# Alpha-Structural Recursion and Induction

Andrew M. Pitts
Cambridge University Computer Laboratory
Cambridge CB3 0FD, UK

*Version distributed at the*
*International Summer School On Applied Semantics*
*Frauenchiemsee, Germany, 8-12 September 2005*

**Abstract**

The *nominal* approach to abstract syntax deals with the issues of bound names and $\alpha$-equivalence by considering constructions and properties that are invariant with respect to permuting names. The use of permutations gives rise to an attractively simple formalisation of common, but often technically incorrect uses of structural recursion and induction for abstract syntax modulo $\alpha$-equivalence. At the heart of this approach 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 $\alpha$-*structural* recursion and induction for $\alpha$-equivalence classes from the ordinary versions of these principles for abstract syntax trees.

# Index of Definitions

## 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 [23, 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* $\mathsf{s}$ (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* $\mathsf{s}$. 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 $\mathsf{s}$, and if $t_i$ is a term of sort $\mathsf{s}_i$ for $i = 1..n$, then $K(t_1, \ldots, t_n)$ is a term of sort $\mathsf{s}$. One gets off the ground in this inductive definition with the $n = 0$ instance of the rule for forming terms covering 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 good properties, when we operate on syntax at a level of abstraction represented not by ASTs themselves, but by $\alpha$-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 [6]. 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 things 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 $\alpha$-equivalence classes.[1] This approach is signalled by

[1]This includes the metatheory of "higher-order abstract syntax" [19], where the questions we are addressing are pushed up one meta-level to a single binding-form, $\lambda$-abstraction.

a form of words such as "we identify expressions up to $\alpha$-equivalence" and means that: (a) occurrences of "$t$" now really mean its $\alpha$-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 $\alpha$-variant whose bound names are fresh (i.e. are ones not used in the current context). In other words it is assumed that the "Barendregt variable convention" [2, 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 [13, 20], to reduce the burden of such proofs.

However, proving that pre-existing functions respect $\alpha$-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 $\lambda$-term $t$ for all free occurrences of a variable $x$ in a $\lambda$-term $t'$. To bring out the issues with binders more clearly, let us consider this operation not for the pure $\lambda$-calculus, but for an applied calculus that also has expressions for local recursive function declarations. Thus the terms are either variables ($x, f, y, \ldots$), applications ($t_1\, t_2$), function abstractions ($\lambda x.t$), or local recursive function declarations of the form $\mathit{letrec}\, f\, x = t_1\ \mathit{in}\, t_2$. The leftmost occurrence of the variable $f$ in $\mathit{letrec}\, f\, x = t_1\ \mathit{in}\, t_2$ binds all free occurrences of $f$ in both $t_1$ and $t_2$; whereas the left-most occurrence of the variable $x$ only binds free occurrences of $x$ in $t_1$. A systematic way of specifying such patterns of binding and the associated notion of $\alpha$-equivalence is given in Section 2.2; for the moment I assume the reader can supply a suitable definition of $\alpha$-equivalence for $\lambda$-terms involving such *letrec*-expressions.

How does one define capture-avoiding substitution for such terms up to $\alpha$-equivalence? In the vernacular of programming semantics, one might specify $(x := t)(-)$ by saying it has the following properties, where $\mathit{fv}(t)$ indicates the finite set of free variables of $t$.

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

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

$$y \notin \mathit{fv}(t) \cup \{x\} \;\Rightarrow\; (x := t)\lambda y.\, t_1 \;=\; \lambda y.\, (x := t)t_1 \tag{3}$$

$$\begin{aligned} &y \notin \mathit{fv}(t_2) \cup \{f\} \;\&\; f, y \notin \mathit{fv}(t) \cup \{x\} \;\Rightarrow \\ &\quad (x := t)\mathit{letrec}\, f\, y = t_1\ \mathit{in}\, t_2 \;=\; \mathit{letrec}\, f\, y = (x := t)t_1\ \mathit{in}\, (x := t)t_2 \end{aligned} \tag{4}$$

The condition on the equation in (3) should be familiar enough: there is no need to say what happens when $y$ occurs free in $t$ or when $y = x$, since we are working "up to $\alpha$-equivalence" and can change $\lambda y.\, t_1$ to an $\alpha$-variant satisfying these conditions.

The same goes for the more complicated condition on the equation in (4): given $x$ and $t$, we can change $\mathit{letrec}\, f\, y = t_1\, \mathit{in}\, t_2$ up to $\alpha$-equivalence to ensure that $f$ and $y$ are distinct variables not occurring free in $t$ and not equal to $x$; but we can also assume that $y$ does not occur free in $t_2$, since that term lies outside the binding scope of $y$ in the term $\mathit{letrec}\, f\, y = t_1\, \mathit{in}\, t_2$.

To see what this specification of $(x := t)(-)$ really amounts to, let us restore the usually-invisible notation $[t]_\alpha$ for the $\alpha$-equivalence class of a term $t$. Writing $\Lambda$ for the set of terms and $\Lambda/{=_\alpha}$ for its quotient by $\alpha$-equivalence $=_\alpha$, then capture-avoiding substitution of an $\alpha$-equivalence class $e$ for a variable $x$ is a function $\hat{s}_{x,e} \in \Lambda/{=_\alpha} \to \Lambda/{=_\alpha}$. Every such function corresponds to a function $s_{x,e} \in \Lambda \to \Lambda/{=_\alpha}$ respecting $=_\alpha$, i.e. satisfying

$$t_1 =_\alpha t_2 \;\Rightarrow\; s_{x,e}(t_1) = s_{x,e}(t_2) \tag{5}$$

(enabling us to define $\hat{s}_{x,e}([t]_\alpha)$ as $[s_{x,e}(t)]_\alpha$). The requirements (1)–(4) mean that we want $s_{x,e}$ to satisfy:

$$s_{x,e}(y) = \begin{cases} e & \text{if } y = x \\ [y]_\alpha & \text{if } y \neq x \end{cases} \tag{6}$$

$$s_{x,e}(t_1\, t_2) = [t'_1\, t'_2]_\alpha \quad \text{where } s_{x,e}(t_i) = [t'_i]_\alpha \text{ for } i = 1, 2 \tag{7}$$

$$\begin{aligned} & y \notin \mathit{fv}(e) \cup \{x\} \;\Rightarrow \\ & \qquad s_{x,e}(\lambda y.t_1) = [\lambda y.t'_1]_\alpha \quad \text{where } s_{x,e}(t_1) = [t'_1]_\alpha \end{aligned} \tag{8}$$

$$\begin{aligned} & y \notin \mathit{fv}(t_2) \cup \{f\} \;\&\; f, y \notin \mathit{fv}(e) \cup \{x\} \;\Rightarrow \\ & \qquad s_{x,e}(\mathit{letrec}\, f\, y = t_1\, \mathit{in}\, t_2) = [\mathit{letrec}\, f\, y = t'_1\, \mathit{in}\, t'_2]_\alpha \\ & \qquad \text{where } s_{x,e}(t_i) = [t'_i]_\alpha \text{ for } i = 1, 2. \end{aligned} \tag{9}$$

The problem is not one of proving that a certain well-defined function $s_{x,e}$ respects $\alpha$-equivalence, but rather of proving that a function exists satisfying (5)—(9). Note that (6)—(9) 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 "where" conditions in clauses (7)–(9) 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 clauses (8) and (9) and only specify what to do for certain pairs $(y, t_1)$, rather than for all such pairs. Of course it is possible to complicate the specification by saying what to do for $\lambda$- and $\mathit{letrec}$-terms that do not meet the preconditions in (8) and (9), thereby arriving at a way of constructing $s_{x,e}(t)$ for any $t$ (either by giving up structural properties and using a less natural recursion on the height of trees; or by fixing an enumeration of variables and using structural recursion to define a more general operation of simultaneous substitution [27]). 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)–(4) 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 *$\alpha$-structural recursion* and it comes with an associated induction principle, *$\alpha$-structural induction*.

These recursion and induction principles for $\alpha$-equivalence classes of ASTs are simplifications and generalisations of the ones introduced by Gabbay and the author in [13] as part of a new mathematical model of fresh names and name binding. That paper expresses its results in terms of an axiomatic set theory, based on the classical Fraenkel-Mostowski permutation model of set theory. In my experience this formalism impedes the take up within computer science of the new ideas contained in [13]. 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 [20] and I will use them here to express $\alpha$-structural recursion and induction within "ordinary mathematics", or more precisely, within Church's higher-order logic [5].

## How to read this paper

Having read the Introduction this far, impatient readers may wish to turn to Theorem 25 to see the statement of the $\alpha$-structural recursion principle for $\lambda$-calculus with *letrec*-terms and how it is used to define $(x := t)(-) \in \Lambda/{=_\alpha} \to \Lambda/{=_\alpha}$ satisfying (1)–(4). To understand the statement of this theorem, they must then look up the definitions of "nominal set", "finite support" and the freshness relation $(-) \mathrel{\#} (-)$ in Section 3. This recursion principle and the corresponding induction principle are generalised to arbitrary signatures with binding information in Section 5. The particular way of specifying binding information that we use (via *nominal signatures* [28]) is explained in Section 2. Section 4 gives a first, simple version of the $\alpha$-structural recursion and induction principles that are derived from ordinary structural recursion/induction for ASTs by, roughly speaking, taking into account an implicit parameterisation by name-permutations. The reduction of the practically more useful principles of Section 5 to the simpler ones of Section 4 is quite involved and is relegated to the Appendices. Section 6 contains an extended example (on *normalisation by evaluation* for the simply-typed $\lambda$-calculus [3]) that not only uses $\alpha$-structural recursion and induction, but also shows off some of the power of nominal sets and the notion of freshness of names that they support. The final Section 7 assesses this paper's "nominal" approach to abstract syntax in the context of related work, both from a mathematical perspective and from the perspective of automated theorem proving.

## 2 Nominal Syntax

The usual principles of structural recursion and induction are parameterised by an algebraic signature that specifies the allowed constructors for forming abstract syntax trees (ASTs) of each sort. In order to state principles of recursion and induction for $\alpha$-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, 15, 17, 22, 28]. Here we will use the notion of *nominal signature* [28]. It has the advantage of dealing with binding and $\alpha$-equivalence independently of any considerations to do with variables, substitution and $\beta$-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 (for example, in the full version of $F_{<:}$ with records and pattern-matching used in Part 2B of the "POPLMARK challenge" [1]). I believe that the notion of $\alpha$-structural recursion given here can be extended to cover more general forms of statically scoped binding, such as those used by Pottier in his C$\alpha$ml library [24]; but for simplicity's sake I will stick with constructors binding a fixed number of names.

### 2.1 Atoms

From a logical point of view (as opposed to a pragmatic one that also encompasses issues of parsing and pretty-printing), 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.

### 2.2 Nominal signatures

A **nominal signature** $\Sigma$ consists of a subset of the atom-sorts, $\Sigma_{\mathrm{A}} \subseteq \mathbb{AS}$, a set $\Sigma_{\mathrm{D}}$ of **data-sorts** and a set $\Sigma_{\mathrm{C}}$ of **constructors**. Each constructor $K \in \Sigma_{\mathrm{C}}$ comes with an **arity** $\sigma$ and a **result sort** $\mathsf{s} \in \Sigma_{\mathrm{D}}$, and we write $K : \sigma \to \mathsf{s}$ to indicate this information. The arities $\sigma$ of $\Sigma$ are given as follows:

**Atom-sorts:** every atom-sort $\mathsf{a} \in \Sigma_{\mathrm{A}}$ is an arity.

**Data-sorts:** every data-sort $\mathsf{s} \in \Sigma_{\mathrm{D}}$ is an arity.

**Unit arity:** $1$ is an arity.

**Pair arities:** if $\sigma_1$ and $\sigma_2$ are arities, then $\sigma_1 * \sigma_2$ is an arity.

**Atom-binding arities:** if $\mathsf{a} \in \Sigma_{\mathrm{A}}$ and $\sigma$ is an arity, then $«\mathsf{a}»\sigma$ is an arity.

The **terms** $t$ over $\Sigma$ of each arity are defined as follows, where we write $t : \sigma$ to indicate that $t$ has arity $\sigma$.[2]

**Atoms:** If $a \in \mathbb{A}_{\mathsf{a}}$ is an atom of sort $\mathsf{a}$, then $a : \mathsf{a}$.

**Constructed terms:** If $K : \sigma \to \mathsf{s}$ is in $\Sigma_{\mathrm{C}}$ and $t : \sigma$, then $K\,t : \mathsf{s}$.

**Unit:** $\langle\rangle : 1$ is the unique term of unit arity.

**Pairs:** If $t_1 : \sigma_1$ and $t_2 : \sigma_n$, then $\langle t_1, t_2 \rangle : \sigma_1 * \sigma_2$.

**Atom-binding:** If $a \in \mathbb{A}_{\mathsf{a}}$ and $t : \sigma$, then $«a»t : «\mathsf{a}»\sigma$.

We write $Ar(\Sigma)$ for the set of all arities over a nominal signature $\Sigma$, $\mathsf{T}(\Sigma)$ for the set of all terms over $\Sigma$, and $ar \in \mathsf{T}(\Sigma) \to Ar(\Sigma)$ for the function assigning to each term $t$ the unique arity $\sigma$ 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 $\sigma$.

**Example 1 ($\boldsymbol{\lambda}$-calculus with letrec).** Here is a nominal signature for the untyped $\lambda$-calculus [2]. There is a single atom-sort $\mathsf{v}$ for variables, and a single data-sort $\mathsf{t}$ for $\lambda$-terms.

| atom-sorts | data-sorts | constructors |
|---|---|---|
| $\mathsf{v}$ | $\mathsf{t}$ | $V : \mathsf{v} \to \mathsf{t}$ |
| | | $A : \mathsf{t} * \mathsf{t} \to \mathsf{t}$ |
| | | $L : «\mathsf{v}»\mathsf{t} \to \mathsf{t}$ |

To illustrate the inter-mixing of the arity-formers for pairing and atom-binding that is allowed in a nominal signature, consider augmenting $\lambda$-calculus with the local recursive function declarations, $\mathit{letrec}\, f\, x = t_1\ \mathit{in}\, t_2$, that were used in the discussion of capture-avoiding substitution in the Introduction. Recall that free occurrences of $x$ in $t_1$ are bound in $\mathit{letrec}\, f\, x = t_1\ \mathit{in}\, t_2$; and free occurrences of $f$ in either of $t_1$ or $t_2$ are bound in the term. To get the effect of this we can add to the above nominal signature a constructor

$$\mathit{Letrec} : «\mathsf{v}»((«\mathsf{v}»\mathsf{t}) * \mathsf{t}) \to \mathsf{t}\ .$$

[2]Compared with [28, Definition 2.3] we only define *ground* terms, since we do not need to consider variables ranging over terms here.

So for example, the expression $\mathit{letrec}\, f\, x \;=\; f\, x\, \mathit{in}\, f(\lambda y.y)$ corresponds to the nominal term

$$\mathit{Letrec}«f»\langle «x» A\langle V f,\, V x\rangle, A\langle V f, L«y» V y\rangle\rangle$$

of arity $\mathsf{t}$ over this signature (where $f, x, y \in \mathbb{A}_{\mathsf{v}}$).

**Example 2 (*π*-calculus).** Here is a nominal signature for the version of the Milner-Parrow-Walker $\pi$-calculus given in [25, Definition 1.1.1]. There is an atom-sort chan for channel names and a data-sort proc for process expressions; but there are also auxiliary data-sorts gsum, for processes that are guarded sums, and pre, for prefixed processes.

| atom-sorts | data-sorts | constructors |
|---|---|---|
| chan | proc | $\mathit{Gsum} : \mathsf{gsum} \longrightarrow \mathsf{proc}$ |
| | gsum | $\mathit{Par} : \mathsf{proc} * \mathsf{proc} \longrightarrow \mathsf{proc}$ |
| | pre | $\mathit{Res} : «\mathsf{chan}»\mathsf{proc} \longrightarrow \mathsf{proc}$ |
| | | $\mathit{Rep} : \mathsf{proc} \longrightarrow \mathsf{proc}$ |
| | | $\mathit{Zero} : 1 \longrightarrow \mathsf{gsum}$ |
| | | $\mathit{Pre} : \mathsf{pre} \longrightarrow \mathsf{gsum}$ |
| | | $\mathit{Plus} : \mathsf{gsum} * \mathsf{gsum} \longrightarrow \mathsf{gsum}$ |
| | | $\mathit{Out} : (\mathsf{chan} * \mathsf{chan}) * \mathsf{proc} \longrightarrow \mathsf{pre}$ |
| | | $\mathit{In} : \mathsf{chan} * «\mathsf{chan}»\mathsf{proc} \longrightarrow \mathsf{pre}$ |
| | | $\mathit{Tau} : \mathsf{proc} \longrightarrow \mathsf{pre}$ |
| | | $\mathit{Match} : (\mathsf{chan} * \mathsf{chan}) * \mathsf{pre} \longrightarrow \mathsf{pre}$ |

For example, the $\pi$-calculus process expression $\nu x((\overline{x}u.0 + \overline{y}v.0)|x(z).\overline{z}w.0)$ corresponds to the following nominal term of arity proc over this signature (where $x, u, y, v, z, w \in \mathbb{A}_{\mathsf{chan}}$):

$$\begin{aligned}\mathit{Res}«x»\mathit{Par}\langle \mathit{Gsum}\,\mathit{Plus}\langle &\mathit{Pre}\,\mathit{Out}\langle\langle x, u\rangle, \mathit{Gsum}\,\mathit{Zero}\langle\rangle\rangle,\\ &\mathit{Pre}\,\mathit{Out}\langle\langle y, v\rangle, \mathit{Gsum}\,\mathit{Zero}\langle\rangle\rangle\rangle,\\ &\mathit{Gsum}\,\mathit{Pre}\,\mathit{In}\langle x, «z»\mathit{Gsum}\,\mathit{Pre}\,\mathit{Out}\langle\langle z, w\rangle, \mathit{Gsum}\,\mathit{Zero}\langle\rangle\rangle\rangle\rangle.\end{aligned}$$

## 2.3 Ordinary structural recursion and induction

The terms over a nominal signature $\Sigma$ are just the abstract syntax trees determined by an ordinary signature associated with $\Sigma$ whose sorts are the arities of $\Sigma$, whose constructors are those of $\Sigma$, plus constructors for unit, pairs 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 $\Sigma$; and we can use ordinary structural induction to prove properties of those terms. The following two theorems give versions of these principals that we use later. We regard their proofs as standard and omit them.[3]

[3]Each theorem can be used to prove the other; and either of them can be proved using induction for the natural numbers once one has fixed upon a particular construction of abstract syntax trees.

**Theorem 3 (structural recursion for nominal terms).** *Let $\Sigma$ be a nominal signature. Suppose we are given sets $S_\sigma$, for each $\sigma \in Ar(\Sigma)$, and elements*

$$\begin{aligned}
g_{\mathsf{a}} &\in \mathbb{A}_{\mathsf{a}} \to S_{\mathsf{a}} && (\mathsf{a} \in \Sigma_{\mathrm{A}})\\
g_K &\in S_\sigma \to S_{\mathsf{s}} && ((K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}})\\
g_{\mathsf{1}} &\in S_{\mathsf{1}} && \\
g_{\sigma_1 * \sigma_2} &\in S_{\sigma_1} \times S_{\sigma_2} \to S_{\sigma_1 * \sigma_2} && (\sigma_1, \sigma_2 \in Ar(\Sigma))\\
g_{«\mathsf{a}»\sigma} &\in \mathbb{A}_{\mathsf{a}} \times S_\sigma \to S_{«\mathsf{a}»\sigma} && (\mathsf{a} \in \Sigma_{\mathrm{A}}, \sigma \in Ar(\Sigma))
\end{aligned}$$

(We write $X \times Y$ for the cartesian product of two sets $X$ and $Y$; and write $X \to Y$ for the set of functions from $X$ to $Y$.) *Then there is a unique family of functions $(\hat{g}_\sigma \in \mathsf{T}(\Sigma)_\sigma \to S_\sigma \mid \sigma \in Ar(\Sigma))$ satisfying the following properties*

$$\hat{g}\, a = g_{\mathsf{a}}(a) \tag{10}$$

$$\hat{g}(K\, t) = g_K(\hat{g}\, t) \tag{11}$$

$$\hat{g}\langle\rangle = g_{\mathsf{1}} \tag{12}$$

$$\hat{g}\langle t_1, t_2\rangle = g_{\sigma_1 * \sigma_2}\langle \hat{g}\, t_1, \hat{g}\, t_2\rangle \tag{13}$$

$$\hat{g}\, «a»t = g_{«\mathsf{a}»\sigma}(a, \hat{g}\, t) \tag{14}$$

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

**Theorem 4 (structural induction for nominal terms).** *Let $\Sigma$ be a nominal signature and $S \subseteq \mathsf{T}(\Sigma)$ a set of terms over $\Sigma$. To prove that $S$ is the whole of $\mathsf{T}(\Sigma)$ it suffices to show*

$$(\forall \mathsf{a} \in \Sigma_{\mathrm{A}}, a \in \mathbb{A}_{\mathsf{a}})\ a \in S \tag{15}$$

$$(\forall (K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}}, t : \sigma)\ t \in S \ \Rightarrow\ K\, t \in S \tag{16}$$

$$\langle\rangle \in S \tag{17}$$

$$(\forall (\sigma_i \in Ar(\Sigma), t_i : \sigma_i \mid i = 1, 2))\ t_1 \in S \ \&\ t_2 \in S \ \Rightarrow\ \langle t_1, t_2\rangle \in S \tag{18}$$

$$(\forall \mathsf{a} \in \Sigma_{\mathrm{A}}, a \in \mathbb{A}_{\mathsf{a}}, \sigma \in Ar(\Sigma), t : \sigma)\ t \in S \ \Rightarrow\ «a»t \in S\ . \tag{19}$$

□

## 2.4 $\alpha$-Equivalence and $\alpha$-terms

So far we have taken no account of the fact that atom-binding terms $«a»t$ should be identified up to renaming the bound atom $a$. Given a nominal signature $\Sigma$, the relation of **$\boldsymbol{\alpha}$-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 rules in Figure 1. They generalise to terms over a nominal signature a version of the definition of $\alpha$-equivalence of $\lambda$-terms [16, p. 36] that is conveniently syntax-directed compared with the classic version [2, 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

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

$$(=_\alpha\text{-}3)\ \frac{}{\langle\rangle =_\alpha \langle\rangle : 1} \qquad (=_\alpha\text{-}4)\ \frac{t_1 =_\alpha t_1' : \sigma_1 \qquad t_2 =_\alpha t_2' : \sigma_2}{\langle t_1, t_2\rangle =_\alpha \langle t_1', t_2'\rangle : \sigma_1 * \sigma_2}$$

$$(=_\alpha\text{-}5)\ \frac{\begin{array}{c}\mathsf{a} \in \Sigma_{\mathrm{A}} \qquad a, a', a'' \in \mathbb{A}_{\mathsf{a}} \qquad a'' \notin \mathit{atm}\,\langle a, t, a', t'\rangle \\ t\{a''/a\} =_\alpha t'\{a''/a'\} : \sigma\end{array}}{«a»t =_\alpha «a'»t' : «\mathsf{a}»\sigma}$$

In rule $(=_\alpha\text{-}5)$, $\mathit{atm}\,t$ 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); both can defined by structural recursion (Theorem 3):

$$\begin{array}{rcl} \mathit{atm}\,a &=& \{a\} \\ \mathit{atm}(K\,t) &=& \mathit{atm}\,t \\ \mathit{atm}\langle\rangle &=& \emptyset \\ \mathit{atm}\langle t_1, t_2\rangle &=& \mathit{atm}\,t_1 \cup \mathit{atm}\,t_2 \\ \mathit{atm}\,«a»t &=& \{a\} \cup \mathit{atm}\,t \end{array} \qquad \begin{array}{rcl} a''\{a'/a\} &=& \begin{cases} a' & \text{if } a'' = a \\ a'' & \text{if } a'' \neq a \end{cases} \\ (K\,t)\{a'/a\} &=& K(t\{a'/a\}) \\ \langle\rangle\{a'/a\} &=& \langle\rangle \\ \langle t_1, t_2\rangle\{a'/a\} &=& \langle t_1\{a'/a\}, t_2\{a'/a\}\rangle \\ («a''»t)\{a'/a\} &=& «a''\{a'/a\}»t\{a'/a\} \end{array}$$

Figure 1: $\alpha$-Equivalence of nominal terms

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 later, based on the action of atom-permutations on terms; see [20, Example 1].

