does not have to change to an unfamiliar logic (we remain in higher-order logic), or a new way of representing syntax (we use the familiar notion of α-equivalence classes of abstract syntax trees). One does have to get used to thinking in terms of permutations and finite support; and the latter is undoubtedly a subtle concept at higher types. However, the relativisation from arbitrary mathematical objects to finitely supported ones called for by this approach is made easier by the fact, noted in Sect. 4.4, that the finite support property is conserved by all the usual constructs of higher-order logic except for uses of the axiom of choice. Based also on my experience with other formalisms, I claim that the use of permutations and finitely supported objects is a simple, effective and yet rigorous way of dealing with binders and α-equivalence in "paper-and-pencil" proofs in programming language semantics. But how easy is it to provide computer support for reasoning with α-structural recursion and induction? In Sect.4.4, I mentioned the three types (I–III) of task involved in applying these principles in any particular case. Task (III) will require human-intervention; but in view of the meta-theorem mentioned in Sect.4.4, there is the possibility of making tasks (I) and (II) fully automatic. One way of attempting that is to develop a new higher-order logic in which types only denote nominal sets and that axiomatises properties of permutations and finite support; this is the route taken by Gabbay with his FM-HOL [7]. The disadvantage of such a "new logic" approach is that one does not have easy access to the legacy of already-proved results in systems such as HOL (`hol.sourceforge.net`) and Isabelle/HOL (`www.cl.cam.ac.uk/Research/HVG/Isabelle/`). To what extent tasks (I) and (II) can be automated within these "legacy" mechanised logics remains to be seen. The work of Norrish [14] provides a starting point within the HOL system; and Urban and Tasson [23] have already developed a theory equivalent to nominal sets within Isabelle/HOL up to and including what I am here calling α-structural induction for the particular nominal signature for λ-terms (but not yet α-structural recursion for that signature).[11] HOL and Isabelle/HOL feature type variables and predicative polymorphism. As a result, in principle it is possible to formulate and prove within such logics a result like Theorem 8 that makes a statement about *all* nominal signatures and *all* nominal sets. Of more use in practice would be would an augmentation of the HOL or Isabelle datatype packages, allowing the user to declare a nominal signature and then have the principles of α-structural recursion and induction for that signature proved and ready to be applied.[12]

Acknowledgements. I am grateful to James Cheney, Murdoch Gabbay, Michael Norrish, Mark Shinwell and Christian Urban for their many contributions to the subject of this paper.

[11] Their proof of validity of the induction principle follows a different route from the one used here to prove Theorem 8 and Corollary 10.

[12] How best in such a package to deal with the relativisation to nominal sets is certainly an issue; Urban and Tasson report that Isabelle's *axiomatic type classes* are helpful in this respect.

References

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

[2] J. Cheney. *Nominal Logic Programming*. PhD thesis, Cornell University, August 2004.

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

[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] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proc. LICS'99*, pages 193–202. IEEE Computer Society Press, 1999.

[6] M. J. Gabbay. *A Theory of Inductive Definitions with α-Equivalence: Semantics, Implementation, Programming Language*. PhD thesis, University of Cambridge, 2000.

[7] M. J. Gabbay. FM-HOL, a higher-order theory of names. In F. Kamareddine, editor, *Workshop on Thirty Five years of Automath, Informal Proceedings*. Heriot-Watt University, Edinburgh, Scotland, April 2002.

[8] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.

[9] A. D. Gordon and T. Melham. Five axioms of alpha-conversion. In *Proc. TPHOLS'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 173–191. Springer-Verlag, 1996.

[10] T. G. Griffin. Notational definition — a formal account. In *Proc. LICS'88*, pages 372–383. IEEE Computer Society Press, 1988.

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

[12] F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on nominal algebras in HOAS. In *Proc. ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 963–978. Springer-Verlag, 2001.

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

[14] M. Norrish. Recursive function definition for types with binders. In *Proc. TPHOLS 2004*, volume 3223 of *Lecture Notes in Computer Science*, pages 241–256. Springer-Verlag, 2004.

[15] F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proc. PLDI'88*, pages 199–208. ACM Press, 1988.

[16] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.

[17] G. D. Plotkin. An illative theory of relations. In R. Cooper, Mukai, and J. Perry, editors, *Situation Theory and its Applications, Volume 1*, volume 22 of *CSLI Lecture Notes*, pages 133–146. Stanford University, 1990.

[18] G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60–61:17–139, 2004.

[19] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[20] M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 2005. To appear.

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

[22] C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323:473–497, 2004.

[23] C. Urban and C. Tasson. Nominal techniques in Isabelle/HOL. In *Proc. CADE-20*, *Lecture Notes in Computer Science*, Springer-Verlag, 2005.