**Definition 5.** 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 $\alpha$-equivalence classes of terms of arity $\sigma$; we write $[t]_\alpha$ for the class of $t$ and refer to $[t]_\alpha$ as an ***α*-term** of arity $\sigma$ over the nominal signature $\Sigma$.

## 3 Finite Support

The crucial ingredient in the formulation of structural recursion and induction for $\alpha$-terms over a nominal signature is the notion of finite[4] *support*. It gives a well-behaved way, phrased in terms of atom-permutations, of expressing the fact that

[4] Both Gabbay [12] and Cheney [4] 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 suffice here.

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. For example, the identity function on $\mathbb{A}$ "mentions" every atom in its graph; nevertheless, it has empty support and any atom is fresh for it.

## 3.1 Nominal sets

Let $\mathit{Perm}$ 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 $\mathit{sort}(\pi(a)) = \mathit{sort}(a)$ for all $a \in \mathbb{A}$. The operation of composing bijections gives a binary operation $\pi, \pi' \in \mathit{Perm} \mapsto \pi \circ \pi' \in \mathit{Perm}$

$$(\pi \circ \pi')(a) \triangleq \pi(\pi'(a)) \qquad (a \in \mathbb{A})$$

that makes $\mathit{Perm}$ into a group; we write $\iota$ for the identity atom-permutation and $\pi^{-1}$ for the inverse of $\pi$. Among the elements of $\mathit{Perm}$ 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 $\pi \in \mathit{Perm}$ is equal to a finite composition of such transpositions.

An **action** of $\mathit{Perm}$ on a set $X$ is a function $\mathit{Perm} \times X \to X$, whose effect on $(\pi, x) \in \mathit{Perm} \times X$ 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 $\pi, \pi' \in \mathit{Perm}$. 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 $A$.

**Definition 6.** A **nominal set** is by definition a set $X$ equipped with an action of $\mathit{Perm}$ such that every element $x \in X$ is supported by some *finite* set of atoms.

If $X$ is a nominal set and $A_1$ and $A_2$ are both finite sets of atoms supporting $x \in X$, then it is the case that $A_1 \cap A_2$ also supports $x$. To see this, suppose that $a$ and $a'$ are atoms of the same sort not in $A_1 \cap A_2$; we have to show $(a\ a') \cdot x = x$. This is certainly the case if $a = a'$ (because $(a\ a) = \iota$); and if $a \neq a'$, picking any $a''$ in the infinite set $\mathbb{A} - (A_1 \cup A_2 \cup \{a, a'\})$, then $(a\ a') = (a\ a'') \circ (a'\ a'') \circ (a\ a'')$ is a composition of transpositions each of which fixes $x$ (since for each of the three pairs of atoms, each element of the pair is either not in $A_1$, or not in $A_2$), so itself fixes $x$, as required. It follows immediately from this intersection property of finite supports that in a nominal set $X$, each element $x \in X$ possesses a *smallest* finite support, which we write as $\mathit{supp}_X(x)$, or just $\mathit{supp}(x)$ if $X$ is clear from the context, and call the **support of $x$** in $X$.

**Example 7.** (i) Each set $\mathbb{A}_\mathsf{a}$ of atoms of a particular sort $\mathsf{a}$ is a nominal set once we endow it with the atom-permutation action given by $\pi \cdot a = \pi(a)$; as one might expect, $\mathit{supp}(a) = \{a\}$. 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 $\mathsf{a}$ 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 $\Sigma$ be a nominal signature. Using Theorem 3 we can define an atom-permutation action on the sets $\mathsf{T}(\Sigma)_\sigma$ of terms over $\Sigma$ of each arity $\sigma \in Ar(\Sigma)$:

$$\begin{aligned}
\pi \cdot a &\triangleq \pi(a)\\
\pi \cdot K\,t &\triangleq K(\pi \cdot t)\\
\pi \cdot \langle\rangle &\triangleq \langle\rangle\\
\pi \cdot \langle t_1, t_2\rangle &\triangleq \langle \pi \cdot t_1, \pi \cdot t_2\rangle\\
\pi \cdot «a»t &\triangleq «\pi \cdot a»(\pi \cdot t)\ .
\end{aligned}$$

Using Theorem 4 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 $\alpha$-terms over $\Sigma$ (Section 2.4), first note that the action of atom-permutations on terms preserves $\alpha$-equivalence: this is a consequence of a general property (Theorem 12) of rule-based inductive definitions that we will establish at the end of Section 3.2. Therefore we get a well-defined action on $\alpha$-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 3) by:

$$\begin{aligned}
fa\,a &\triangleq \{a\}\\
fa(K\,t) &\triangleq fa\,t\\
fa\langle\rangle &\triangleq \emptyset\\
fa\langle t_1, t_2\rangle &\triangleq fa\,t_1 \cup fa\,t_n\\
fa\,«a»t &\triangleq fa\,t - \{a\}\ .
\end{aligned}$$

(iv) Each set $S$ becomes a nominal set, called the **discrete nominal set** on $S$, when 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 one-element set $1 = \{()\}$, the set of booleans $\mathbb{B} = \{true, false\}$ and the set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ as nominal sets in this way.

## 3.2 Products, functions and powersets

If $X_1$ and $X_2$ are nominal sets, then we get an action of atom-permutations on their cartesian product $X_1 \times X_2$ by defining $\pi \cdot (x_1, x_2)$ to be $(\pi \cdot x_1, \pi \cdot x_2)$, for each $(x_1, x_2) \in X_1 \times X_2$. If $A_i$ supports $x_i \in X_i$ for $i = 1, 2$, then it is not hard to see that $A_1 \cup A_2$ supports $(x_1, x_2) \in X_1 \times X_2$. Thus $X_1 \times X_2$ is also a nominal set. Note that

$$supp((x_1, x_2)) = supp(x_1) \cup supp(x_2) \tag{20}$$

since we have already observed that $supp(x_1) \cup supp(x_2)$ supports $(x_1, x_2)$, so that $supp((x_1, x_2)) \subseteq supp(x_1) \cup supp(x_2)$; and conversely, if $A$ supports $(x_1, x_2)$ then it also supports each $x_i$, so that $supp(x_i) \subseteq supp((x_1, x_2))$.

Turning next to functions, 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 more complicated than expected; however, note that it is equivalent to the requirement that function application be respected by atom-permutations:

$$\pi \cdot (f(x)) = (\pi \cdot f)(\pi \cdot x) \ . \tag{21}$$

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 whose objects are sets equipped with an atom-permutation action and whose morphisms are functions preserving the action. 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 9 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 the set

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

of **finitely supported functions** from $X$ to $Y$ is closed under the atom-permutation action and is a nominal set.

Given a nominal set $X$, we can use the usual bijection between subsets of $X$ and functions in $X \to \mathbb{B}$ (where $\mathbb{B} = \{true, false\}$) to transfer the action of atom-permutations on $X \to \mathbb{B}$ to one on subsets of $X$. From the definition of the action of atom-permutations on functions and using the fact that the action on $\mathbb{B}$ is trivial (see Example 7(iv)), one can calculate that this action sends $\pi \in Perm$ and $S \subseteq X$ to the subset

$$\pi \cdot S \ \triangleq \ \{\pi \cdot x \mid x \in S\} \ .$$

Note that if $S$ is supported by a set of atoms $A$ with respect to this action, then $\pi \cdot S$ is supported by $\{\pi(a) \mid a \in A\}$. So the set

$$P_{\mathrm{fs}}(X) \ \triangleq \ \{S \subseteq X \mid (\exists \text{ finite } A \subseteq \mathbb{A})\ A \text{ supports } S\}$$

of **finitely supported subsets** of the nominal set $X$ is closed under the atom-permutation action on all subsets of $X$ and hence is a nominal set.

**Example 8.** 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 7(i), the action of atom-permutations on $\pi$ *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 $\pi$ with respect to this action (and is in fact the smallest such set); so $Perm$ is a nominal set.

**Example 9.** 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**[5] for the set $\mathbb{A}$ of atoms, i.e. any function $choose \in (\mathbb{A} \to_{\mathrm{fs}} \mathbb{B}) \to \mathbb{A}$ (where $\mathbb{B} = \{true, false\}$) satisfying $f(a) = true \;\Rightarrow\; f(choose(f)) = true$, for all $f \in \mathbb{A} \to_{\mathrm{fs}} \mathbb{B}$ and $a \in \mathbb{A}$. To see this, we suppose that $choose$ is supported by some finite set $A \subseteq \mathbb{A}$ and derive a contradiction. Let $f \in \mathbb{A} \to \mathbb{B}$ be the function mapping $a \in \mathbb{A}$ to $true$ if $a \notin A$ and to $false$ if $a \in A$. It is not hard to see that $A$ supports $f$; in particular $f \in \mathbb{A} \to_{\mathrm{fs}} \mathbb{B}$ and we can apply $choose$ to obtain an atom $a_0 \triangleq choose(f)$. Let $\mathsf{a} = sort(a_0)$. Since $\mathbb{A}_{\mathsf{a}}$ is infinite and $A \cup \{a_0\}$ is finite, there is some atom $a_1 \in \mathbb{A}_{\mathsf{a}}$ with $a_1 \neq a_0$ and $a_1 \notin A$. Since $a_1 \notin A$, $f(a_1) = true$ by definition of $f$; and so since $choose$ is a choice function, we also have $f(choose(f)) = true$. By definition of $f$ and $a_0$ this means that $a_0 = choose(f) \notin A$. Since $a_0, a_1 \notin A$ and $A$ supports both $choose$ and $f$, we have $(a_0\; a_1) \cdot choose = choose$ and $(a_0\; a_1) \cdot f = f$. Thus by (21), $a_1 = (a_0\; a_1) \cdot a_0 = (a_0\; a_1) \cdot choose(f) = ((a_0\; a_1) \cdot choose)((a_0\; a_1) \cdot f) = choose(f) = a_0$, contradicting the fact that we picked $a_1$ to be different from $a_0$ and completing the proof.

We note a property of support with respect to function application that we use in what follows.

**Lemma 10.** *If $X$ and $Y$ are nominal sets, $f \in X \to_{\mathrm{fs}} Y$ and $x \in X$, then $supp(f(x)) \subseteq supp(f) \cup supp(x)$.*

*Proof.* For all atoms $a, a'$ of the same sort, if $a, a' \notin supp(f) \cup supp(x)$ then $(a\; a') \cdot f = f$ and $(a\; a') \cdot x = x$; so by (21) $(a\; a') \cdot f\, x = ((a\; a') \cdot f)((a\; a') \cdot x) = f\, x$. Thus $supp(f) \cup supp(x)$ supports $f\, x$ and hence $supp(f\, x)$ is contained in this finite set. $\square$

A corollary of this is that for nominal sets $X$ and $Y$, the operation of function application $app_{X,Y}(f, x) \triangleq f\, x$ is an element of $(X \to_{\mathrm{fs}} Y) \times X \to_{\mathrm{fs}} Y$; indeed $app_{X,Y}$ is supported by the empty set of atoms. Similarly, it is not hard to see

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

that currying, $cur_{X,Y,Z}(f) \triangleq \lambda x \in X.\lambda y \in Y.f(x,y)$, determines an element $cur_{X,Y,Z} \in ((X \times Y \to_{\text{fs}} Y) \to_{\text{fs}} (X \to_{\text{fs}} (Y \to_{\text{fs}} Z)))$ with empty support. The constantly true function $true_X(x) \triangleq true$ and the equality function $eq_X(x,x') \triangleq$ *if* $x = x'$ *then true else false* also determine elements $true_X \in (X \to_{\text{fs}} \mathbb{B})$ and $eq_X \in (X \times X \to_{\text{fs}} \mathbb{B})$ with empty support. It is for these reasons that the following general principle holds good.

**Theorem 11 (finite support principle).** *Any function or relation that is defined from finitely supported functions and relations using higher-order logic is itself finitely supported.* □

Because of this, the collection of finitely supported functions and subsets of nominal sets forms a very rich collection that is closed under the usual constructions of informal mathematics.[6] 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 [12]. In this paper I stick with ordinary higher-order logic: by considering all functions and subsets rather than just finitely supported ones, one sometimes gets more information about a construction. A good example of this is provided by a cornerstone of programming language semantics, namely *rule-based inductive definitions*. Given a nominal set $X$, let $R$ be a finitely supported set of rules for defining a subset of $X$; more precisely, let $R$ be an element of the nominal set $P_{\text{fs}}(P_{\text{fs}}(X) \times X)$. As usual, a subset $S \subseteq X$ is closed under the rules in $R$ if

$$(\forall (H,c) \in R)\ H \subseteq S \;\Rightarrow\; c \in S$$

and the smallest such subset, $ind(R)$, is given by the intersection of all such closed subsets. If we worked systematically in "FM-HOL" [12], rather than using arbitrary subsets of $X$, we would only consider *finitely supported* subsets that are closed under the rules, and would replace $ind(R)$ by $\bigcap\{S \in P_{\text{fs}}(X) \mid (\forall (H,c) \in R)\ H \subseteq S \;\Rightarrow\; c \in S\}$. However these two subsets coincide, as the following theorem shows.

**Theorem 12 (finitely supported inductive definitions).** *Let $X$ be a nominal set. For any set of rules $R \in P_{\text{fs}}(P_{\text{fs}}(X) \times X)$, the subset $ind(R) \subseteq X$ inductively defined by $R$ is a finitely supported subset of $X$; indeed $supp(ind(R)) \subseteq supp(R)$.*

*Proof.* Suppose $a, a'$ are atoms of the same sort that are not in the support of $R$. We have to show that $(a\ a') \cdot ind(R) = ind(R)$. It suffices to just show $ind(R) \subseteq$

[6]The only exception being that the finite support property is not conserved by all uses of choice: see Example 9.

$(a\ a')\cdot \mathit{ind}(R)$. (For applying $(a\ a')$ to both sides then gives the reverse inclusion.) For this it suffices to show that $(a\ a')\cdot \mathit{ind}(R)$ is closed under the rules in $R$, since $\mathit{ind}(R)$ is the smallest such subset. But if $(H, c) \in R$, then $((a\ a')\cdot H, (a\ a')\cdot c)$ is also a rule in $R$, because $(a\ a')\cdot(H, c) \in (a\ a')\cdot R = R$ since $a, a' \notin \mathit{supp}(R)$. So if $H \subseteq (a\ a')\cdot \mathit{ind}(R)$, then $(a\ a')\cdot H \subseteq (a\ a')\cdot(a\ a')\cdot \mathit{ind}(R) = \mathit{ind}(R)$, so $(a\ a')\cdot c \in \mathit{ind}(R)$ since $\mathit{ind}(R)$ is closed under the rule $((a\ a')\cdot H, (a\ a')\cdot c)$; and hence $c = (a\ a')\cdot(a\ a')\cdot c \in (a\ a')\cdot \mathit{ind}(R)$. □

In this paper we will confine ourselves to finitely supported *finitary* rules, i.e. those $R$ only containing (hypothesis, conclusion)-pairs $(H, c)$ for which $H$ is a finite subset of $X$. Every finite subset of a nominal set is in particular a finitely supported subset: clearly $\{x_1, \ldots, x_n\}$ is supported by $\mathit{supp}(x_1)\cup\cdots\cup \mathit{supp}(x_n)$. Furthermore the action of atom-permutations on subsets of $X$ clearly sends finite subsets to finite subsets. So the finite powerset $P_{\mathrm{fin}}(X)$ is a nominal set when $X$ is. For finitary rules we just have to check that $R$ is a finitely supported subset of the nominal set $P_{\mathrm{fin}}(X) \times X$ to conclude from the above theorem that the subset $\mathit{ind}(R)$ it inductively defines is again finitely supported. For example, it is not hard to see that the rule set given schematically in Figure 1 is a subset of $P_{\mathrm{fin}}(\mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma)) \times (\mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma))$ that is supported by the empty set of atoms. Therefore by the theorem, the relation $=_\alpha$ of $\alpha$-equivalence is supported by the empty set. So $=_\alpha$ is preserved by all atom-transpositions and hence also by any $\pi \in \mathit{Perm}$ (since each $\pi$ is a composition of transpositions):

$$t =_\alpha t' : \sigma \;\Rightarrow\; \pi\cdot t =_\alpha \pi\cdot t' : \sigma\ . \tag{22}$$

We used this property in Example 7(iii) when discussing the nominal set structure of $\mathsf{T}_\alpha(\Sigma)$.

## 3.3 Nominal subsets and quotients

If $X$ is a nominal set and $S \in P_{\mathrm{fs}}(X)$ is a finitely supported subset of it, then $S$ is not necessarily itself a nominal set, because for any $x \in S$ and $\pi \in \mathit{Perm}$ we have no guarantee that $\pi\cdot x$ again lies in $S$. But if $\mathit{supp}(S) = \emptyset$, then $(a\ a')\cdot S = S$ for all atoms $a, a'$ of the same sort; and since each $\pi \in \mathit{Perm}$ is the composition of transpositions, in this case it follows that $\pi\cdot S = S$. From this it is not hard to see that the condition $\mathit{supp}(S) = \emptyset$ is equivalent to

$$(\forall \pi \in \mathit{Perm}, x \in X)\; x \in S \;\Rightarrow\; \pi\cdot x \in S\ . \tag{23}$$

So when (23) holds the action of atom-permutations on elements of $X$ restricts to an action on $S$; and furthermore, the support of each $x \in S$ is the same as its support as an element of $X$. Therefore $S$ is a nominal set. We call such an $S$ a **nominal subset** of $X$. (The term **equivariant subset** is also commonly used for this.)

Another useful way of forming nominal sets is by taking quotients. If $\sim$ is an equivalence relation on a nominal set $X$, then so long as $\sim$ is a nominal subset of

$X \times X$, the usual set $X/{\sim}$ of equivalence classes inherits a well-defined atom-permutation action, given by $\pi \cdot [x] = [\pi \cdot x]$. Furthermore, an equivalence class is supported by any set of atoms that supports a representative of the class. So $X/{\sim}$ is a nominal set. The construction of the nominal set $\mathsf{T}_\alpha(\Sigma)_\sigma$ of $\alpha$-terms (of arity $\sigma$ over a nominal signature $\Sigma$) from the nominal set $\mathsf{T}(\Sigma)_\sigma$ is an example of this construction, since we saw in (22) that $\alpha$-equivalence is a nominal subset of $\mathsf{T}(\Sigma)_\sigma \times \mathsf{T}(\Sigma)_\sigma$.

## 3.4 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 $x \mathrel{\#} y$ when $\mathit{supp}_X(x) \cap \mathit{supp}_Y(y) = \emptyset$ and say that $x$ is **fresh** for $y$.

**Lemma 13.** *Let $X$ and $Y$ be nominal sets. For any $x \in X$, $y \in Y$, $\pi \in \mathit{Perm}$, and atoms $a, a' \in \mathbb{A}$ of the same sort,*

$$\pi \cdot ((a\ a') \cdot x) = (\pi(a)\ \pi(a')) \cdot (\pi \cdot x) \tag{24}$$

$$x \mathrel{\#} y \Rightarrow \pi \cdot x \mathrel{\#} \pi \cdot y\ . \tag{25}$$

*Proof.* Equation (24) follows immediately from the fact that the atom-permutations $\pi \circ (a\ a')$ and $(\pi(a)\ \pi(a')) \circ \pi$ are always equal.

Given a set of atoms $A$, recall from the previous section that we write $\pi \cdot A$ for the set $\{\pi(a) \mid a \in A\}$. Note that since each permutation is in particular a bijection, it is the case that $\pi \cdot (A \cap A') = \pi \cdot A \cap \pi \cdot A'$. Therefore to prove (25) it suffices to show that $\pi \cdot \mathit{supp}(x) = \mathit{supp}(\pi \cdot x)$. But if $A$ supports $x$, then for any atoms $a, a' \notin \pi \cdot A$ (of the same sort), we have $\pi^{-1}(a), \pi^{-1}(a') \notin A$ and hence $(\pi^{-1}(a)\ \pi^{-1}(a')) \cdot x = x$. Applying $\pi \cdot (-)$ to both sides of this equation and using (24), we get $(a\ a') \cdot (\pi \cdot x) = \pi \cdot x$. Thus $\pi \cdot A$ supports $\pi \cdot x$ when $A$ supports $x$. So

$$(\forall \pi \in \mathit{Perm})(\forall x \in X)\ \mathit{supp}(\pi \cdot x) \subseteq \pi \cdot \mathit{supp}(x)\ .$$

Hence $\mathit{supp}(x) = \mathit{supp}(\pi^{-1} \cdot \pi \cdot x) \subseteq \pi^{-1} \cdot \mathit{supp}(\pi \cdot x)$ and thus we also have $\pi \cdot \mathit{supp}(x) \subseteq \mathit{supp}(\pi \cdot x)$. So we do indeed have $\pi \cdot \mathit{supp}(x) = \mathit{supp}(\pi \cdot x)$ and hence also (25). □

Recall that the set of atoms $\mathbb{A}$ is a nominal set as in Example 7(i). The following simple property of finitely supported sets of atoms is extremely useful when dealing with properties of fresh atoms; it subsumes [13, Proposition 4.10] and [20, Proposition 4].

**Theorem 14 (some/any theorem).** *Let $S \in P_{\mathrm{fs}}(\mathbb{A})$ be a set of atoms supported by some finite set of atoms $A$. For each atom-sort $\mathsf{a} \in \mathbb{AS}$, the following are equivalent:*

$$(\forall a \in \mathbb{A}_{\mathsf{a}})\ a \notin A\ \Rightarrow\ a \in S \tag{26}$$

$$(\exists a \in \mathbb{A}_{\mathsf{a}})\ a \notin A\ \&\ a \in S\ . \tag{27}$$

*Proof.* Since $\mathbb{A}_{\mathsf{a}} - A$ is infinite, it is in particular non-empty; thus (26) implies (27). Conversely, suppose $a \in \mathbb{A}_{\mathsf{a}} - A$ satisfies $a \mathrel{\#} S$. Given any other $a' \in \mathbb{A}_{\mathsf{a}} - A$, we have to show $a' \mathrel{\#} S$; but $(a\ a') \cdot S = S$ (since $a, a' \notin A \supseteq \mathit{supp}(S)$) and hence $a' = (a\ a') \cdot a \in (a\ a') \cdot S = S$. □

**Example 15.** Recall from Example 7(ii) that in the nominal set $\mathsf{T}(\Sigma)_\sigma$ of terms of arity $\sigma$ over a nominal signature $\Sigma$, the support of a term $t$ is just the finite set $\mathit{atm}\, t$ of atoms that occur in $t$. Furthermore, if $a' \notin \mathit{atm}\, t$, then the terms $t\{a'/a\}$ (replace $a$ by $a'$ throughout $t$) and $(a\ a) \cdot t$ (swap $a$ and $a'$ throughout $t$) are the same. Therefore the crucial rule ($=_\alpha$-5) in the definition of $\alpha$-equivalence of nominal terms can be rewritten as:

$$\frac{\mathsf{a} \in \Sigma_{\mathrm{A}} \qquad a, a', a'' \in \mathbb{A}_{\mathsf{a}} \qquad a'' \mathrel{\#} (a, t, a', t') \qquad (a''\ a) \cdot t =_\alpha (a''\ a') \cdot t' : \sigma}{«a»t =_\alpha «a'»t' : «\mathsf{a}»\sigma}$$

In particular we have

$$«a»t =_\alpha «a'»t' : «\mathsf{a}»\sigma \;\Leftrightarrow\; (\exists a'' \in \mathbb{A}_{\mathsf{a}})\; a'' \mathrel{\#} (a, t, a', t') \;\;\&\;\; (a\ a'') \cdot t =_\alpha (a'\ a'') \cdot t' : \sigma\ . \tag{28}$$

Applying Theorem 14 to the set of atoms $S = \{a'' \in \mathbb{A}_{\mathsf{a}} \mid (a\ a'') \cdot t =_\alpha (a'\ a'') \cdot t' : \sigma\}$, which (by Lemma 13) is supported by $A = \mathit{supp}(a, t, a', t')$, we get from (28) a useful property of $\alpha$-equivalence of atom-binding terms:

$$«a»t =_\alpha «a'»t' : «\mathsf{a}»\sigma \;\Leftrightarrow\; (\forall a'' \in \mathbb{A}_{\mathsf{a}})\; a'' \mathrel{\#} (a, t, a', t') \;\Rightarrow\; (a\ a'') \cdot t =_\alpha (a'\ a'') \cdot t' : \sigma\ . \tag{29}$$

The next result provides a very general criterion for when a construction that "picks a fresh atom" is independent of which fresh atom is chosen.

**Theorem 16 (freshness theorem).** *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*

$$(\exists a \in \mathbb{A}_{\mathsf{a}})\; a \mathrel{\#} h \;\;\&\;\; a \mathrel{\#} h(a) \tag{30}$$

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

$$(\forall a \in \mathbb{A}_{\mathsf{a}})\; a \mathrel{\#} h \;\Rightarrow\; h(a) = \mathit{fresh}(h)\ . \tag{31}$$

*Furthermore,* $\mathit{supp}(\mathit{fresh}(h)) \subseteq \mathit{supp}(h)$.

*Proof.* Given (30) we have to prove that $h$ is constant on the non-empty set $\mathbb{A}_{\mathsf{a}} - \mathit{supp}(h)$. First note that by Theorem 14 (with $S \triangleq \{a \in \mathbb{A}_{\mathsf{a}} \mid a \mathrel{\#} h(a)\}$ and $A \triangleq \mathit{supp}(h) \supseteq \mathit{supp}(S)$), if (30) holds then

$$(\forall a \in \mathbb{A}_{\mathsf{a}})\; a \mathrel{\#} h \;\Rightarrow\; a \mathrel{\#} h(a)\ . \tag{32}$$

Suppose $a, a' \in \mathbb{A}_{\mathsf{a}} - \mathit{supp}(h)$. To see that $h(a) = h(a')$, without loss of generality we may assume $a \neq a'$. By Lemma 10, $a \mathrel{\#} h(a')$ (since $a \mathrel{\#} h$ and $a \mathrel{\#} a'$); and $a' \mathrel{\#} h(a')$ holds by (32). Hence

$$\begin{aligned} h(a') &= (a\ a') \cdot h(a') && \text{since } (a, a') \mathrel{\#} h(a') \\ &= ((a\ a') \cdot h)((a\ a') \cdot a') && \text{by (21)} \\ &= h((a\ a') \cdot a') && \text{since } (a, a') \mathrel{\#} h \\ &= h(a)\ . \end{aligned}$$

So there is a unique element $\mathit{fresh}(h) \in X$ satisfying (31). To see that it is supported by $\mathit{supp}(h)$, if $a, a'$ are atoms (of the same sort) satisfying $(a, a') \mathrel{\#} h$, choosing any $a''$ in the infinite set $\mathbb{A}_{\mathsf{a}} - \mathit{supp}(h, a, a')$, we have $(a\ a') \cdot \mathit{fresh}(h) = (a\ a') \cdot h(a'') = ((a\ a') \cdot h)((a\ a') \cdot a'') = h(a'') = \mathit{fresh}(h)$. Thus $\mathit{supp}(h)$ supports $\mathit{fresh}(h)$. □

## 4 Recursion and Induction Principles for $\alpha$-Terms

Recall from Definition 5 that $\mathsf{T}_\alpha(\Sigma)_\sigma$ denotes the set of $\alpha$-terms of arity $\sigma$ over a nominal signature $\Sigma$; by definition these are $\alpha$-equivalence classes $[t]_\alpha$ of terms $t : \sigma$. Elementary properties of the relation $=_\alpha$ of $\alpha$-equivalence yield the following structural properties of $\alpha$-terms; at the same time we introduce some concrete syntax for $\alpha$-terms mirroring the informal notation for $\alpha$-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 $\alpha$-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 $K\ e'$.*

**Unit:** $\mathsf{T}_\alpha(\Sigma)_1$ contains a unique equivalence class, $[\langle\rangle]_\alpha$, *which we write as* $()$.

**Pairs:** if $\sigma_1, \sigma_2 \in Ar(\Sigma)$ and $e \in \mathsf{T}_\alpha(\Sigma)_{\sigma_1 * \sigma_2}$, then there are unique $e_i \in \mathsf{T}_\alpha(\Sigma)_{\sigma_i}$ for $i = 1, 2$ such that there exist $t_i$ with $e_i = [t_i]_\alpha$ $(i = 1, 2)$ and $e = [\langle t_1, t_2\rangle]_\alpha$. *In this case we write $e$ as $(e_1, e_2)$.*

**Atom-binding:** if $\mathsf{a} \in \Sigma_{\mathrm{A}}$, $\sigma \in Ar(\Sigma)$ and $e \in \mathsf{T}_\alpha(\Sigma)_{«\mathsf{a}»\sigma}$, then for each $a \in \mathbb{A}_{\mathsf{a}}$ with $a \mathrel{\#} e$ (i.e. with $a$ not a free atom of $e$—cf. Example 7(iii)), there is a unique $e' \in \mathsf{T}_\alpha(\Sigma)_\sigma$ such that there exists $t'$ with $e' = [t']_\alpha$ and $e = [«a»t']_\alpha$. *In this case we write $e$ as $a.\, e'$ .*

Using this notation we now give a first version of structural recursion for $\alpha$-terms over a nominal signature. Compared with Theorem 3, 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 atom-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 $\alpha$-equivalence classes and satisfies the required recursion equations—for *all* sufficiently fresh bound atoms, in the case of the recursion equation for binders.

**Theorem 17 (first $\alpha$-structural recursion theorem).** *Let $\Sigma$ be a nominal signature. Suppose we are given an arity-indexed family of nominal sets $(X_\sigma \mid \sigma \in Ar(\Sigma))$ and elements*

$$\begin{aligned}
f_{\mathsf{a}} &\in \mathbb{A}_{\mathsf{a}} \to_{\mathrm{fs}} X_{\mathsf{a}} && (\mathsf{a} \in \Sigma_{\mathrm{A}})\\
f_K &\in X_\sigma \to_{\mathrm{fs}} X_{\mathsf{s}} && ((K : \sigma \to_{\mathrm{fs}} \mathsf{s}) \in \Sigma_{\mathrm{C}})\\
f_{\mathtt{1}} &\in X_{\mathtt{1}} &&\\
f_{\sigma_1 * \sigma_2} &\in X_{\sigma_1} \times X_{\sigma_2} \to_{\mathrm{fs}} X_{\sigma_1 * \sigma_2} && (\sigma_1, \sigma_2 \in Ar(\Sigma))\\
f_{«\mathsf{a}»\sigma} &\in \mathbb{A}_{\mathsf{a}} \times X_\sigma \to_{\mathrm{fs}} X_{«\mathsf{a}»\sigma} && (\mathsf{a} \in \Sigma_{\mathrm{A}}, \sigma \in Ar(\Sigma))
\end{aligned}$$

*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 $«\mathsf{a}»\sigma \in Ar(\Sigma)$, the function $f_{«\mathsf{a}»\sigma}$ satisfies*

$$(\exists a' \in \mathbb{A}_{\mathsf{a}})\ a' \notin A \ \ \&\ \ (\forall x \in X_\sigma)\ a' \mathrel{\#} f_{«\mathsf{a}»\sigma}(a', x)\ . \tag{FCB}$$

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

$$\hat{f} a = f_{\mathsf{a}}(a) \tag{33}$$

$$\hat{f}(K\, e) = f_K(\hat{f} e) \tag{34}$$

$$\hat{f}() = f_{\mathtt{1}} \tag{35}$$

$$\hat{f}(e_1, e_2) = f_{\sigma_1 * \sigma_2}(\hat{f} e_1, \hat{f} e_2) \tag{36}$$

$$a \notin A \ \Rightarrow\ \hat{f}(a.\, e) = f_{«\mathsf{a}»\sigma}(a, \hat{f} e) \tag{37}$$

*where we have abbreviated $\hat{f}_\sigma(e)$ to $\hat{f} e$ and used the notation for $\alpha$-terms introduced above.*

*Proof.* We can reduce the proof of the theorem to an application of Theorem 3, taking advantage of the fact that we are working (informally) in higher-order logic.[7] From the $Ar(\Sigma)$-indexed family of nominal sets $X_\sigma$ we define another such family: $S_\sigma \triangleq Perm \to_{\mathrm{fs}} X_\sigma$ (regarding $Perm$ as a nominal set as in Example 8 and using

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

the $\to_{\mathrm{fs}}$ construct from Section 3.2). Now define elements $g_{\mathsf{a}}$, $g_K$, $g_1$, $g_{\sigma_1 * \sigma_2}$ and $g_{\text{«a»}\sigma}$ as in the statement of Theorem 3, as follows.

$$\begin{aligned}
g_{\mathsf{a}}\, a &\triangleq \lambda\pi \in \mathit{Perm}.\ f_{\mathsf{a}}(\pi(a))\\
g_K\, s &\triangleq \lambda\pi \in \mathit{Perm}.\ f_K(s(\pi))\\
g_1 &\triangleq \lambda\pi \in \mathit{Perm}.\ f_1\\
g_{\sigma_1 * \sigma_2}(s_1, s_2) &\triangleq \lambda\pi \in \mathit{Perm}.\ f_{\sigma_1 * \sigma_2}(s_1(\pi), s_2(\pi))\\
g_{\text{«a»}\sigma}(a, s) &\triangleq \lambda\pi \in \mathit{Perm}.\ \mathit{fresh}(\lambda a' \in \mathbb{A}_{\mathsf{a}}.\ f_{\text{«a»}\sigma}(a', s(\pi \circ (a\ a'))))
\end{aligned}$$

The crucial clause in this definition is the last one, where we are using the *fresh* functional from Theorem 16 applied to the function $h \triangleq \lambda a' \in \mathbb{A}_{\mathsf{a}}.\ f_{\text{«a»}\sigma}(a', s(\pi \circ (a\ a')))$. For this to make sense it has to be the case that $h$ is finitely supported and satisfies condition (30) of that lemma; let us see why this is so. Since $\mathit{supp}(f_{\text{«a»}\sigma}) \subseteq A$ by assumption, it follows that $h$ is supported by the finite set $A \cup \mathit{supp}(s, \pi, a)$. To see that (30) holds of $h$, let $a'$ be the atom whose existence is asserted by (FCB); thus $a' \notin A$ and $a' \mathrel{\#} f_{\text{«a»}\sigma}(a', x)$ for any $x \in X_\sigma$. For any other $a'' \in \mathbb{A}_{\mathsf{a}} - A$, we have $(a'\ a'') \cdot f_{\text{«a»}\sigma} = f_{\text{«a»}\sigma}$ (since $a', a'' \notin \mathit{supp}(f_{\text{«a»}\sigma})$); hence applying $(a'\ a'')$ to $a' \mathrel{\#} f_{\text{«a»}\sigma}(a', x)$, from Lemma 13 we get $a'' \mathrel{\#} f_{\text{«a»}\sigma}(a'', (a'\ a'') \cdot x)$ for any $x \in X_\sigma$. Choosing $a''$ to be in the infinite set $\mathbb{A}_{\mathsf{a}} - (A \cup \mathit{supp}(s, \pi, a))$ and $x = (a'\ a'') \cdot s(\pi \circ (a\ a''))$, we conclude that $a'' \mathrel{\#} h$ and $a'' \mathrel{\#} f_{\text{«a»}\sigma}(a'', s(\pi \circ (a\ a''))) = h(a'')$, as required for (30).

Applying Theorem 3 with this data, we get a family of functions

$$\hat{g}_\sigma \in \mathsf{T}(\Sigma)_\sigma \to (\mathit{Perm} \to_{\mathrm{fs}} X_\sigma)$$

satisfying the recursion equations (10)–(14) of that theorem. Next one proves that these functions respect $\alpha$-equivalence:

$$t_1 =_\alpha t_2 : \sigma \ \Rightarrow\ \hat{g}_\sigma\, t_1 = \hat{g}_\sigma\, t_2. \tag{38}$$

This is done by induction over the derivation of $t_1 =_\alpha t_2 : \sigma$ from the rules in Figure 1; the induction step for rule ($=_\alpha$-5) uses the following property of $\hat{g}$, which follows by induction on the structure of $t$, i.e. using Theorem 4:

$$(\forall \sigma \in \mathit{Ar}(\Sigma), t : \sigma)(\forall \pi, \pi' \in \mathit{Perm})\ \hat{g}_\sigma\, t\, (\pi \circ \pi') = \hat{g}_\sigma(\pi' \cdot t)\, \pi. \tag{39}$$

In view of (38), the functions $\hat{g}_\sigma$ induce functions $\hat{f}_\sigma \in \mathsf{T}_\alpha(\Sigma)_\sigma \to X_\sigma$ given by $\hat{f}_\sigma[t]_\alpha \triangleq \hat{g}_\sigma\, t\, \iota$ for any $t : \sigma$ (recalling that $\iota$ stands for the identity permutation). One proves that these functions $\hat{f}_\sigma$ are all supported by $A$ by first proving that the functions $\hat{g}_\sigma$ are so supported; the latter follows from the uniqueness part of Theorem 3: if $a, a'$ are atoms of the same sort not in $A$, then one can show that $(a\ a') \cdot \hat{g}_\sigma$ satisfies the same recursion equations as $\hat{g}_\sigma$ and hence is equal to that function. The fact that the $\hat{f}_\sigma$ satisfy the required recursion equations (33)–(37) follows from the recursion equations (10)–(14) satisfied by the $\hat{g}_\sigma$. That concludes the existence part of the proof of Theorem 17.

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 (33)–(37) for $\hat{f}_\sigma$. Define $g'_\sigma \in \mathsf{T}(\Sigma)_\sigma \to S_\sigma$ by $g'_\sigma\, t\, \pi \triangleq f'_\sigma[\pi \cdot t]_\alpha$ ($\sigma \in Ar(\Sigma), t : \sigma, \pi \in \mathit{Perm}$). One can show that the $g'_\sigma$ satisfy the same recursion equations (10)–(14) from Theorem 3 as the functions $\hat{g}_\sigma$; so by the uniqueness part of that theorem, $g'_\sigma = \hat{g}_\sigma$. Therefore for all $t : \sigma$, $f'_\sigma[t]_\alpha = f'_\sigma[\iota \cdot t]_\alpha \triangleq g'_\sigma\, t\, \iota = \hat{g}_\sigma\, t\, \iota \triangleq \hat{f}_\sigma[t]_\alpha$; hence $f'_\sigma = \hat{f}_\sigma$. □

**Example 18 (length of an $\alpha$-term).** In [14, Section 3.3] Gordon and Melham give the usual recursion scheme for defining the length of a $\lambda$-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 [18, Section 3] on the way to his improved version of Gordon and Melham's recursion scheme (discussed further in Example 27 and Section 7). Pleasingly, the usual recursive definition of the length of a $\lambda$-term, or more generally of an $\alpha$-term over any nominal signature, is a very simple application of the First $\alpha$-Structural Recursion Theorem.[8] Thus in Theorem 17 we take $X_\sigma$ to be the discrete nominal set $\mathbb{N}$ of natural numbers and

$$\begin{aligned}
f_{\mathsf{a}} &\triangleq \lambda a \in \mathbb{A}_{\mathsf{a}}.\, 1\\
f_K &\triangleq \lambda k \in \mathbb{N}.\, k+1\\
f_1 &\triangleq 0\\
f_{\sigma_1 * \sigma_2} &\triangleq \lambda(k_1,k_2) \in \mathbb{N} \times \mathbb{N}.\, k_1 + k_2\\
f_{\text{«a»}\sigma} &\triangleq \lambda(a,k) \in \mathbb{A}_{\mathsf{a}} \times \mathbb{N}.\, k+1\ .
\end{aligned}$$

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

$$\begin{aligned}
\mathit{length}\, a &= 1\\
\mathit{length}(K\, e) &= \mathit{length}\, e + 1\\
\mathit{length}() &= 0\\
\mathit{length}(e_1, e_2) &= \mathit{length}\, e_1 + \mathit{length}\, e_2\\
\mathit{length}(a.\, e) &= \mathit{length}\, e + 1\ .
\end{aligned}$$

Note that the last clause holds for all $a$, because in (37) the condition "$a \notin A$" is vacuously true (since $A = \emptyset$).

**Remark 19.** In Theorem 17 we gave (FCB) as an existential statement. It is in fact equivalent to the universal statement

$$(\forall a' \in \mathbb{A}_{\mathsf{a}})\ a \notin A \ \Rightarrow\ (\forall x \in X_\sigma)\ a' \mathrel{\#} f_{\text{«a»}\sigma}(a', x)\ .$$

[8]The same goes for Norrish's `stripc` function, used to illustrate the limitations of Gordon and Melham's workaround for the *length* function [18, p. 247].

This follows from the "some/any" Theorem 14 by taking $S$ to be $\{a' \in \mathbb{A}_{\mathsf{a}} \mid (\forall x \in X_\sigma)\ a' \mathrel{\#} f_{\text{«a»}\sigma}(a', x)\}$ and checking that $A$ supports $S$.

**Remark 20 (primitive recursion).** Theorem 17 gives a simple "iterative" form of structural recursion for $\alpha$-terms, rather than a more complicated "primitive recursive" form with recursion equations

$$\begin{aligned}
\hat{f} a &= f_{\mathsf{a}}(a)\\
\hat{f}(K\,e) &= f_K(e, \hat{f} e)\\
\hat{f}() &= f_{\mathsf{1}}\\
\hat{f}(e_1, e_2) &= f_{\sigma_1 * \sigma_2}(e_1, e_2, \hat{f} e_1, \hat{f} e_2)\\
a \notin A \Rightarrow \hat{f}(a.\,e) &= f_{\text{«a»}\sigma}(a, e, \hat{f} e)\ .
\end{aligned}$$

In fact this more general form can be deduced from the simple one given in the theorem by adapting to our nominal setting a similar result for ordinary structural recursion: defining $X'_\sigma \triangleq \mathsf{T}_\alpha(\Sigma)_\sigma \times X_\sigma$ and functions

$$\begin{aligned}
f'_{\mathsf{a}}(a) &\triangleq (a, f_{\mathsf{a}}(a))\\
f'_K(e, x) &\triangleq (K\,e, f_K(e, x))\\
f'_{\mathsf{1}} &\triangleq ((), f_{\mathsf{1}})\\
f'_{\sigma_1 * \sigma_2}((e_1, x_1), (e_2, x_2)) &\triangleq ((e_1, e_2), f_{\sigma_1 * \sigma_2}(e_1, e_2, x_1, , x_2))\\
f'_{\text{«a»}\sigma}(a, e, x) &\triangleq (a.\,e, f_{\text{«a»}\sigma}(a, e, x))
\end{aligned}$$

we first apply Theorem 17 to get functions $\hat{f}'_\sigma \in \mathsf{T}_\alpha(\Sigma)_\sigma \to_{\mathrm{fs}} \mathsf{T}_\alpha(\Sigma)_\sigma \times X_\sigma$. The uniqueness part of the theorem allows us to deduce that the first components of these functions are all identity functions; it follows from this that the second component of $\hat{f}'_\sigma$ is a function $\hat{f}_\sigma \in \mathsf{T}_\alpha(\Sigma)_\sigma \to_{\mathrm{fs}} X_\sigma$ satisfying the above scheme of primitive recursion (and is the unique such).

The next theorem gives a version of structural induction for $\alpha$-terms. Just as Theorem 17 was derived from ordinary structural recursion (Theorem 3), we prove this theorem as a corollary of ordinary structural induction (Theorem 4).

**Theorem 21 (first $\alpha$-structural induction theorem).** *Let $\Sigma$ be a nominal signature. Suppose we are given a finitely supported set $S \in P_{\mathrm{fs}}(\mathsf{T}_\alpha(\Sigma))$ of $\alpha$-terms over $\Sigma$. To prove that $S$ is the whole of $\mathsf{T}_\alpha(\Sigma)$, it suffices to show*

$$(\forall \mathsf{a} \in \Sigma_{\mathrm{A}}, a \in \mathbb{A}_{\mathsf{a}})\ a \in S \tag{40}$$

$$(\forall (K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}}, e \in \mathsf{T}_\alpha(\Sigma)_\sigma)\ e \in S \ \Rightarrow\ K\,e \in S \tag{41}$$

$$() \in S \tag{42}$$

$$\begin{aligned}&(\forall (\sigma_i \in Ar(\Sigma), e_i \in \mathsf{T}_\alpha(\Sigma)_{\sigma_i} \mid i = 1, 2))\\ &\qquad e_1 \in S \mathbin{\&} e_2 \in S \ \Rightarrow\ (e_1, e_2) \in S\end{aligned} \tag{43}$$

$$\begin{aligned}&(\forall \mathsf{a} \in \Sigma_{\mathrm{A}}, \sigma \in Ar(\Sigma))(\exists a \in \mathbb{A}_{\mathsf{a}})\ a \mathrel{\#} S\ \&\\ &\qquad (\forall e \in \mathsf{T}_\alpha(\Sigma)_\sigma)\ e \in S \ \Rightarrow\ a.\,e \in S\ .\end{aligned} \tag{44}$$

*Proof.* Let $\overline{S}$ be the set of nominal terms over $\Sigma$ whose $\alpha$-equivalence classes lie in $S$ no matter how we permute the atoms occurring in the term:

$$\overline{S} \triangleq \{t \in \mathsf{T}(\Sigma) \mid (\forall \pi \in \mathit{Perm})\ [\pi \cdot t]_\alpha \in S\}\ .$$

Clearly $S = \mathsf{T}_\alpha(\Sigma)$ if $\overline{S} = \mathsf{T}(\Sigma)$; and to prove the latter it suffices to check that $\overline{S}$ satisfies conditions (15)–(19) of Theorem 4. The first four of these follow immediately from (40)–(43) respectively. So it just remains to show that (44) implies that $\overline{S}$ satisfies condition (19). First note that by Theorem 14 applied to the set of atoms $\{a \in \mathbb{A}_{\mathsf{a}} \mid (\forall e \in \mathsf{T}_\alpha(\Sigma)_\sigma)\ e \in S \ \Rightarrow\ a.\, e \in S\}$, which is supported by $\mathit{supp}(S)$, (44) is equivalent to

$$\begin{aligned}(\forall \mathsf{a} \in \Sigma_{\mathrm{A}}, \sigma \in \mathit{Ar}(\Sigma))(\forall a \in \mathbb{A}_{\mathsf{a}})\ a \mathrel{\#} S \ \Rightarrow& \\ &(\forall e \in \mathsf{T}_\alpha(\Sigma)_\sigma)\ e \in S \ \Rightarrow\ a.\, e \in S\ . \quad (45)\end{aligned}$$

Given $a \in \mathbb{A}_{\mathsf{a}}$ and $t \in \overline{S}$, we have to prove that $«a»t \in \overline{S}$, i.e. that $[\pi \cdot «a»t]_\alpha \in S$ for any $\pi \in \mathit{Perm}$. Choosing any atom $a'$ in the infinite set $\mathbb{A}_{\mathsf{a}} - \mathit{supp}(S, \pi, a, t)$, we have

$$\begin{aligned}\pi \cdot «a»t &= «\pi(a)»(\pi \cdot t) \\ &=_\alpha «a'»((\pi \cdot t)\{a'/\pi(a)\}) && \text{by definition of } =_\alpha \text{ (Section 2.4)} \\ &= «a'»((\pi(a)\ a') \cdot (\pi \cdot t)) && \text{since } a' \notin \mathit{atm}(\pi \cdot t) \\ &= «a'»(\pi' \cdot t) && \text{where } \pi' \triangleq (\pi(a)\ a') \circ \pi.\end{aligned}$$

So $[\pi \cdot «a»t]_\alpha = a'.\, [\pi' \cdot t]_\alpha \in S$ by (45), since $a' \mathrel{\#} S$ (by choice of $a'$) and $[\pi' \cdot t]_\alpha \in S$, because $t \in \overline{S}$. So it is indeed the case that $«a»t \in \overline{S}$ when $a \in \mathbb{A}_{\mathsf{a}}$ and $t \in \overline{S}$. □

## 5 Second $\alpha$-Structural Recursion & Induction Theorems

Theorem 17 is an "arity-directed" recursion principle for $\alpha$-terms: one has to specify nominal sets $X_\sigma$ for each arity $\sigma$, and give functions $f_{(\_)}$ for atom-sorts, unit, pair and atom-binding arities in addition to ones for constructors. Although this gives flexibility over how to treat atom, unit, pair and atom-binding $\alpha$-terms when giving an $\alpha$-structurally recursive definition of some functions, this flexibility is often more of a hindrance than a help. In most cases one is primarily interested in defining functions only on $\alpha$-terms whose arities are data-sorts $\mathsf{s} \in \Sigma_{\mathrm{D}}$, with $\alpha$-terms of other kinds of arity (atom-sorts, unit, pair and atom-binding arities) playing an auxiliary role. For example, when $\Sigma$ is the nominal signature for $\lambda$-terms with local recursive function declarations (Example 1), to define the capture-avoiding substitution function $\hat{s}_{x,e} \in \mathsf{T}_\alpha(\Sigma)_{\mathsf{t}} \to_{\mathrm{fs}} \mathsf{T}_\alpha(\Sigma)_{\mathsf{t}}$ discussed in the Introduction, we should only have to specify finitely supported functions corresponding to the right-hand sides of the defining equations (6)–(9), i.e. one function for each of the signature's four constructors $V$, $A$, $L$ and $\mathit{Letrec}$. But as it stands, to define

$\hat{s}_{x,e}$ using Theorem 17 we have to work out suitable choices for $X_\sigma$ and for the functions $f_{\mathsf{v}}, f_1, f_{\sigma_1 * \sigma_2}, f_{«\mathsf{v}»\sigma}$ for any $\sigma, \sigma_1, \sigma_2 \in Ar(\Sigma)$.

So we will develop a second, "sort-directed" version of $\alpha$-structural recursion in which one only has to give $X_\sigma$ when $\sigma = \mathsf{s}$ is a data-sort, and only has to give the functions $f_{(\_)}$ for constructors. Here is the statement of the new form of the recursion principle; the notations used in it are defined in Figure 2 and discussed below.

**Theorem 22 (second $\alpha$-structural recursion theorem).** *Let $\Sigma$ be a nominal signature. Suppose we are given a family of nominal sets $X = (X_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ indexed by the data-sorts of $\Sigma$, a finite set $A$ of atoms, and functions*

$$f_K \in X^{(\sigma)} \to_{\mathrm{fs}} X^{(\mathsf{s})} \quad ((K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}})$$

*all of which are supported by $A$ and satisfy*

$$(\exists \bar{a} \in \mathbb{A}^\sigma)\ \bar{a} \# A \ \ \&\ \ (\forall \bar{x} \in X^{|\sigma|})\ \bar{a} \sharp_\sigma \bar{x} \ \Rightarrow\ \bar{a} \# f_K(\bar{a}, \bar{x})_\sigma\ . \qquad (\mathrm{FCB}_K)$$

*Then there is a unique family of finitely supported functions $(\hat{f}_{\mathsf{s}} \in \mathsf{T}_\alpha(\Sigma)_{\mathsf{s}} \to_{\mathrm{fs}} X_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ with $supp(\hat{f}_{\mathsf{s}}) \subseteq A$ and satisfying*

$$(\forall \bar{a} \in \mathbb{A}^\sigma)\ \bar{a} \# A \ \Rightarrow$$
$$(\forall \bar{e} \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|})\ \bar{a} \sharp_\sigma \bar{e} \ \Rightarrow\ \hat{f}_{\mathsf{s}}(K\,\bar{a}.\,\bar{e}) = f_K(\bar{a}, \hat{f}^{|\sigma|}\,\bar{e})_\sigma \qquad (46)$$

*for each $(K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}}$.*

In this theorem we start with a family of nominal sets $X = (X_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ indexed by the data-sorts of the signature $\Sigma$ and with a family of finitely supported functions $(f_K \mid K \in \Sigma_{\mathrm{C}})$ indexed by the constructors of $\Sigma$. The domain $X^{(\sigma)}$ of $f_K$ is obtained from the arity $\sigma$ of $K$ by interpreting each atom-sort as the corresponding nominal set of atoms (Example 7(i)), each data-sort as given by $X$, the unit arity as the one-element discrete nominal set (Example 7(iv)), pair arities using products of nominal sets (Section 3.2), and atom-binding arities just using product with nominal sets of atoms. The aim is to use this data to specify some functions $\hat{f}_{\mathsf{s}}$ mapping $\alpha$-terms $e : \mathsf{s}$ to elements $\hat{f}_{\mathsf{s}}\,e \in X_{\mathsf{s}}$ by giving recursion equations as in (46), with one (conditional) equation for each way of forming $\alpha$-terms of data-sort, i.e. for each constructor $K : \sigma \to \mathsf{s}$ in $\Sigma$. The conditional equation for $K$ specifies the effect of $\hat{f}_{\mathsf{s}}$ not for arbitrary $\alpha$-terms constructed with $K$, but rather just for those of the form $K\,\bar{a}.\,\bar{e}$ where $\bar{a} \# A$ and $\bar{a} \sharp_\sigma \bar{e}$ hold. Here $\bar{a} \in \mathbb{A}^\sigma$ is a nested tuple of distinct atoms matching the *binding* occurrences of atom-sorts in the arity $\sigma$; $\bar{e} \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|}$ is a nested tuple of $\alpha$-terms matching the *non-binding* occurrences of atom-sorts and the occurrences of data-sorts in $\sigma$; and the operation $\bar{a}, \bar{e} \mapsto \bar{a}.\,\bar{e}$ assembles these two nested tuples into an $\alpha$-term of arity $\sigma$, to which $K$ can be applied to get a constructed $\alpha$-term $K\,\bar{a}.\,\bar{e} : \mathsf{s}$. The equation in (46) is restricted by two conditions:

| $\sigma$ | $X^{(\sigma)}$ | $\mathbb{A}^{\sigma}$ | $X^{\lvert\sigma\rvert}$ |
|---|---|---|---|
| $\mathsf{a} \in \Sigma_{\mathrm{A}}$ | $\mathbb{A}_{\mathsf{a}}$ | $1$ | $\mathbb{A}_{\mathsf{a}}$ |
| $\mathsf{s} \in \Sigma_{\mathrm{D}}$ | $X_{\mathsf{s}}$ | $1$ | $X_{\mathsf{s}}$ |
| $1$ | $1$ | $1$ | $1$ |
| $\sigma_1 * \sigma_2$ | $X^{(\sigma_1)} \times X^{(\sigma_2)}$ | $\mathbb{A}^{\sigma_1} \otimes \mathbb{A}^{\sigma_2}$ | $X^{\lvert\sigma_1\rvert} \times X^{\lvert\sigma_2\rvert}$ |
| «a»$\sigma_1$ | $\mathbb{A}_{\mathsf{a}} \times X^{(\sigma_1)}$ | $\mathbb{A}_{\mathsf{a}} \otimes \mathbb{A}^{\sigma_1}$ | $X^{\lvert\sigma_1\rvert}$ |

The third column of the above table uses the **separated product** of nominal sets: $X \otimes Y \triangleq \{(x, y) \in X \times Y \mid x \mathrel{\#} y\}$ (with atom-permutation action the same as for the product $X \times Y$).

| $\sigma$ | $(\bar{a}, \bar{x}) \in \mathbb{A}^{\sigma} \times X^{\lvert\sigma\rvert}$ | $\bar{a} \sharp_{\sigma} \bar{x} \in \mathbb{B}$ | $(\bar{a}, \bar{x})_{\sigma} \in X^{(\sigma)}$ |
|---|---|---|---|
| $\mathsf{a} \in \Sigma_{\mathrm{A}}$ | $((), a)$ | *true* | $a$ |
| $\mathsf{s} \in \Sigma_{\mathrm{D}}$ | $((), x)$ | *true* | $x$ |
| $1$ | $((), ())$ | *true* | $()$ |
| $\sigma_1 * \sigma_2$ | $((\bar{a}_1, \bar{a}_2), (\bar{x}_1, \bar{x}_2))$ | $\bar{a}_1 \sharp_{\sigma_1} \bar{x}_1$ & $\bar{a}_2 \sharp_{\sigma_2} \bar{x}_2$ & $\bar{a}_1 \mathrel{\#} \bar{x}_2$ & $\bar{a}_2 \mathrel{\#} \bar{x}_1$ | $((\bar{a}_1, \bar{x}_1)_{\sigma_1}, (\bar{a}_2, \bar{x}_2)_{\sigma_2})$ |
| «a»$\sigma_1$ | $((a, \bar{a}_1), \bar{x}_1)$ | $\bar{a}_1 \sharp_{\sigma} \bar{x}_1$ | $(a, (\bar{a}_1, \bar{x}_1)_{\sigma})$ |

| $\sigma$ | $(\bar{a}, \bar{e}) \in \mathbb{A}^{\sigma} \times \mathsf{T}_{\alpha}(\Sigma)^{\lvert\sigma\rvert}$ | $\bar{a}.\, \bar{e} \in \mathsf{T}_{\alpha}(\Sigma)_{\sigma}$ | $\hat{f}^{\lvert\sigma\rvert}\, \bar{e} \in X^{\lvert\sigma\rvert}$ |
|---|---|---|---|
| $\mathsf{a} \in \Sigma_{\mathrm{A}}$ | $((), a)$ | $a$ | $a$ |
| $\mathsf{s} \in \Sigma_{\mathrm{D}}$ | $((), e)$ | $e$ | $\hat{f}_{\mathsf{s}}\, e$ |
| $1$ | $((), ())$ | $()$ | $()$ |
| $\sigma_1 * \sigma_2$ | $((\bar{a}_1, \bar{a}_2), (\bar{e}_1, \bar{e}_2))$ | $(\bar{a}_1.\, \bar{e}_1, \bar{a}_2.\, \bar{e}_2)$ | $(\hat{f}^{\lvert\sigma_1\rvert}\, \bar{e}_1, \hat{f}^{\lvert\sigma_2\rvert}\, \bar{e}_2)$ |
| «a»$\sigma_1$ | $((a, \bar{a}_1), \bar{e}_1)$ | $a.\, (\bar{a}_1.\, \bar{e}_1)$ | $\hat{f}^{\lvert\sigma_1\rvert}\, \bar{e}_1$ |

In the second column of the above table, $\mathsf{T}_{\alpha}(\Sigma)^{\lvert\sigma\rvert}$ indicates the construction $X \mapsto X^{\lvert\sigma\rvert}$ from the first table applied to the data-sort indexed family $X = (\mathsf{T}_{\alpha}(\Sigma)_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$.

Figure 2: Definitions used in Theorem 22

– the first, $\bar{a} \# A$, just requires that the atoms in $\bar{a}$ do not occur in the common finite support of the functions $f_K$;

– the second, $\bar{a} \sharp_\sigma \bar{e}$, ensures that in addition to the atoms in $\bar{a}$ being mutually distinct (by virtue of the definition of $\mathbb{A}^\sigma$), they avoid the support of the constituents of $\bar{e}$ in an appropriate way (the precise definition of "appropriate" being given in Figure 2).

The theorem guarantees the unique existence of such functions on $\alpha$-terms provided the functions $f_K$ satisfy the *freshness condition on binders* given by (FCB$_K$). This asserts the existence of a nested tuple $\bar{a}$ of distinct atoms that can appear in binding positions in elements of $X^{(\sigma)}$ (i.e. $\bar{a} \in \mathbb{A}^\sigma$) such that:

– the atoms are distinct from $A$, i.e. $\bar{a} \# A$ (they are also mutually distinct by definition of $\mathbb{A}^\sigma$);

– whenever $\bar{x}$ is a nested tuple of atoms and $X$-elements that can appear in non-binding positions in elements of $X^{(\sigma)}$ (i.e. $\bar{x} \in X^{|\sigma|}$) for which $\bar{a}$ is suitably fresh, i.e. satisfying $\bar{a} \sharp_\sigma \bar{x}$,[9] then assembling $\bar{a}$ and $\bar{x}$ into an element $(\bar{a}, \bar{x})_\sigma \in X^{(\sigma)}$, $f_K$ maps this element to one in $X^{(\mathsf{s})} = X_\mathsf{s}$ whose support does not contain any of the atoms in $\bar{a}$ (i.e. $\bar{a} \# f_K(\bar{a}, \bar{x})_\sigma$).

The easiest way I know of proving Theorem 22 is to derive it from the following "sort-directed" version of $\alpha$-structural induction, which uses notations that are defined in Figures 2 and 3, and which are discussed below.

**Theorem 23 (second $\alpha$-structural induction theorem).** *Let $\Sigma$ be a nominal signature. Suppose we are given a family of finitely supported subsets $(S_\mathsf{s} \in P_{\mathrm{fs}}(\mathsf{T}_\alpha(\Sigma)_\mathsf{s}) \mid \mathsf{s} \in \Sigma_\mathrm{D})$ indexed by the data-sorts of $\Sigma$ and all supported by a finite set of atoms $A$. Then to prove that $S_\mathsf{s}$ is the whole of $\mathsf{T}_\alpha(\Sigma)_\mathsf{s}$ for all $\mathsf{s} \in \Sigma_\mathrm{D}$, it suffices to show for each constructor $(K : \sigma \to \mathsf{s}) \in \Sigma_\mathrm{C}$ that*

$$(\exists \bar{a} \in \mathbb{A}^\sigma)\ \bar{a} \# A\ \ \&\ \ (\forall \bar{e} \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|})\ \bar{a} \sharp_\sigma \bar{e}\ \ \&\ \ \bar{e} \in S^{|\sigma|}\ \Rightarrow\ K\,\bar{a}.\bar{e} \in S_\mathsf{s}\ . \qquad (\mathrm{IH}_K)$$

In this theorem we start with a family of subsets $S_\mathsf{s} \subseteq \mathsf{T}_\alpha(\Sigma)_\mathsf{s}$ of $\alpha$-terms whose arities are data-sorts and that are all supported by some finite set of atoms $A$. We wish to prove that every $t : \mathsf{s}$ is in $S_\mathsf{s}$ (for all $\mathsf{s} \in \Sigma_\mathrm{D}$). The theorem guarantees this provided each constructor $(K : \sigma \to \mathsf{s}) \in \Sigma_\mathrm{C}$ satisfies the induction hypothesis given by (IH$_K$). This asserts the existence of a nested tuple $\bar{a}$ of distinct atoms that can appear in binding positions in $\alpha$-terms of arity $\sigma$ (i.e. $\bar{a} \in \mathbb{A}^\sigma$) such that:

– the atoms are distinct from $A$, i.e. $\bar{a} \# A$ (they are also mutually distinct by definition of $\mathbb{A}^\sigma$);

[9]This relation $\sharp_\sigma$, defined in Figure 2, is a subtlety of the freshness condition on binders that is not apparent in the simpler First $\alpha$-Structural Recursion Theorem.

| $\sigma$ | $S^{\lvert\sigma\rvert} \in P_{\mathrm{fs}}(\mathsf{T}_\alpha(\Sigma)^{\lvert\sigma\rvert})$ | $S_\sigma \in P_{\mathrm{fs}}(\mathsf{T}_\alpha(\Sigma)_\sigma)$ |
|---|---|---|
| $\mathsf{a} \in \Sigma_{\mathrm{A}}$ | $\mathbb{A}_{\mathsf{a}}$ | $\{a \mid a \in \mathbb{A}_{\mathsf{a}}\}$ |
| $\mathsf{s} \in \Sigma_{\mathrm{D}}$ | $S_{\mathsf{s}}$ | $S_{\mathsf{s}}$ |
| $1$ | $1$ | $\{()\}$ |
| $\sigma_1 * \sigma_2$ | $S^{\lvert\sigma_1\rvert} \times S^{\lvert\sigma_2\rvert}$ | $\{(e_1, e_2) \mid e_1 \in S_{\sigma_1} \ \&\ e_2 \in S_{\sigma_2}\}$ |
| «a»$\sigma_1$ | $S^{\lvert\sigma_1\rvert}$ | $\{a.\, e_1 \mid a \in \mathbb{A}_{\mathsf{a}} - \mathit{supp}(S_{\sigma_1}) \ \&\ e_1 \in S_{\sigma_1}\}$ |

Figure 3: Definitions used in Theorem 23

– whenever $\bar{e}$ is a nested tuple of $\alpha$-terms that can appear in non-binding positions in an $\alpha$-term of arity $\sigma$ (i.e. $\bar{e} \in \mathsf{T}_\alpha(\Sigma)^{\lvert\sigma\rvert}$) for which $\bar{a}$ is suitably fresh, i.e. satisfying $\bar{a} \ \sharp_\sigma \ \bar{e}$, then assembling $\bar{a}$ and $\bar{e}$ into the $\alpha$-term $\bar{a}.\, \bar{e} : \sigma$, the constructed $\alpha$-term $K\, \bar{a}.\, \bar{e}$ must lie in the subset $S_{\mathsf{s}}$.

The proof of Theorem 23 is given in Appendix A and the proof of Theorem 22 in Appendix B. In Sections 5.1 and 5.2 we explore what these principles look like for particular nominal signatures, using the examples from Section 2.2.

**Remark 24** (**atom-abstraction** and the **initial algebra property**). For each atom-sort $\mathsf{a} \in \mathbb{AS}$ and each nominal set $X$, let $[\mathbb{A}_{\mathsf{a}}]X$ denote the set of equivalence classes of pairs $(a, x) \in \mathbb{A}_{\mathsf{a}} \times X$ for the equivalence relation $(a, x) \sim (a', x')$ given by

$$(\exists a'' \in \mathbb{A}_{\mathsf{a}})\ a'' \# (a, x, a', x') \ \ \&\ \ (a\ a'') \cdot x = (a'\ a'') \cdot x' \ . \tag{47}$$

The relation $\sim$ is evidently reflexive and symmetric; to see that it is also transitive, one first applies Theorem 14 with $S = \{a'' \in \mathbb{A}_{\mathsf{a}} \mid (a\ a'') \cdot x = (a'\ a'') \cdot x'\}$ and $A = \mathit{supp}(a, x, a', x')$ to show that $(a, x) \sim (a', x')$ holds if and only if

$$(\forall a'' \in \mathbb{A}_{\mathsf{a}})\ a'' \# (a, x, a, x') \ \Rightarrow\ (a\ a'') \cdot x = (a'\ a'') \cdot x' \ . \tag{48}$$

We write $[a]x$ for the $\sim$-equivalence class of the pair $(a, x)$ and call it an **atom-abstraction**. It follows from Lemma 13 that $\sim$ is a nominal subset of $\mathbb{A}_{\mathsf{a}} \times X$. So the quotient $[\mathbb{A}_{\mathsf{a}}]X$ is a nominal set as in Section 3.3. One can calculate that the support of each element $[a]x$ of $[\mathbb{A}_{\mathsf{a}}]X$ is $\mathit{supp}(x) - \{a\}$.

Using these atom-abstraction nominal sets, it is possible to give an *initial algebra* characterisation of $(\mathsf{T}_\alpha(\Sigma)_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ that is equivalent to Theorem 22. Consider the category whose objects are families of nominal sets $X = (X_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ indexed by the data-sorts of $\Sigma$, and whose morphisms $f : X \rightarrow X'$ are indexed families $f = (f_s \in X_{\mathsf{s}} \rightarrow_{\mathrm{fs}} X'_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ of functions with empty support (i.e. functions that respect the action of all atom-permutations). The constructors of $\Sigma$ determine a functor $F_\Sigma$ from this category to itself, defined in Figure 4. An $F_\Sigma$-**algebra** is simply an object $I$ equipped with a morphism $i : F_\Sigma\, I \rightarrow I$. Such

Action of $F_\Sigma$ on objects:

$$(F_\Sigma\, X)_{\mathsf{s}} \triangleq \sum_{(K:\sigma\to\mathsf{s})\in\Sigma_{\mathrm{C}}} X^{[\sigma]} = \{(K,x) \mid (K:\sigma\to\mathsf{s})\in\Sigma_{\mathrm{C}} \;\;\&\;\; x\in X^{[\sigma]}\}\ .$$

Action of $F_\Sigma$ on morphisms:

$$(F_\Sigma\, f)_{\mathsf{s}}(K,x) \triangleq (K, f^{[\sigma]}\, x)\ .$$

| $\sigma$ | $X^{[\sigma]}$ | $x \in X^{[\sigma]}$ | $\mapsto$ | $f^{[\sigma]}\, x \in X'^{[\sigma]}$ |
|---|---|---|---|---|
| $\mathsf{a}\in\Sigma_{\mathrm{A}}$ | $\mathbb{A}_{\mathsf{a}}$ | $a$ | $\mapsto$ | $a$ |
| $\mathsf{s}$ | $S_{\mathsf{s}}$ | $x$ | $\mapsto$ | $f_{\mathsf{s}}\, x$ |
| $1$ | $1$ | $()$ | $\mapsto$ | $()$ |
| $\sigma_1 * \sigma_2$ | $X^{[\sigma_1]}\times X^{[\sigma_2]}$ | $(x_1,x_2)$ | $\mapsto$ | $(f^{[\sigma_1]}\, x_1, f^{[\sigma_2]}\, x_2)$ |
| «a»$\sigma_1$ | $[\mathbb{A}_{\mathsf{a}}](X^{[\sigma_1]})$ | $[a]x_1$ | $\mapsto$ | $[a](f^{[\sigma_1]}\, x_1)$ |

Figure 4: Functor $F_\Sigma$ associated with a nominal signature $\Sigma$

an algebra is **initial** if for any other such algebra $f : F_\Sigma\, X \to X$, there is a unique morphism $\hat{f} : I \to X$ so that

$$\begin{tikzcd} F_\Sigma\, I \arrow[r, "i"] \arrow[d, "F_\Sigma\, \hat{f}"'] & I \arrow[d, "\hat{f}"] \\ F_\Sigma\, X \arrow[r, "f"'] & X \end{tikzcd} \tag{49}$$

commutes, i.e. satisfying

$$(\forall \mathsf{s}\in\Sigma_{\mathrm{D}})(\forall (K:\sigma\to\mathsf{s})\in\Sigma_{\mathrm{C}})(\forall x\in I^{[\sigma]})\ \hat{f}_{\mathsf{s}}(i_{\mathsf{s}}(K,x)) = f_{\mathsf{s}}(K, \hat{f}^{[\sigma]}\, x)\ .$$

Standard category-theoretic results give that $i$ is an isomorphism and that the initial algebra $(I,i)$ is unique up to isomorphism. Theorem 22 can be used to prove that $(\mathsf{T}_\alpha(\Sigma)_{\mathsf{s}} \mid \mathsf{s}\in\Sigma_{\mathrm{D}})$ is the (object part of) an initial $F_\Sigma$-algebra. Conversely, one can give a direct inductive construction of an initial $F_\Sigma$-algebra and use the initial algebra property (49) to deduce Theorem 22: see [13, Section 6].

## 5.1 Example: $\lambda$-calculus (with *letrec*)

Let $\Sigma$ be the nominal signature from Example 1. Thus $\mathsf{T}(\Sigma)_{\mathsf{t}}$ is the set $\Lambda$ of abstract syntax trees for $\lambda$-calculus with *letrec*; and $\mathsf{T}_\alpha(\Sigma)_{\mathsf{t}}$ is the quotient $\Lambda/{=_\alpha}$ of that set by the usual notion of $\alpha$-equivalence—in other words $\mathsf{T}_\alpha(\Sigma)_\mathsf{t}$ is what is normally meant by the set of all (open or closed) untyped $\lambda$-terms with local recursive function declarations. Suppose we are given a nominal set $X$ and functions

$$f_V \in \mathbb{A}_\mathsf{v} \to_{\mathrm{fs}} X \tag{50}$$
$$f_A \in X \times X \to_{\mathrm{fs}} X \tag{51}$$
$$f_L \in \mathbb{A}_\mathsf{v} \times X \to_{\mathrm{fs}} X \tag{52}$$
$$f_{\mathit{Letrec}} \in \mathbb{A}_\mathsf{v} \times ((\mathbb{A}_\mathsf{v} \times X) \times X) \to_{\mathrm{fs}} X \tag{53}$$

all supported by a finite set of atoms $A$. Applying the definitions in Figure 2, one finds that the conditions (FCB$_K$) for $K = V, A$ are equivalent to $\mathit{true}$, that (FCB$_{\mathit{Lam}}$) is equivalent to

$$(\exists a \in \mathbb{A}_\mathsf{v})\ a \notin A \ \ \&\ \ (\forall x \in X)\ a \mathbin{\#} f_L(a, x) \tag{54}$$

and that (FCB$_{\mathit{Letrec}}$) is equivalent to

$$(\exists a, a' \in \mathbb{A}_\mathsf{v})\ a \neq a' \ \ \&\ \ a, a' \notin A \ \ \&\ \ (\forall x, x' \in X)\ a' \mathbin{\#} x \ \Rightarrow \\ (a, a') \mathbin{\#} f_{\mathit{Letrec}}(a, ((a', x'), x))\ . \tag{55}$$

So for this nominal signature, Theorem 22 gives us the following recursion principle. We state it using the usual concrete syntax for $\lambda$-calculus and using the fact (noted in Example 7(iii)) that the support of a term $e \in \Lambda/{=_\alpha}$ is its finite set $\mathit{fv}(e)$ of free variables.

**Theorem 25.** *Let $\Lambda/{=_\alpha}$ be the nominal set of $\alpha$-equivalence classes of $\lambda$-terms with local recursive function declarations:*

$$e ::= x \mid e\,e \mid \lambda x.e \mid \mathit{letrec}\,x\,x = e\ \mathit{in}\ e$$

*where the variables $x$ are drawn from the nominal set $\mathbb{A}_\mathsf{v}$ of atoms of some fixed sort $\mathsf{v}$. Given any nominal set $X$ and functions as in (50)–(53) all supported by some finite set of atoms $A$ and with $f_L$ and $f_{\mathit{Letrec}}$ satisfying (54) and (55), then there is a unique function $\hat{f} \in (\Lambda/{=_\alpha} \to_{\mathrm{fs}} X)$ supported by $A$ and satisfying*

$$\hat{f}(x) = f_V(x) \tag{56}$$
$$\hat{f}(e_1\,e_2) = f_A(\hat{f}\,e_1, \hat{f}\,e_2) \tag{57}$$
$$x \notin A \ \Rightarrow\ \hat{f}(\lambda x.e) = f_L(x, \hat{f}\,e) \tag{58}$$
$$x, y \notin A \ \ \&\ \ x \notin \mathit{fv}(e_2) \cup \{y\} \ \Rightarrow \\ \hat{f}(\mathit{letrec}\,y\,x = e_1\ \mathit{in}\ e_2) = f_{\mathit{Letrec}}(y, ((x, \hat{f}\,e_1), \hat{f}\,e_2))\ . \tag{59}$$

□

Turning to the induction principle for this signature, if $S \subseteq \Lambda/{=_\alpha}$ is a set of terms supported by a finite set of atoms $A$, then (IH$_V$), (IH$_A$), (IH$_L$) and (IH$_{Letrec}$) are equivalent to

$$(\forall x \in \mathbb{A}_\mathsf{v})\ x \in S \tag{60}$$

$$(\forall e_1, e_2 \in S)\ e_1\, e_2 \in S \tag{61}$$

$$(\exists x \in \mathbb{A}_\mathsf{v})\ x \notin A \ \ \&\ \ (\forall e \in S)\ \lambda x.\, e \in S \tag{62}$$

$$\begin{aligned}&(\exists x, y \in \mathbb{A}_\mathsf{v})\ x \neq y \ \ \&\ \ x, y \notin A \ \ \&\\ &\qquad (\forall e_1, e_2 \in S)\ x \notin \mathit{fv}(e_2) \ \Rightarrow\ \mathit{letrec}\, y\, x = e_1\ \mathit{in}\, e_2 \in S\end{aligned} \tag{63}$$

respectively. So for this signature Theorem 23 says that $S$ contains all terms if it satisfies (60)–(63).

**Example 26** (**capture-avoiding substitution**)**.** The example mentioned in the Introduction of capture-avoiding substitution of $\lambda$-terms, $\hat{s}_{x,e} \in \Lambda/{=_\alpha} \to \Lambda/{=_\alpha}$, is obtained from the above theorem by taking $X$ to be the nominal set $\Lambda/{=_\alpha}$. Given $x \in \mathbb{A}_\mathsf{v}$ and $e \in X$, then $\hat{s}_{x,e}$ is given by $\hat{f}$ where

$$\begin{aligned}f_V(y) &\triangleq \begin{cases} e & \text{if } y = x \\ y & \text{if } y \neq x\end{cases}\\ f_A(e_1, e_2) &\triangleq e_1\, e_2\\ f_L(y, e_1) &\triangleq \lambda y.\, e_1\\ f_{Letrec}(z, ((y, e_1), e_2)) &\triangleq \mathit{letrec}\, z\, y = e_1\ \mathit{in}\, e_2\\ A &\triangleq \mathit{fv}(e) \cup \{x\}\ .\end{aligned}$$

Condition (54) is satisfied because, as noted in Example 7(iii), for each $e_1 \in X = \Lambda/{=_\alpha}$, $\mathit{supp}(e_1)$ is the finite set $\mathit{fv}(e_1)$ of free variables of $e_1$; in particular we have $y \mathrel{\#} f_{Lam}(y, e_1) = \lambda y.\, e_1$ simply because $y \notin \mathit{fv}(\lambda y.\, e_1) = \mathit{fv}(e_1) - \{y\}$. Similarly, condition (55) is satisfied because

$$\mathit{fv}(\mathit{letrec}\, z\, y = e_1\ \mathit{in}\, e_2) = (\mathit{fv}(e_1) - \{y, z\}) \cup (\mathit{fv}(e_2) - \{z\})$$

so that $(y, z) \mathrel{\#} f_{Letrec}(z, ((y, e_1), e_2)) = \mathit{letrec}\, z\, y = e_1\ \mathit{in}\, e_2$ provided $y \notin \mathit{fv}(e_2)$. Note that $f_A$, $f_L$ and $f_{Letrec}$ are all supported by the empty set of atoms, whereas $f_V$ is supported by $\mathit{fv}(e) \cup \{x\}$; so this is what we take for the common support $A$. Thus the conditions on the recursion equations in (58) and (59) correspond precisely to the conditions in (8) and (9).

**Example 27** (**recursion with varying parameters**)**.** Norrish [18, p 245] considers a variant of capture-avoiding substitution whose definition involves recursion with varying parameters; it motivates the parametrised recursion principle he presents in that paper. The $\alpha$-structural recursion principles we have given here do not involve extra parameters, let alone varying ones; nevertheless it is possible to derive parameterised versions from them. In the case of ordinary structural recursion, one can derive a parameterised version from an unparameterised one by currying parameters and defining maps into function sets using Theorem 3. In the presence of binders, one has to do something slightly more complicated, involving the Freshness Theorem 16, to derive a parameterised (FCB) from the unparameterised version of the condition.

Let us see how this works for Norrish's example, using the nominal signature for the pure $\lambda$-calculus obtained from Example 1 by deleting the *Letrec* constructor.. Fixing on a pair of atoms $x_1, x_2 \in \mathbb{A}_{\mathsf{v}}$, we seek a function $s \in (\Lambda/{=_\alpha}) \to_{\mathrm{fs}} (\Lambda/{=_\alpha}) \to_{\mathrm{fs}} (\Lambda/{=_\alpha})$ satisfying for all $y, e, e_1, e_2$:

$$s(y)(e) = \begin{cases} e & \text{if } y = x_1 \\ y & \text{if } y \neq x_1 \end{cases} \tag{64}$$

$$s(e_1\, e_2)(e) = (s(e_1)(e))\,(s(e_2)(e)) \tag{65}$$

$$y \notin \mathit{fv}(e) \cup \{x_1, x_2\} \;\Rightarrow\; s(\lambda y.\, e_1)(e) = \lambda y.\, s(e_1)(x_2\, e) \tag{66}$$

Thus the same parameter $e$ appears in each clause defining $s(e_1)(e)$ by recursion on the structure of $e_1$ except for clause (66), where the application term $x_2\, e$ appears instead. Such a function $s$ can be obtained from Theorem 25 (restricted to pure $\lambda$-terms) as $s = \hat{f}$ if we take $X$ to be the nominal set $(\Lambda/{=_\alpha}) \to_{\mathrm{fs}} (\Lambda/{=_\alpha})$ and use the functions

$$\begin{aligned} f_V &\triangleq \lambda y \in \mathbb{A}_{\mathsf{v}}.\lambda e \in (\Lambda/{=_\alpha}).\, \mathit{if}\ y = x_1\ \mathit{then}\ e\ \mathit{else}\ y \\ f_A &\triangleq \lambda(\xi_1, \xi_2) \in X \times X.\, \lambda e \in (\Lambda/{=_\alpha}).\, (\xi_1\, e)\,(\xi_2\, e) \\ f_L &\triangleq \lambda(y, \xi_1) \in \mathbb{A}_{\mathsf{v}} \times X.\lambda e \in (\Lambda/{=_\alpha}).\, \mathit{fresh}(h(y, \xi_1, e)) \end{aligned}$$

where the last clause uses Theorem 16 applied to the finitely supported function $h(y, \xi_1, e) \in \mathbb{A}_{\mathsf{v}} \to_{\mathrm{fs}} (\Lambda/{=_\alpha})$ that maps each $y' \in \mathbb{A}_{\mathsf{v}}$ to

$$h(y, \xi_1, e)(y') \;\triangleq\; \lambda y'.\, ((y\ y') \cdot \xi_1)(x_2\, e)$$

This function is easily seen to satisfy the property (30) needed to apply the theorem. All the above functions are supported by $A \triangleq \{x_1, x_2\}$. Properties (56) and (57) of $\hat{f}$ give (64) and (65) respectively. When $y \neq x_1, x_2$, property (58) gives us $\hat{f}(\lambda y.\, e_1) = f_L(y, \hat{f}\, e_1) = \mathit{fresh}(h(y, \hat{f}\, e_1, e))$. So if $y \mathrel{\#} (x_1, x_2, e)$, picking any $y' \mathrel{\#} (x_1, x_2, e, e_1, h)$, then by Theorem 16 we have $\mathit{fresh}(h(y, \hat{f}\, e_1, e)) = h(y, \hat{f}\, e_1, e)(y') \triangleq \lambda y'.\, ((y\ y') \cdot (\hat{f}\, e_1))(x_2\, e) = \lambda y'.\, (y\ y') \cdot (\hat{f}\, e_1\, (x_2\, e))$. Hence by definition of $=_\alpha$, $\hat{f}(\lambda y.\, e_1) = \lambda y.\, \hat{f}\, e_1\, (x_2\, e)$, as required for (66).

## 5.2 Example: $\pi$-calculus

Let $\Sigma$ be the nominal signature from Example 2. Suppose we are given nominal sets $X_{\mathsf{proc}}$, $X_{\mathsf{gsum}}$, $X_{\mathsf{pre}}$ and functions

$$\begin{array}{rcl} f_{\mathit{Gsum}} & \in & X_{\mathsf{gsum}} \to_{\mathrm{fs}} X_{\mathsf{proc}} \\ f_{\mathit{Par}} & \in & X_{\mathsf{proc}} \times X_{\mathsf{proc}} \to_{\mathrm{fs}} X_{\mathsf{proc}} \\ f_{\mathit{Res}} & \in & \mathbb{A}_{\mathsf{chan}} \times X_{\mathsf{proc}} \to_{\mathrm{fs}} X_{\mathsf{proc}} \\ f_{\mathit{Rep}} & \in & X_{\mathsf{proc}} \to_{\mathrm{fs}} X_{\mathsf{proc}} \end{array}$$

$$\begin{array}{rcl} f_{\mathit{Zero}} & \in & 1 \to_{\mathrm{fs}} X_{\mathsf{gsum}} \\ f_{\mathit{Pre}} & \in & X_{\mathsf{pre}} \to_{\mathrm{fs}} X_{\mathsf{gsum}} \\ f_{\mathit{Plus}} & \in & X_{\mathsf{gsum}} \times X_{\mathsf{gsum}} \to_{\mathrm{fs}} X_{\mathsf{gsum}} \end{array}$$

$$\begin{array}{rcl} f_{\mathit{Out}} & \in & (\mathbb{A}_{\mathsf{chan}} \times \mathbb{A}_{\mathsf{chan}}) \times X_{\mathsf{proc}} \to_{\mathrm{fs}} X_{\mathsf{pre}} \\ f_{\mathit{In}} & \in & \mathbb{A}_{\mathsf{chan}} \times (\mathbb{A}_{\mathsf{chan}} \times X_{\mathsf{proc}}) \to_{\mathrm{fs}} X_{\mathsf{pre}} \\ f_{\mathit{Tau}} & \in & 1 \to_{\mathrm{fs}} X_{\mathsf{pre}} \\ f_{\mathit{Match}} & \in & (\mathbb{A}_{\mathsf{chan}} \times \mathbb{A}_{\mathsf{chan}}) \times X_{\mathsf{pre}} \to_{\mathrm{fs}} X_{\mathsf{pre}} \end{array}$$

all supported by a finite set of atoms $A$. The conditions (FCB$_{\mathit{Res}}$) and (FCB$_{\mathit{In}}$) are equivalent to

$$(\exists a \in \mathbb{A}_{\mathsf{chan}})\ a \notin A \ \ \& \ \ (\forall x \in X_{\mathsf{proc}})\ a \mathrel{\#} f_{\mathit{Res}}(a, x) \tag{67}$$

$$\begin{array}{l}(\exists a \in \mathbb{A}_{\mathsf{chan}})\ a \notin A \ \ \& \\ \qquad (\forall a' \in \mathbb{A}_{\mathsf{chan}})(\forall x \in X_{\mathsf{proc}})\ a \neq a' \ \Rightarrow \ a \mathrel{\#} f_{\mathit{In}}(a', (a, x))\end{array} \tag{68}$$

respectively; and conditions (FCB$_K$) for $K \neq \mathit{Res}, \mathit{In}$ are all equivalent to $\mathit{true}$. So if $f_{\mathit{Res}}$ and $f_{\mathit{In}}$ satisfy (67) and (68), then by Theorem 22, there are unique finitely supported functions $\hat{f}_{\mathsf{s}} \in \mathsf{T}_\alpha(\Sigma)_{\mathsf{s}} \to_{\mathrm{fs}} X_{\mathsf{s}}$ (for $\mathsf{s} = \mathsf{proc}, \mathsf{gsum}, \mathsf{pre}$) all supported by $A$ and satisfying for all $e, e_1, e_2, a_1, a_2, a, a'$ of suitable arity

$$\hat{f}(K\,e) = f_K(\hat{f}\,e) \qquad (K = \mathit{Gsum}, \mathit{Rep}, \mathit{Pre}) \tag{69}$$

$$\hat{f}(K(e_1, e_2)) = f_K(\hat{f}\,e_1, \hat{f}\,e_2) \qquad (K = \mathit{Par}, \mathit{Plus}) \tag{70}$$

$$\hat{f}(K()) = f_K() \qquad (K = \mathit{Zero}, \mathit{Tau}) \tag{71}$$

$$\hat{f}(K((a_1, a_2), e)) = f_K((a_1, a_2), \hat{f}\,e) \qquad (K = \mathit{Out}, \mathit{Match}) \tag{72}$$

$$a \notin A \ \Rightarrow \ \hat{f}(\mathit{Res}\,a.\,e) = f_{\mathit{Res}}(a, \hat{f}\,e) \tag{73}$$

$$a \notin A \ \ \& \ \ a \neq a' \ \Rightarrow \ \hat{f}(\mathit{In}(a', a.\,e)) = f_{\mathit{In}}(a', (a, \hat{f}\,e)) \tag{74}$$

where we have abbreviated $\hat{f}_{\mathsf{s}}(e)$ to $\hat{f}\,e$.

We leave the reader to work out what induction principle Theorem 23 gives for this signature.

## 6 Extended Example: Normalisation by Evaluation

One of the most important aspects of nominal sets is that they provide, via the notion of finite support, a notion of *freshness* of names with respect to mathematical structures. This notion generalises the usual "not a free variable of" relation from finite syntactical structures to infinite objects (sets, functions, ...) where there is no obvious notion of free name. The theory comes into its own in situations where syntax and semantics have to be considered together and yet one still needs a workable notion of fresh name. We give an example in this section by treating **normalisation by evaluation** (**NBE**) for simply typed $\lambda$-calculus [3]. This produces $\beta\eta$-long normal forms for typed $\lambda$-terms by first taking their denotational semantics in the standard, extensional functions model of the calculus over a ground type of syntax trees and then composing with a *reification* function that turns elements of the denotational model back into syntax (in normal form). When reifying an extensional function into a $\lambda$-abstraction one wants to choose a fresh name $v$ for the $\lambda$-bound variable; but as the survey [8, p 157] eloquently puts it when discussing an informal version of the reification function

> "The problem is the "$v$ fresh" condition; what exactly does it mean? Unlike such conditions as "$x$ does not occur free in $E$", it is not even locally checkable whether a variable is fresh; freshness is a global property, defined with respect to a term that may not even be fully constructed yet."

As a result, treatments of NBE in the literature adopt some device for making the current finite context of used names explicit and threading it through all the mathematical definitions involved in the denotational and reification functions used for NBE: see [8, section 3.3] and [9], for example. This tends to obscure the simple, but informal idea behind reification. The authors of [8] go on to mention after the above quote that "freshness" can be characterised rigorously in the framework of [13]. The details are presented here for the first time, using nominal sets rather than the FM-set theory of *loc. cit.* The point is not just that this setting provides a rigorous explanation of "freshness" (since the formal approaches mentioned above also do that), but that it allows us to retain the essential simplicity of an informal account such as in [8, section 3.2].

### 6.1 Typed $\lambda$-terms and their $\beta\eta$-long normal forms

Rather than give a signature for raw $\lambda$-terms and then cut down to the well-typed ones using typing contexts, we make do with a simpler, but less extensible[10] treatment using explicitly typed variables. Let

$$Ty \triangleq \{\tau ::= \iota \mid \tau \dot{\to} \tau\} \tag{75}$$

[10]Such an approach is fine for simply-typed terms, but becomes unworkable for calculi with type variables or dependent types.

be the set of **simple type symbols** over a single **ground type** $\iota$. We assume given an injective function $\tau \in \mathit{Ty} \mapsto \mathsf{v}_\tau \in \mathbb{AS}$ that codes the simple type symbols as atom-sorts. We use atoms of sort $\mathsf{v}_\tau$ to stand for variables of type $\tau$ in the simply typed $\lambda$-calculus. Note that when $\tau$ and $\tau'$ are different simple type symbols, $\mathsf{v}_\tau$ and $\mathsf{v}_{\tau'}$ are different atom-sorts; so recalling the assumptions we made in section 2.1, the sets of atoms $\mathbb{A}_{\mathsf{v}_\tau}$ and $\mathbb{A}_{\mathsf{v}_{\tau'}}$ are disjoint.

Consider the nominal signature $\Sigma^{\text{STL}}$ with

| atom-sorts | data-sorts | constructors |
|---|---|---|
| $\mathsf{v}_\tau$ | $\mathsf{t}_\tau$ | $\mathit{Vr}_\tau : \mathsf{v}_\tau \longrightarrow \mathsf{t}_\tau$ |
| | | $\mathit{Ap}_{\tau,\tau'} : \mathsf{t}_{\tau \dot{\to} \tau'} * \mathsf{t}_\tau \longrightarrow \mathsf{t}_{\tau'}$ |
| | | $\mathit{Lm}_{\tau,\tau'} : «\mathsf{v}_\tau»\mathsf{t}_{\tau'} \longrightarrow \mathsf{t}_{\tau \dot{\to} \tau'}$ |

as $\tau$ and $\tau'$ range over $\mathit{Ty}$. Thus the (nominal) set

$$\Lambda(\tau) \triangleq \mathsf{T}_\alpha(\Sigma^{\text{STL}})_{\mathsf{t}_\tau} \tag{76}$$

of $\alpha$-terms of arity $\mathsf{t}_\tau$ over $\Sigma^{\text{STL}}$ is precisely the usual set of $\alpha$-equivalence classes of abstract syntax trees for $\lambda$-terms of simple type $\tau \in \mathit{Ty}$, using variables that are explicitly tagged with types.

Next we give a nominal signature $\Sigma^{\text{LNF}}$ for $\beta\eta$-long normal forms:

| atom-sorts | data-sorts | constructors |
|---|---|---|
| $\mathsf{v}_\tau$ | $\mathsf{n}_\tau$ | $V_\tau : \mathsf{v}_\tau \longrightarrow \mathsf{u}_\tau$ |
| | $\mathsf{u}_\tau$ | $A_{\tau,\tau'} : \mathsf{u}_{\tau \dot{\to} \tau'} * \mathsf{n}_\tau \longrightarrow \mathsf{u}_{\tau'}$ |
| | | $L_{\tau,\tau'} : «\mathsf{v}_\tau»\mathsf{n}_{\tau'} \longrightarrow \mathsf{n}_{\tau \dot{\to} \tau'}$ |
| | | $I : \mathsf{u}_\iota \longrightarrow \mathsf{n}_\iota$ |

where $\tau$ and $\tau'$ range over $\mathit{Ty}$. The (nominal) set

$$N(\tau) \triangleq \mathsf{T}_\alpha(\Sigma^{\text{LNF}})_{\mathsf{n}_\tau} \tag{77}$$

corresponds to the set of $\alpha$-equivalence classes of abstract syntax trees for simply-typed $\lambda$-terms of type $\tau$ in **$\beta\eta$-long normal form**, whereas

$$U(\tau) \triangleq \mathsf{T}_\alpha(\Sigma^{\text{LNF}})_{\mathsf{u}_\tau} \tag{78}$$

corresponds to the set of $\alpha$-equivalence classes of **neutral** (or *atomic*) terms of type $\tau$; see [8, p 155], for example.

**Notation 28.** From now on we will use the following concrete, overloaded, but hopefully more familiar notations for $\alpha$-terms over the signatures $\Sigma^{\text{STL}}$ and $\Sigma^{\text{LNF}}$.

- Typical elements of $\mathbb{A}_{\mathsf{v}_\tau}$, $\Lambda(\tau)$, $N(\tau)$ and $U(\tau)$ will be written $x$, $e$, $n$ and $u$ respectively.

- $\mathit{Vr}_\tau\, x$ will be written just as $x$, $\mathit{Ap}_{\tau,\tau'}(e_1, e_2)$ as $e_1\, e_2$, and $\mathit{Lm}_{\tau,\tau'}\, x.\, e$ as $\lambda x : \tau.\, e$.

- $V_\tau\, x$ will be written just as $x$, $A_{\tau,\tau'}(u,n)$ as $u\,n$, $L_{\tau,\tau'}\,x.\,n$ as $\lambda x:\tau.\,n$ and $I\,u$ just as $u$.

Every $\beta\eta$-long normal form and every neutral term can be regarded as a $\lambda$-term of the corresponding type; indeed a very simple application of the second $\alpha$-structural recursion theorem for the nominal signature $\Sigma^{\mathrm{LNF}}$ tells us that there are functions

$$i_\tau \in N(\tau) \to_{\mathrm{fs}} \Lambda(\tau) \qquad\qquad j_\tau \in U(\tau) \to_{\mathrm{fs}} \Lambda(\tau) \tag{79}$$

supported by the empty set of atoms and satisfying for all $\tau, \tau' \in \mathit{Ty}$ and $x, u, n$ of suitable arity

$$\begin{aligned} j_\tau\, x &= x \\ j_{\tau'}(u\, n) &= (j_{\tau \dot{\to} \tau'}\, u)(i_\tau\, n) \\ i_{\tau \dot{\to} \tau'}(\lambda x : \tau.\, n) &= \lambda x : \tau.\, i_{\tau'}\, n \\ i_\iota\, u &= j_\iota\, u\ . \end{aligned}$$

We aim to use the technique of NBE [3] to define the **normalisation function**

$$\mathit{norm}_\tau \in \Lambda(\tau) \to_{\mathrm{fs}} N(\tau) \tag{80}$$

with the following properties:

$$(\forall \tau \in \mathit{Ty}, e_1, e_2 \in \Lambda(\tau))\ e_1 =_{\beta\eta} e_2 \ \Rightarrow\ \mathit{norm}_\tau\, e_1 = \mathit{norm}_\tau\, e_2 \tag{81}$$
$$(\forall \tau \in \mathit{Ty}, n \in N(\tau))\ \mathit{norm}_\tau(i_\tau\, n) = n \tag{82}$$
$$(\forall \tau \in \mathit{Ty}, e \in \Lambda(\tau))\ i_\tau(\mathit{norm}_\tau\, e) =_{\beta\eta} e\ . \tag{83}$$

Here $=_{\beta\eta} \subseteq \Lambda(\tau) \times \Lambda(\tau)$ is the usual relation of ***βη*-conversion** between ($\alpha$-equivalence classes of) $\lambda$-terms of the same simple type $\tau$. It is by definition the smallest congruence relation satisfying

$$\begin{aligned} &(\forall \tau, \tau' \in \mathit{Ty}, x \in \mathbb{A}_{\mathsf{v}_\tau}, e_1 \in \Lambda(\tau'), e_2 \in \Lambda(\tau)) \\ &\qquad (\lambda x : \tau.\, e_1) e_2 =_{\beta\eta} e_1[x := e_2] \end{aligned} \tag{84}$$

$$\begin{aligned} &(\forall \tau, \tau' \in \mathit{Ty}, e \in \Lambda(\tau \dot{\to} \tau'), x \in \mathbb{A}_{\mathsf{v}_\tau})\ x \mathrel{\#} e \ \Rightarrow \\ &\qquad e =_{\beta\eta} \lambda x : \tau.\, e\, x\ . \end{aligned} \tag{85}$$

Here $e_1[x := e_2]$ indicates the **capture-avoiding substitution** of $e_2$ for all free occurrences of $x$ in $e_1$. It can be defined using the second $\alpha$-structural recursion theorem for the nominal signature $\Sigma^{\mathrm{STL}}$ much as in Example 26; instead we will regard it as a special case of the simultaneous substitution functions defined in the next section.

Once (81)–(83) are proved, then it follows that for every $e \in \Lambda(\tau)$ there is a unique $n \in N(\tau)$ with $e =_{\beta\eta} i_\tau\, n$; and, modulo some considerations about computability, deciding $\beta\eta$-conversion is reduced to the decidable relation of $\alpha$-equivalence on $\mathsf{T}(\Sigma^{\mathrm{LNF}})_{\mathsf{n}_\tau}$ by applying the $\mathit{norm}_\tau$ function. We aim to show how to use $\alpha$-structural recursion to define $\mathit{norm}_\tau$; and how to prove (81)–(83) using, among other things, $\alpha$-structural induction.

$\tau \in Ty, e \in \Lambda(\tau), \sigma \in Sub \mapsto [e]\sigma \in \Lambda(\tau)$:

$$[x]\sigma = \sigma\, x \tag{86}$$

$$[e_1\, e_2]\sigma = ([e_1]\sigma)([e_2]\sigma) \tag{87}$$

$$x \mathrel{\#} \sigma \;\Rightarrow\; [\lambda x : \tau.e]\sigma = \lambda x : \tau.[e]\sigma \tag{88}$$

$\tau \in Ty, e \in \Lambda(\tau), \rho \in Env \mapsto [\![e]\!]\rho \in D(\tau)$:

$$[\![x]\!]\rho = \rho\, x \tag{89}$$

$$[\![e_1\, e_2]\!]\rho = [\![e_1]\!]\rho\,([\![e_2]\!]\rho) \tag{90}$$

$$[\![\lambda x : \tau.e]\!]\rho = \lambda d \in D(\tau).\, [\![e]\!](\rho\{x \mapsto d\}) \tag{91}$$

Figure 5: Substitution and denotation

## 6.2 Substitution

A (simultaneous) **substitution** $\sigma$ is a function that maps atoms in $\mathbb{A}_{\mathsf{v}_\tau}$ to $\alpha$-terms in $\Lambda(\tau)$ (for any $\tau \in Ty$) and that has the property that its **domain**

$$dom(\sigma) \triangleq \{x \in \textstyle\bigcup_{\tau \in Ty} \mathbb{A}_{\mathsf{v}_\tau} \mid \sigma\, x \neq x\}$$

is a finite set. We let atom-permutations $\pi \in Perm$ act on such functions in the usual way (Section 3.2): the substitution $\pi \cdot \sigma$ maps $x \in \mathbb{A}_{\mathsf{v}_\tau}$ to $\pi \cdot (\rho(\pi^{-1}(x))$ and (therefore) has domain $dom(\pi \cdot \sigma) = \pi \cdot dom(\sigma) = \{\pi(x) \mid x \in dom(\sigma)\}$. It is not hard to see that with respect to this action, each substitution $\sigma$ is supported by the finite set of atoms $dom(\sigma) \cup \bigcup_{x \in dom(\sigma)} supp(\sigma\, x)$. Therefore the collection of substitutions forms a nominal set that we write as $Sub$.

Given $\tau \in Ty$, $e \in \Lambda(\tau)$ and $\sigma \in Sub$, we let $[e]\sigma \in \Lambda(\tau)$ denote the result of carrying out on $e$ the simultaneous, **capture-avoiding substitution** given by $\sigma$. This is specified by the recursion equations (86)–(88) in Figure 5. For each $\sigma \in Sub$, we can use the second $\alpha$-structural recursion theorem for $\Sigma^{\mathrm{STL}}$ to define the functions $([-]\sigma \in \Lambda(\tau) \to_{\mathrm{fs}} \Lambda(\tau) \mid \tau \in Ty)$ satisfying these equations, much as in Example 26, but using $supp(\sigma)$ as the common finite support $A$. (In particular, in Theorem 22 the only non-trivial freshness condition on binders, (FCB$_{Lm_{\tau,\tau'}}$), is easily verified.)

The **identity substitution** $\sigma_0 \in Sub$ maps each $x \in \mathbb{A}_{\mathsf{v}_\tau}$ to $x \in \Lambda(\tau)$. The **composition** $\sigma_1; \sigma_2$ of $\sigma_1, \sigma_2 \in Sub$ is the element of $Sub$ that maps each $x \in \mathbb{A}_{\mathsf{v}_\tau}$ to $[\sigma_1\, x]\sigma_2$. These operations satisfy:

$$[e]\sigma_0 = e \tag{92}$$

$$[e](\sigma_1; \sigma_2) = [[e]\sigma_1]\sigma_2\ . \tag{93}$$

Both properties can be proved easily by applying the second $\alpha$-structural induction principle (Theorem 23) for the nominal signature $\Sigma^{\mathrm{STL}}$. For example, to prove (93), for each $\sigma_1, \sigma_2 \in \mathit{Sub}$ we take $S_{\mathsf{t}_\tau}$ to be $\{e \in \Lambda(\tau) \mid [e](\sigma_1;\sigma_2) = [[e]\sigma_1]\sigma_2\}$, which is supported by $A = \mathit{supp}(\sigma_1, \sigma_2)$; then $(\mathrm{IH}_{\mathit{Vr}_\tau})$ and $(\mathrm{IH}_{\mathit{Ap}_{\tau,\tau'}})$ are easy to verify, using (86) and (87) respectively; for $(\mathrm{IH}_{\mathit{Lm}_{\tau,\tau'}})$, which is the statement

$$(\exists x \in \mathbb{A}_{\mathsf{v}_\tau})\; x \notin A \;\;\&\;\; (\forall e \in S_{\mathsf{t}_{\tau'}})\; \lambda x : \tau.\, e \in S_{\mathsf{t}_{\tau \dot{\to} \tau'}} \;,$$

we can choose any $x$ in the infinite set $\mathbb{A}_{\mathsf{v}_\tau} - A$ (so that $x \mathrel{\#} (\sigma_1, \sigma_2)$): if $e \in S_{\mathsf{t}_{\tau'}}$, then

$$\begin{array}{ll}
[\lambda x : \tau.\, e](\sigma_1;\sigma_2) & \\
\quad = [[\lambda x : \tau.\, e]\sigma_1]\sigma_2 & \text{by definition of } \sigma_1;\sigma_2 \\
\quad = [\lambda x : \tau.\, [e]\sigma_1]\sigma_2 & \text{by (88), since } x \mathrel{\#} \sigma_1 \\
\quad = \lambda x : \tau.\, [[e]\sigma_1]\sigma_2 & \text{by (88), since } x \mathrel{\#} \sigma_2 \\
\quad = \lambda x : \tau.\, [e](\sigma_1;\sigma_2) & \text{since } e \in S_{\mathsf{t}_{\tau'}} \\
\quad = [\lambda x : \tau.\, e](\sigma_1;\sigma_2) & \text{by (88), since } x \mathrel{\#} (\sigma_1;\sigma_2) \text{ (because } \sigma_1;\sigma_2 \text{ is} \\
& \text{supported by } \mathit{supp}(\sigma_1) \cup \mathit{supp}(\sigma_2) \subseteq A)
\end{array}$$

and hence $\lambda x : \tau.\, e \in S_{\mathsf{t}_{\tau \dot{\to} \tau'}}$.

The single-variable substitution used in the definition of $=_{\beta\eta}$ in the previous section can be defined as:

$$e_1[x := e_2] \;\triangleq\; [e_1](\sigma_0\{x \mapsto e_2\}) \tag{94}$$

where in general the **updated substitution** $\sigma\{x \mapsto e\} \in \mathit{Sub}$ maps $x \in \mathbb{A}_{\mathsf{v}_\tau}$ to $e \in \Lambda(\tau)$ and otherwise acts like $\sigma \in \mathit{Sub}$.

## 6.3 Denotation

We interpret each simple type $\tau \in \mathit{Ty}$ as a nominal set $D(\tau)$ as follows:

$$D(\iota) \;\triangleq\; N(\iota) \tag{95}$$

$$D(\tau \dot{\to} \tau') \;\triangleq\; D(\tau) \to_{\mathrm{fs}} D(\tau') \;. \tag{96}$$

An **environment** $\rho$ is a function that, for any $\tau \in \mathit{Ty}$, maps the atoms in $\mathbb{A}_{\mathsf{v}_\tau}$ to elements of $D(\tau)$. We let atom-permutations $\pi \in \mathit{Perm}$ act on such functions in the usual way (Section 3.2): the environment $\pi \cdot \rho$ maps $x \in \mathbb{A}_{\mathsf{v}_\tau}$ to $\pi \cdot (\rho(\pi^{-1}(x))$. Let $\mathit{Env}$ be the nominal set of environments that are finitely supported with respect to this action.

Given $\tau \in \mathit{Ty}$, $e \in \Lambda(\tau)$ and $\rho \in \mathit{Env}$, we wish to define the **denotation** $[\![e]\!]\rho \in D(\tau)$, satisfying equations (89)–(91) in Figure 5. It is clear from the form of these equations that we should try to define $[\![e]\!]\rho$ by $\alpha$-structural recursion for $\Sigma^{\mathrm{STL}}$ *for all $\rho$ simultaneously*, because of the use of an **updated environment** in

(91): $\rho\{x \mapsto d\}$ is by definition the function mapping $x$ to $d$ and otherwise acting like $\rho$ (and is finitely supported by $supp(\rho) \cup \{x\} \cup supp(d)$). So in Theorem 22 it seems that we should take $X_{\mathsf{t}_\tau}$ to be $Env \to_{\mathrm{fs}} D(\tau)$ and use the functions

$$f_{Vr_\tau} \triangleq \lambda x \in \mathbb{A}_{\mathsf{v}_\tau}.\lambda\rho \in Env.\, \rho\, x \tag{97}$$

$$f_{Ap_{\tau,\tau'}} \triangleq \lambda(\xi,\xi') \in X_{\mathsf{t}_{\tau \dot\to \tau'}} \times X_{\mathsf{t}_\tau}.\lambda\rho \in Env.\, \xi\,\rho\,(\xi'\rho) \tag{98}$$

$$f_{Lm_{\tau,\tau'}} \triangleq \lambda(x,\xi') \in \mathbb{A}_{\mathsf{v}_\tau} \times X_{\mathsf{t}_{\tau'}}.\lambda\rho \in Env.\lambda d \in D(\tau).\, \xi'(\rho\{x \mapsto d\})\ . \tag{99}$$

These functions are all supported by $A = \emptyset$ and the only non-trivial freshness condition on binders is (FCB$_{Lm_{\tau,\tau'}}$), which in view of Theorem 14 (the "some/any" theorem) is the requirement that for all $x \in \mathbb{A}_{\mathsf{v}_\tau}$ and $\xi \in Env \to_{\mathrm{fs}} D(\tau')$

$$x \mathrel{\#} \lambda\rho \in Env.\lambda d \in D(\tau).\, \xi(\rho\{x \mapsto d\})\ . \tag{100}$$

If we could prove that, then the theorem gives us functions $\hat{f}_{\mathsf{t}_\tau} \in \Lambda(\tau) \to_{\mathrm{fs}} (Env \to_{\mathrm{fs}} D(\tau))$ satisfying (46)—from which it follows that $[\![e]\!]\rho \triangleq \hat{f}_{\mathsf{t}_\tau}\, e\, \rho$ satisfies (89)–(91). The problem is that (100) is not true for all elements $\xi$ of $Env \to_{\mathrm{fs}} D(\tau')$! We have to strengthen the "recursion hypothesis" by suitably restricting the class of functions $\xi$ that we consider.

This is the first time in this paper we have encountered a really non-trivial "freshness condition on binders". Equations (89)–(91) are typical of many such definitions in denotational semantics; but why is it the case that the right-hand side of equation (91) is independent of the choice of bound variable $x$ on the left-hand side? Compared with the similar question for equation (88) a few lines above it, there seems no quick answer to this question. However, the freshness of $x$ for $\lambda\rho \in Env.\lambda d \in D(\tau).\, [\![e]\!](\rho\{x \mapsto d\})$ does follow from two expected properties of denotations:

1. The denotation of $e$ with respect to an environment $\rho$ only depends on the value of $\rho$ at the free variables of $e$.
2. The denotation of a permuted version $\pi\cdot e$ of $e$ with respect to an environment $\rho$ is the denotation of $e$ with respect to the composition $\rho \circ \pi$.

It is possible to take account of these facts in advance when applying Theorem 22 to construct denotations.[11] To do so we cut down to the following nominal subset of $Env \to_{\mathrm{fs}} D(\tau)$:

$$X_{\mathsf{t}_\tau} \triangleq \{\xi \in Env \to_{\mathrm{fs}} D(\tau) \mid \Phi_1(\xi)\ \ \&\ \ \Phi_2(\xi)\} \tag{101}$$

where

$$\begin{aligned}\Phi_1(\xi) \triangleq\ & (\exists A \in P_{\mathrm{fin}}(\mathbb{A}))(\forall \tau \in Ty, x \in \mathbb{A}_{\mathsf{v}_\tau}, d \in D(\tau), \rho \in Env)\\ & \qquad\qquad x \notin A \Rightarrow \xi(\rho\{x \mapsto d\}) = \xi\,\rho\end{aligned} \tag{102}$$

$$\Phi_2(\xi) \triangleq (\forall \pi \in Perm, \rho \in Env)\ (\pi\cdot\xi)\,\rho = \xi(\rho\circ\pi)\}\ . \tag{103}$$

[11] We are in effect carrying out a simultaneous inductive-recursive definition: see [7].

The slightly elaborate form of $\Phi_1$ compared with property 1 above is needed to show that the functions defined in (97)–(99) satisfy

$$\begin{aligned}
&\Phi_1(f_{Vr_\tau}\, x)\\
&\Phi_1(\xi)\ \&\ \Phi_1(\xi') \ \Rightarrow\ \Phi_1(f_{Ap_{\tau,\tau'}}(\xi,\xi'))\\
&\Phi_1(\xi') \ \Rightarrow\ \Phi_1(f_{Lm_{\tau,\tau'}}(x,\xi'))\ .
\end{aligned}$$

Similar properties hold for $\Phi_2$. Therefore $f_{Vr_\tau} \in \mathbb{A}_{\mathsf{v}_\tau} \to_{\mathrm{fs}} X_{\mathsf{t}_\tau}$, $f_{Ap_{\tau,\tau'}} \in X_{\mathsf{t}_{\tau \dot{\to} \tau'}} \times X_{\mathsf{t}_\tau} \to_{\mathrm{fs}} X_{\mathsf{t}_{\tau'}}$ and $f_{Lm_{\tau,\tau'}} \in \mathbb{A}_{\mathsf{v}_\tau} \times X_{\mathsf{t}_{\tau'}} \to_{\mathrm{fs}} X_{\mathsf{t}_{\tau \dot{\to} \tau'}}$. Furthermore, it is now the case that if $x \in \mathbb{A}_{\mathsf{v}_\tau}$ and $\xi' \in X_{\mathsf{t}_{\tau'}}$, then (100) holds. To see this, first note that since $\Phi_1(\xi')$ holds, there is a finite set of atoms $A$ such that

$$(\forall \tau \in Ty, x' \in \mathbb{A}_{\mathsf{v}_\tau}, d' \in D(\tau), \rho' \in Env)\ x' \notin A\ \Rightarrow\\ \xi'(\rho'\{x' \mapsto d'\}) = \xi'\rho'\ . \quad (104)$$

Choose any $x'$ in the infinite set $\mathbb{A}_{\mathsf{v}_\tau} - supp(x,\xi',A)$. Hence $x' \mathrel{\#} \lambda\rho \in Env.\lambda d \in D(\tau).\, \xi'(\rho\{x \mapsto d\})$; and so applying the transposition $(x\ x')$ to this we get

$$\begin{aligned}
x &= (x\ x')\cdot x'\\
&\mathrel{\#}\ (x\ x')\cdot \lambda\rho \in Env.\lambda d \in D(\tau).\xi'(\rho\{x \mapsto d\})\\
&= \lambda\rho \in Env.\lambda d \in D(\tau).((x\ x')\cdot \xi')(\rho\{x' \mapsto d\})\\
&= \lambda\rho \in Env.\lambda d \in D(\tau).\xi'(\rho\{x' \mapsto d\} \circ (x\ x')) && \text{because } \Phi_2(\xi') \text{ holds}\\
&= \lambda\rho \in Env.\lambda d \in D(\tau).\xi'(\rho\{x \mapsto d\}\{x' \mapsto \rho\, x\}) && \text{because } x' \neq x\\
&= \lambda\rho \in Env.\lambda d \in D(\tau).\xi'(\rho\{x \mapsto d\}) && \text{by (104), since } x' \notin A.
\end{aligned}$$

So (100) does indeed hold. Therefore we can apply Theorem 22 to these nominal sets and functions to obtain $\hat{f}_{\mathsf{t}_\tau} \in X_{\mathsf{t}_\tau}$ satisfying (46). Defining $[\![e]\!]\rho \triangleq \hat{f}_{\mathsf{t}_\tau}\, e\, \rho$, we get the required properties (89)–(91).

Denotations respect $\beta\eta$-conversion:

$$e_1 =_{\beta\eta} e_2 \ \Rightarrow\ [\![e_1]\!] = [\![e_2]\!]\ . \quad (105)$$

To see this it suffices to show that $[\![-]\!] = [\![-]\!]$ is a congruence relation equating $\beta$-convertible (84) and $\eta$-convertible (85) terms. Congruence is immediate from the defining properties (89)–(91) of $[\![-]\!]$. To see that $[\![-]\!]$ respects $\beta$-conversion one has to show

$$[\![e'[x := e]]\!]\rho = [\![e']\!](\rho\{x \mapsto [\![e]\!]\rho\}) \quad (106)$$

and for $\eta$-conversion one has to show

$$x \notin fv(e) \ \Rightarrow\ [\![e]\!](\rho\{x \mapsto d\}) = [\![e]\!]\rho\ . \quad (107)$$

These properties can be proved using the second $\alpha$-structural induction principle (Theorem 23) for $\Sigma^{\mathrm{STL}}$, with the second one used in the proof of the first.[12] For

[12] It might seem that (107) is a consequence of the property (102) that we built in to the construction of $[\![-]\!]$; but unfortunately that property only tells us that $[\![e]\!](\rho\{x \mapsto d\})$ and $[\![e]\!]\rho$ are equal when $x$ avoids some finite set of atoms, rather than the particular finite set $supp(e)$.

(107) one can use the subsets

$$S_{\mathsf{t}_\tau} \triangleq \{e \in \Lambda(\tau) \mid (\forall x \in \mathbb{A}_{\mathsf{v}_\tau}, \rho \in \mathit{Env}, d \in D(\tau))\; x \# e \;\Rightarrow \\ \llbracket e \rrbracket(\rho\{x \mapsto d\}) = \llbracket e \rrbracket\rho\}$$

which are all supported by the empty set. Properties (89)–(91) of $\llbracket - \rrbracket$ allow one to show that these subsets satisfy the induction hypotheses (IH$_{Vr_\tau}$), (IH$_{Ap_{\tau,\tau'}}$) and (IH$_{Lm_{\tau,ty'}}$); hence by Theorem 23 each $S_{\mathsf{t}_\tau}$ is the whole of $\Lambda(\tau)$ and (107) holds.

For (106), for each $x \in \mathbb{A}_{\mathsf{v}_\tau}$ and $e \in \Lambda(\tau)$ we apply Theorem 23 to the subsets

$$S'_{\mathsf{t}_\tau} \triangleq \{e' \in \Lambda(\tau) \mid (\forall \rho \in \mathit{Env})\; \llbracket e'[x := e] \rrbracket\rho = \llbracket e' \rrbracket(\rho\{x \mapsto \llbracket e \rrbracket\rho\})\}$$

which are all supported by $A \triangleq \{x\} \cup \mathit{supp}(e)$. To conclude that $S'_{\mathsf{t}_\tau} = \Lambda(\tau)$ we have to prove the these subsets $S'_{\mathsf{t}_\tau}$ satisfy the induction hypotheses (IH$_{Vr_\tau}$), (IH$_{Ap_{\tau,\tau'}}$) and (IH$_{Lm_{\tau,ty'}}$). The first two follow easily from (89) and (90). For (IH$_{Lm_{\tau,ty'}}$) we have to show $(\exists x' \in \mathbb{A}_{\mathsf{v}_\tau})\; x' \# (x, e) \;\;\&\;\; (\forall e' \in S_{\mathsf{t}_\tau})\; \lambda x' : \tau.\, e' \in S_{\mathsf{t}_\tau}$; but choosing any $x'$ in the infinite set $\mathbb{A}_{\mathsf{v}_\tau} - \mathit{supp}(x, e)$, for each $e' \in S'_{\mathsf{t}_\tau}$ and $\rho \in \mathit{Env}$ we have:

$$\begin{array}{ll}
\llbracket (\lambda x' : \tau.\, e')[x := e] \rrbracket\rho & \\
= \llbracket \lambda x' : \tau.\, (e'[x := e]) \rrbracket\rho & \text{since } x' \notin \mathit{fv}(x, e) \\
= \lambda d \in D(\tau).\, \llbracket e'[x := e] \rrbracket(\rho\{x' \mapsto d\}) & \text{by (91)} \\
= \lambda d \in D(\tau).\, \llbracket e' \rrbracket(\rho\{x' \mapsto d\}\{x \mapsto \llbracket e \rrbracket(\rho\{x' \mapsto d\})\}) & \text{since } e' \in S_{\mathsf{t}_\tau} \\
= \lambda d \in D(\tau).\, \llbracket e' \rrbracket(\rho\{x' \mapsto d\}\{x \mapsto \llbracket e \rrbracket\rho\}) & \text{by (107), since } x' \notin \mathit{fv}(e) \\
= \lambda d \in D(\tau).\, \llbracket e' \rrbracket(\rho\{x \mapsto \llbracket e \rrbracket\rho\}\{x' \mapsto d\} & \text{since } x' \neq x \\
= \llbracket \lambda x' : \tau.\, e' \rrbracket(\rho\{x \mapsto \llbracket e \rrbracket\rho\}) & \text{by (91)}
\end{array}$$

so that $\lambda x' : \tau.\, e' \in S_{\mathsf{t}_\tau}$, as required.

## 6.4 Reification, reflection and normalisation

The **reification function** $\downarrow_\tau \in D(\tau) \to_{\text{fs}} N(\tau)$ turns elements of the denotational model into $\beta\eta$-long normal forms, whereas the **reflection function** $\uparrow_\tau \in U(\tau) \to_{\text{fs}} D(\tau)$ turns neutral terms into denotational elements. Both functions are defined simultaneously by ordinary structural recursion for simple type symbols $\tau \in \mathit{Ty}$ in Figure 6, where for clarity we have written $\alpha$-terms over the signature $\Sigma^{\text{LNF}}$ without the conventions introduced by Notation 28. The interesting part of the definition in this figure is clause (109), where we make use of the *fresh* construct from Theorem 16. To do so we have to check that the conditions of that theorem are satisfied; but in this case that is easy: given $f \in D(\tau \dot{\to} \tau')$, choosing any atom $x$ in the infinite set $\mathbb{A}_{\mathsf{v}_\tau} - \mathit{supp}(\downarrow_{\tau'}, \uparrow_\tau, f)$,[13] then the element

$$h \triangleq \lambda x \in \mathbb{A}_{\mathsf{v}_\tau}.\, L_{\tau,\tau'}\, x.\, \downarrow_{\tau'}(f(\uparrow_\tau(V_\tau\, x)))$$

[13]In fact $\mathit{supp}(\downarrow_{\tau'}, \uparrow_\tau, f) = \mathit{supp}(f)$ because the reification and reflection functions turn out to have empty support.

$\tau \in Ty, d \in D(\tau) \mapsto \downarrow_\tau(d) \in N(\tau)$:

$$\downarrow_\iota(n) \triangleq n \tag{108}$$

$$\downarrow_{\tau \dot{\to} \tau'}(f) \triangleq \mathit{fresh}(\lambda x \in \mathbb{A}_{\mathsf{v}_\tau}.\, L_{\tau,\tau'}\, x.\, \downarrow_{\tau'}(f(\uparrow_\tau(V_\tau\, x)))) \tag{109}$$

$\tau \in Ty, u \in U(\tau) \mapsto \uparrow_\tau(u) \in D(\tau)$:

$$\uparrow_\iota(u) \triangleq I\, u \tag{110}$$

$$\uparrow_{\tau \dot{\to} \tau'}(u) \triangleq \lambda d \in D(\tau).\, \uparrow_{\tau'}(A_{\tau,\tau'}(u, \downarrow_\tau(d))) \tag{111}$$

Figure 6: Reification ($\downarrow_\tau$) and reflection ($\uparrow_\tau$)

of $\mathbb{A}_{\mathsf{v}_\tau} \to_{\mathrm{fs}} N(\tau')$ satisfies $x \,\#\, h$ (by choice of $x$) and $x \,\#\, h(x)$ (because $x \notin \mathit{fv}(L_{\tau,\tau'}\, x.\, n)$ for any $n \in N(\tau')$); so we can form $\mathit{fresh}(h)$ as in the theorem.

**Definition 29.** The **initial environment** $\rho_0$ maps each $x \in \mathbb{A}_{\mathsf{v}_\tau}$ to $\uparrow_\tau(V_\tau\, x) \in D(\tau)$, for all $\tau \in Ty$. This has empty support (because the $\uparrow_\tau$ functions do) and hence in particular it is an element of the nominal set *Env* of finitely supported environments. Using it, we define the **normalisation function** $\mathit{norm}_\tau \in \Lambda(\tau) \to_{\mathrm{fs}} N(\tau)$ by:

$$\mathit{norm}_\tau(e) \triangleq \downarrow_\tau([\![e]\!]\rho_0)\,. \tag{112}$$

Recall that we wish to show that $\mathit{norm}_\tau$ has the properties (81)–(83). Property (81) follows immediately from (105). Property (82) is the first half of the following result.

**Lemma 30.** *For all $\tau \in Ty$, $n \in N(\tau)$ and $u \in U(\tau)$*

$$\downarrow_\tau([\![i_\tau\, n]\!]\rho_0) = n \tag{113}$$

$$[\![j_\tau\, u]\!]\rho_0 = \uparrow_\tau(u)\,. \tag{114}$$

*Proof.* These properties can be proved by using the second $\alpha$-structural induction principle (Theorem 23) for $\Sigma^{\mathrm{LNF}}$ to show that the subsets

$$S_{\mathsf{n}_\tau} \triangleq \{n \in N(\tau) \mid \downarrow_\tau([\![i_\tau\, n]\!]\rho_0) = n\}$$

$$S_{\mathsf{u}_\tau} \triangleq \{u \in U(\tau) \mid [\![j_\tau\, u]\!]\rho_0 = \uparrow_\tau(u)\}$$

are equal to $N(\tau)$ and $U(\tau)$ respectively (for all $\tau \in Ty$). Since these subsets are

$\sim_\tau \subseteq D(\tau) \times \Lambda(\tau)$:

$$n \sim_\iota e \triangleq (i_\iota\, n =_{\beta\eta} e)$$
$$f \sim_{\tau \dot\to \tau'} e \triangleq (\forall d_1 \in D(\tau), e_1 \in \Lambda(\tau))\; d_1 \sim_\tau e_1 \Rightarrow f\, d \sim_{\tau'} Ap_{\tau,\tau'}(e, e_1)$$

$\sim \subseteq Env \times Sub$:

$$\rho \sim \sigma \triangleq (\forall \tau \in Ty, x \in \mathbb{A}_{\mathsf{v}_\tau})\; \rho\, x \sim_\tau \sigma\, x$$

Figure 7: Logical relation

supported by the empty set of atoms one can take $A = \emptyset$ in the theorem and prove

$$(\forall x \in \mathbb{A}_{\mathsf{v}_\tau})\; V_\tau\, x \in S_{\mathsf{u}_\tau} \qquad (\mathrm{IH}_{V_\tau})$$
$$(\forall u \in S_{\mathsf{u}_{\tau \dot\to \tau'}}, n \in S_{\mathsf{n}_\tau})\; A_{\tau,\tau'}(u, n) \in S_{\mathsf{u}_{\tau'}} \qquad (\mathrm{IH}_{A_{\tau,\tau'}})$$
$$(\exists x \in \mathbb{A}_{\mathsf{v}_\tau})(\forall n \in S_{\mathsf{n}_{\tau'}})\; L_{\tau,\tau'}\, x.\, n \in S_{\mathsf{n}_{\tau \dot\to \tau'}} \qquad (\mathrm{IH}_{L_{\tau,\tau'}})$$
$$(\forall u \in S_{\mathsf{u}_\iota})\; I\, u \in S_{\mathsf{u}_\iota}\;. \qquad (\mathrm{IH}_I)$$

These nearly all follow directly from the definitions of $i_\tau$, $j_\tau$, $\downarrow_\tau$, $\uparrow_\tau$, $[\![-]\!]$ and $\rho_0$. The only tricky case is for (IH$_{L_{\tau,\tau'}}$), because of the use of *fresh* in $\downarrow_{\tau \dot\to \tau'}$. Picking any atom $x$ in $\mathbb{A}_{\mathsf{v}_\tau}$, suppose $n \in S_{\mathsf{n}_{\tau'}}$. We wish to prove that $L_{\tau,\tau'}\, x.\, n \in S_{\mathsf{n}_{\tau \dot\to \tau'}}$. Note that $x \mathrel{\#} L_{\tau,\tau'}\, x.\, n$; so since $[\![-]\!]$, $i_{\tau \dot\to \tau'}$ and $\rho_0$ have empty support, it is the case that $x \mathrel{\#} [\![i_{\tau \dot\to \tau'}(L_{\tau,\tau'}\, x.\, n)]\!]\rho_0 = [\![\lambda x : \tau.\, i_{\tau'}\, n]\!]\rho_0 = f$, where $f \triangleq \lambda d \in D(\tau).\, [\![i_{\tau'}\, n]\!](\rho_0\{x \mapsto d\})$. Hence

$$x \mathrel{\#} (\lambda x' \in \mathbb{A}_{\mathsf{v}_\tau}.\, L_{\tau,\tau'}\, x'.\, \downarrow_{\tau'}(f(\uparrow_\tau(V_\tau\, x'))))$$

and therefore by definition of *fresh* in Theorem 16

$$\begin{aligned}
&\downarrow_{\tau \dot\to \tau'}([\![i_{\tau \dot\to \tau'}(L_{\tau,\tau'}\, x.\, n)]\!]\rho_0)\\
&= \mathit{fresh}(\lambda x' \in \mathbb{A}_{\mathsf{v}_\tau}.\, L_{\tau,\tau'}\, x'.\, \downarrow_{\tau'}(f(\uparrow_\tau(V_\tau\, x'))))\\
&= L_{\tau,\tau'}\, x.\, \downarrow_{\tau'}(f(\uparrow_\tau(V_\tau\, x)))\\
&= L_{\tau,\tau'}\, x.\, \downarrow_{\tau'}([\![i_{\tau'}\, n]\!](\rho_0\{x \mapsto \uparrow_\tau(V_\tau\, x)\}))\\
&= L_{\tau,\tau'}\, x.\, \downarrow_{\tau'}([\![i_{\tau'}\, n]\!]\rho_0) && \text{by definition of } \rho_0\\
&= L_{\tau,\tau'}\, x.\, n && \text{since } n \in S_{\mathsf{n}_{\tau'}}
\end{aligned}$$

so that $L_{\tau,\tau'}\, x.\, n$ is indeed an element of $S_{\mathsf{n}_{\tau \dot\to \tau'}}$. □

It just remains to prove that $norm_\tau$ has the property (83). For this we use a **logical relation** $\sim_\tau \subseteq D(\tau) \times \Lambda(\tau)$ between elements of the denotational model and terms. It is defined by ordinary structural recursion for simple type symbols $\tau \in \mathit{Ty}$ in Figure 7, which also extends the relation to one between (finitely supported) environments and substitutions.

**Lemma 31** (**fundamental property** of $\sim$)**.** *For all $\tau \in \mathit{Ty}$, $e \in \Lambda(\tau)$, $\rho \in \mathit{Env}$ and $\sigma \in \mathit{Sub}$*

$$\rho \sim \sigma \;\Rightarrow\; [\![e]\!]\rho \sim_\tau [e]\sigma\;. \tag{115}$$

*Proof.* This can be proved by applying the second $\alpha$-structural induction theorem for the nominal signature $\Sigma^{\mathrm{STL}}$ to show that the subsets

$$S_\tau \;\triangleq\; \{e \in \Lambda(\tau) \mid (\forall \rho \in \mathit{Env}, \sigma \in \mathit{Sub})\; \rho \sim \sigma \;\Rightarrow\; [\![e]\!]\rho \sim_\tau [e]\sigma\}$$

are equal to $\Lambda(\tau)$, for all $\tau \in \mathit{Ty}$. Proving the induction hypotheses ($\mathrm{IH}_{\mathit{Vr}_\tau}$) and ($\mathrm{IH}_{\mathit{Ap}_{\tau,\tau'}}$) is straightforward; and for ($\mathrm{IH}_{\mathit{Lm}_{\tau,\tau'}}$) one first proves

$$d \sim_\tau e \;\;\&\;\; e =_{\beta\eta} e' \;\Rightarrow\; d \sim_\tau e' \tag{116}$$

$$([\lambda x.\, e]\sigma)\, e' =_{\beta\eta} [e](\sigma\{x \mapsto e'\}) \tag{117}$$

where $\sigma\{x \mapsto e'\}$ indicates an updated substitution mapping $x$ to $e'$ and otherwise acting like $\sigma$. □

One can prove by ordinary structural induction for types $\tau \in \mathit{Ty}$ that

$$(\forall \tau \in \mathit{Ty}, u \in U(\tau))\; \uparrow_\tau(u) \sim_\tau j_\tau\, u \tag{118}$$

$$(\forall \tau \in \mathit{Ty}, d \in D(\tau), e \in \Lambda(\tau))\; d \sim_\tau e \;\Rightarrow\; i_\tau(\downarrow_\tau(d)) =_{\beta\eta} e\;. \tag{119}$$

(Both properties are proved simultaneously, and one needs to make use of (116).) Because of (118), the identity substitution $\sigma_0 \in \mathit{Sub}$ satisfies $\rho_0 \sim \sigma_0$ . So by Lemma 31 and (92) we have $[\![e]\!]\rho_0 \sim_\tau [e]\sigma_0 = e$, for all $e \in \Lambda(\tau)$. Thus (119) gives $i_\tau(\downarrow_\tau([\![e]\!]\rho_0)) =_{\beta\eta} e$, i.e. $i_\tau(\mathit{norm}_\tau(e)) =_{\beta\eta} e$, as required for (83).

## 7 Assessment

### 7.1 Mathematical perspective

The results of this paper are directly inspired by my joint work with Gabbay on "FM-set" theory [13] and by his PhD thesis [11]; in particular those works contain structural recursion and induction principles for an inductively defined FM-set isomorphic to $\lambda$-terms modulo $\alpha$-equivalence. Here I have taken an approach that is both a bit more general and more concrete: more general, because the particular signature for $\lambda$-terms has been replaced by an arbitrary nominal signature (a notion which comes from joint work with Urban and Gabbay [28] and is developed further in Cheney's thesis [4]); 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 [4, Chapter 3] for a more leisurely and generalised account of the theory of nominal sets. Secondly, the recursion and induction principles developed here refer directly to $\alpha$-terms, i.e. standard $\alpha$-equivalence classes of abstract syntax trees, rather than using an intial algebra that is merely isomorphic to the set of $\alpha$-terms; see Remark 24. This is also the approach taken by Norrish [18], building on Gordon and Melham's five axioms for $\alpha$-equivalence [14]; and also by Urban and Tasson [29]. Norrish's recursion principle [18, 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 having to specify how binders with only fresh names are mapped by the function. Along with Urban and Tasson [29], I prefer to develop the theory of nominal sets in its own right and then give a simple-looking[14] 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 Theorem 16, independently of the recursion principle. We used Theorem 16 in the reduction of Theorem 17 to Theorem 3, in the reduction of "varying parameters" to "no parameters" (Example 27) and in the definition of the reification functions in the extended example in Section 6; another good example of its use occurs (implicitly) in the denotational semantics of FreshML's `fresh` expression [26, Section 3].

How easy is it to apply these principles of $\alpha$-structural recursion and induction? Just as for the work of Gordon-Melham, Norrish and Urban-Tasson [14, 18, 29], 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 $\alpha$-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 (Theorem 11) that the finite support property is conserved by all the usual constructs of higher-order logic except for uses of the axiom of choice. Thus if some language of interest has been specified as the $\alpha$-terms for a particular nominal signature and one wishes to define a function on those $\alpha$-terms specified by an instance of the recursion scheme (46) in Theorem 22 (for suitable functions $f_K$), then there are three tasks involved in applying the theorem to this data:

(I) Show that the sets $X_{\mathsf{s}}$ that one is mapping into have the structure of nominal sets.

(II) Show that the functions $f_K$ are all supported by a single finite set of atoms $A$.

(III) Show that each function $f_K$ satisfies the "freshness condition on binders" (FCB$_K$).

[14]Once one gets used to the distinctive concepts of nominal sets, I believe that principles such as Theorems 17, 22 and 25 *are* quite simple.

Task (I) is usually carried out by showing how the $X_{\mathrm{s}}$ are built up from some standard nominal sets (such as those in Example 7) using the constructions described in Sections 3.2 and 3.3. Task (II) might seem quite difficult, but in fact the Finite Support Principle (Theorem 11) usually reduces it to seeing how the $f_K$ are constructed within higher-order logic. So really the main difficulty is task (III). In some cases, such as the capture-avoiding substitution function in Example 26, (FCB$_K$) is very easily checked. In other cases, such as in the definition of the denotation functions $[\![-]\!]$ in the extended example of Section 6, one has to work hard to verify the freshness condition on binders.

Applying the $\alpha$-structural induction principles is somewhat easier. For example, for the second $\alpha$-structural induction principle (Theorem 23) one still has the analogues of the easy tasks (I) and (II); and then one just has to verify the induction hypothesis (IH$_K$) for each constructor $K$, which is in fact a more restricted property than the corresponding induction hypothesis in an ordinary structural induction for terms rather than $\alpha$-terms.

## 7.2 Automated theorem proving perspective

Based on my experience with other formalisms, I claim that the use of permutations and finitely supported objects advocated here is a simple, effective and yet rigorous way of dealing with binders and $\alpha$-equivalence in “paper-and-pencil” proofs in programming language semantics. But how easy is it to provide computer support for reasoning with $\alpha$-structural recursion and induction? Of the three types (I–III) of task involved in applying these principles in any particular case that were mentioned above, task (III) will require human-intervention; but in view of Theorem 11, there is the possibility of automating tasks (I) and (II). 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 [12]. 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 HOL4 and Isabelle/HOL. To what extent tasks (I) and (II) can be automated within these “legacy” mechanised logics remains to be seen. The work of Norrish [18] provides a starting point within the HOL4 system; and Urban and Tasson [29] have already developed a theory equivalent to nominal sets within Isabelle/HOL up to and including an induction principle (but not yet a recursion principle) for the particular nominal signature for $\lambda$-terms.[15] It is to be hoped that these works will lead to an augmentation of the treatment of data types withing HOL4 or Isabelle/HOL, allowing the user to declare a nominal signature and then have the principles of $\alpha$-structural recursion and induction for that signature proved and ready to be applied.[16]

[15]Their proof of validity of the induction principle follows a different route from the one used here to prove the $\alpha$-structural recursion and induction principles.

[16]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.

**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 which is a revised and much extended version of [21].

## A Proof of Second $\alpha$-Structural Induction Theorem

Given a nominal signature $\Sigma$, we prove the second $\alpha$-structural induction principle (Theorem 23) as a corollary of the first one, Theorem 21. To do so we must first develop some properties of the operation $\bar{a} \in \mathbb{A}^{\sigma}, \bar{e} \in \mathsf{T}_{\alpha}(\Sigma)^{|\sigma|} \mapsto \bar{a}.\,\bar{e} \in \mathsf{T}_{\alpha}(\Sigma)_{\sigma}$ and the relation $\natural_{\sigma} \subseteq \mathbb{A}^{\sigma} \times \mathsf{T}_{\alpha}(\Sigma)^{|\sigma|}$ defined in Figure 2.

**Lemma 32.** *If $\bar{a} \in \mathbb{A}^{\sigma}$, $\bar{e} \in \mathsf{T}_{\alpha}(\Sigma)^{|\sigma|}$ and $\bar{a} \natural_{\sigma} \bar{e}$, then $supp(\bar{a}.\,\bar{e}) = supp(\bar{e}) - supp(\bar{a})$.*

*Proof.* This can be proved by ordinary structural recursion for arities. The induction step for pair arities uses the fact (20) that the support of a pair is the union of the supports of its two components. The induction step for atom-binding arities uses the fact, noted in Example 7(iii), that the support of an $\alpha$-term is its finite set of free atoms (so that in particular $supp(a.\,e) = supp(e) - \{a\}$). $\square$

**Lemma 33.** *Let $(S_{\mathsf{s}} \in P_{\mathrm{fs}}(\mathsf{T}_{\alpha}(\Sigma)_{\mathsf{s}}) \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ be a family of finitely supported subsets of $\alpha$-terms indexed by the data-sorts of $\Sigma$. Extend this to arity-indexed families*

$$(S_{\sigma} \in P_{\mathrm{fs}}(\mathsf{T}_{\alpha}(\Sigma)_{\sigma}) \mid \sigma \in Ar(\Sigma))$$
$$(S^{|\sigma|} \in P_{\mathrm{fs}}(\mathsf{T}_{\alpha}(\Sigma)^{|\sigma|}) \mid \sigma \in Ar(\Sigma))$$

*as in Figure 3. Then for any finite set $A$ of atoms, each element of $S_{\sigma}$ is of the form $\bar{a}.\,\bar{e}$ for some $\bar{a} \in \mathbb{A}^{\sigma}$ and $\bar{e} \in S^{|\sigma|}$ satisfying $\bar{a} \mathrel{\#} A$ and $\bar{a} \natural_{\sigma} \bar{e}$.*

*Proof.* This can be proved by ordinary structural induction for arities, for all $A$ simultaneously. The induction steps when $\sigma = \mathsf{a} \in \Sigma_{\mathrm{A}}$, $\sigma = \mathsf{s} \in \Sigma_{\mathrm{D}}$ and $\sigma = 1$ are trivial.

*Case* $\sigma = \sigma_1 * \sigma_2$. By definition of $S_{\sigma}$, each $e \in S_{\sigma}$ is of the form $e = (e_1, e_2)$ with $e_i \in S_{\sigma_i}$ for $i = 1, 2$. Given $A$, by induction hypothesis for $\sigma_1$ applied to the finite set of atoms $A \cup supp(e_2)$, we can find $\bar{a}_1 \in \mathbb{A}^{\sigma_1}$ and $\bar{e}_1 \in S^{|\sigma_1|}$ with $e_1 = \bar{a}_1.\,\bar{e}_1$, $\bar{a}_1 \mathrel{\#} (A, e_2)$ and $\bar{a}_1 \natural_{\sigma_1} \bar{e}_1$. Then by induction hypothesis for $\sigma_2$, we can find $\bar{a}_2 \in \mathbb{A}^{\sigma_2}$ and $\bar{e}_2 \in S^{|\sigma_2|}$ with $e_2 = \bar{a}_2.\,\bar{e}_2$, $\bar{a}_2 \mathrel{\#} (A, \bar{a}_1, \bar{e}_1)$ and $\bar{a}_2 \natural_{\sigma_2} \bar{e}_2$. Since $\bar{a}_1 \mathrel{\#} \bar{a}_2$, we have $(\bar{a}_1, \bar{a}_2) \in \mathbb{A}^{\sigma_1} \otimes \mathbb{A}^{\sigma_2} = \mathbb{A}^{\sigma}$. So $e = (e_1, e_2) = (\bar{a}_1, \bar{a}_2).\,(\bar{e}_1, \bar{e}_2)$ with $(\bar{a}_1, \bar{a}_2) \mathrel{\#} A$ and $(\bar{e}_1, \bar{e}_2) \in S^{|\sigma_1|} \times S^{|\sigma_2|} = S^{|\sigma|}$. So it just remains to show $(\bar{a}_1, \bar{a}_2) \natural_{\sigma} (\bar{e}_1, \bar{e}_2)$. For this, since $\bar{a}_i \natural_{\sigma_i} \bar{e}_i$ ($i = 1, 2$) and $\bar{a}_2 \mathrel{\#} \bar{e}_1$ hold by construction, we just need to prove that $\bar{a}_1 \mathrel{\#} \bar{e}_2$. We chose $\bar{a}_1$ to have its support disjoint from $supp(e_2)$, which by Lemma 32 is $supp(\bar{e}_2) - supp(\bar{a}_2)$; since $supp(\bar{a}_1)$ is also disjoint from $supp(\bar{a}_2)$, it follows that $supp(\bar{a}_1) \cap supp(\bar{e}_2) = \emptyset$, i.e. $\bar{a}_1 \mathrel{\#} \bar{e}_2$, as required.

*Case* $\sigma = «\mathsf{a}»\sigma_1$. By definition of $S_\sigma$, each $e \in S_\sigma$ is of the form $e = a.\, e_1$ with $a \in \mathbb{A}_{\mathsf{a}} - \mathit{supp}(S_{\sigma_1})$ and $e_1 \in S_{\sigma_1}$. Given $A$, choosing any atom $a'$ in the infinite set $\mathbb{A}_{\mathsf{a}} - \mathit{supp}(A, a, e_1)$, it follows from the characterisation of $\alpha$-equivalence in (28) that $e = a'.\, e'_1$ where $e'_1 \triangleq (a\ a') \cdot e_1 \in (a\ a') \cdot S_{\sigma_1} = S_{\sigma_1}$, since $a, a' \# S_{\sigma_1}$. Then by induction hypothesis for $\sigma_1$ applied to the finite set of atoms $A \cup \{a'\}$, we can find $\bar{a}_1 \in \mathbb{A}^{\sigma_1}$ and $\bar{e}_1 \in S^{|\sigma_1|}$ with $e'_1 = \bar{a}_1.\, \bar{e}_1$, $\bar{a}_1 \# (A, a')$ and $\bar{a}_1 \sharp_{\sigma_1} \bar{e}_1$. since $a' \# \bar{a}_1$, we have $(a', \bar{a}_1) \in \mathbb{A}_{\mathsf{a}} \otimes \mathbb{A}^{\sigma_1} = \mathbb{A}^\sigma$. So $e = a'.\, e'_1 = (a', \bar{a}_1).\, \bar{e}_1$ with $(a', \bar{a}_1) \# A$, $\bar{e}_1 \in S^{|\sigma_1|} = S^{|\sigma|}$ and $(a', \bar{a}_1) \sharp_\sigma \bar{e}_1$. □

To prove Theorem 23 we also need to see that its induction hypotheses ($\mathrm{IH}_K$) are equivalent to ones that universally rather than existentially quantify over suitably fresh nested tuples $\bar{a}$ of atoms. To do so we make use of the following generalisation of permutations that transpose a pair of atoms.

**Definition 34 (vector-transpositions).** Let $\Sigma$ be a nominal signature and $\sigma \in Ar(\Sigma)$ an arity over the signature. Let the nominal set $\mathbb{A}^\sigma$ of nested tuples of distinct atoms be defined as in Figure 2. For each pair of elements $\bar{a}, \bar{a}' \in \mathbb{A}^\sigma$ with $\bar{a} \# \bar{a}'$, let $\tau_{\bar{a},\bar{a}'} \in \mathit{Perm}$ be the atom-permutation that transposes the atoms at corresponding positions in the nested tuples $\bar{a}$ and $\bar{a}'$. More formally, $\tau_{\bar{a},\bar{a}'}$ is defined by ordinary structural recursion on the arity $\sigma$ as follows:

| $\sigma$ | $(\bar{a}, \bar{a}') \in \mathbb{A}^\sigma \otimes \mathbb{A}^\sigma$ | $\mapsto$ | $\tau_{\bar{a},\bar{a}'} \in \mathit{Perm}$ |
|---|---|---|---|
| $\mathsf{a} \in \Sigma_{\mathrm{A}}$ | $((), ())$ | $\mapsto$ | $\iota$ |
| $\mathsf{s} \in \Sigma_{\mathrm{D}}$ | $((), ())$ | $\mapsto$ | $\iota$ |
| $1$ | $((), ())$ | $\mapsto$ | $\iota$ |
| $\sigma_1 * \sigma_2$ | $((\bar{a}_1, \bar{a}_2), (\bar{a}'_1, \bar{a}'_2))$ | $\mapsto$ | $\tau_{\bar{a}_1,\bar{a}'_1} \circ \tau_{\bar{a}_2,\bar{a}'_2}$ |
| $«\mathsf{a}»\sigma_1$ | $((a, \bar{a}_1), (a', \bar{a}'_1))$ | $\mapsto$ | $(a\ a') \circ \tau_{\bar{a}_1,\bar{a}'_1}$ |

Although the clauses defining $\tau_{\bar{a},\bar{a}'}$ make sense for any pair $\bar{a}, \bar{a}'$, we restrict to pairs satisfying $\bar{a} \# \bar{a}'$ to ensure that $\tau_{\bar{a},\bar{a}'}$ enjoys many properties of single-atom transpositions, $(a\ a')$. In particular one can easily prove by ordinary structural induction for arities that:

$$\pi \circ \tau_{\bar{a},\bar{a}'} \circ \pi^{-1} = \tau_{(\pi\cdot\bar{a}),(\pi\cdot\bar{a}')} \qquad (\text{any } \pi \in \mathit{Perm}) \tag{120}$$

$$\mathit{supp}(\tau_{\bar{a},\bar{a}'}) \subseteq \mathit{supp}(\bar{a}) \cup \mathit{supp}(\bar{a}') \tag{121}$$

$$\tau_{\bar{a},\bar{a}'} = \tau_{\bar{a}',\bar{a}} \tag{122}$$

$$\tau_{\bar{a},\bar{a}'} \circ \tau_{\bar{a},\bar{a}'} = \iota \tag{123}$$

$$\tau_{\bar{a},\bar{a}'} \cdot \bar{a} = \bar{a}' \ . \tag{124}$$

Part (ii) of the following result generalises the "some/any" Theorem 14 from single atoms to elements of $\mathbb{A}^\sigma$.

**Lemma 35.** (i) *For all finite sets of atoms $A$, there is some $\bar{a} \in \mathbb{A}^\sigma$ with $\bar{a} \# A$.*

(ii) *For all finitely supported subsets $S \in P_{\mathrm{fs}}(\mathbb{A}^\sigma)$, if $S$ is supported by the finite set of atoms $A$, then the following statements are equivalent.*

$$(\forall \bar{a} \in \mathbb{A}^\sigma)\ \bar{a} \mathrel{\#} A \ \Rightarrow\ \bar{a} \in S \tag{125}$$

$$(\exists \bar{a} \in \mathbb{A}^\sigma)\ \bar{a} \mathrel{\#} A \ \&\ \bar{a} \in S\,. \tag{126}$$

*Proof.* Part (i) follows easily by ordinary structural induction for arities $\sigma \in Ar(\Sigma)$. For part (ii), first note that in view of part (i), one just has to show that (126) implies (125). Suppose $\bar{a} \in \mathbb{A}^\sigma$ satisfies $\bar{a} \mathrel{\#} A\ \&\ \bar{a} \in S$. Given any other $\bar{a}' \in \mathbb{A}^\sigma$ with $\bar{a}' \mathrel{\#} A$, we have to show $\bar{a}' \in S$. We can use part (i) to find $\bar{a}'' \in \mathbb{A}^\sigma$ with $\bar{a}'' \mathrel{\#} (A, \bar{a}, \bar{a}')$ and then use the vector-transpositions of Definition 34 to define $\pi \triangleq \tau_{\bar{a}',\bar{a}''} \circ \tau_{\bar{a},\bar{a}''} \in Perm$.[17] By (124), we have $\pi \cdot \bar{a} = \bar{a}'$; and because by (121) $\pi$ is supported by $supp(\bar{a}, \bar{a}', \bar{a}'')$, which is disjoint from $A$ and hence from $supp(S)$, we have $\pi \cdot S = S$. So applying $\pi$ to $\bar{a} \in S$ we get $\bar{a}' \in S$, as required. □

The following result generalises the characterisation of $\alpha$-equivalence mentioned in Example 15. Note that in view of the previous lemma, the right-hand side of the bi-implication in (127) could be replaced by $(\forall \bar{a}'' \in \mathbb{A}^\sigma)\ \bar{a}'' \mathrel{\#} A \ \Rightarrow\ \tau_{\bar{a},\bar{a}''} \cdot \bar{e} = \tau_{\bar{a}',\bar{a}''} \cdot \bar{e}' \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|}$.

**Lemma 36.** *Suppose $\bar{a}, \bar{a}' \in \mathbb{A}^\sigma$ and $\bar{e}, \bar{e}' \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|}$ satisfy $\bar{a} \sharp_\sigma \bar{e}$ and $\bar{a}' \sharp_\sigma \bar{e}'$. If $A$ is a finite set of atoms supporting $(\bar{a}, \bar{a}', \bar{e}, \bar{e}')$, then*

$$\begin{aligned}&\bar{a}.\,\bar{e} = \bar{a}'.\,\bar{e}' \in \mathsf{T}_\alpha(\Sigma)_\sigma \ \Leftrightarrow \\ &\qquad (\exists \bar{a}'' \in \mathbb{A}^\sigma)\ \bar{a}'' \mathrel{\#} A \ \&\ \tau_{\bar{a},\bar{a}''} \cdot \bar{e} = \tau_{\bar{a}',\bar{a}''} \cdot \bar{e}' \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|}\,.\end{aligned} \tag{127}$$

*Proof.* This can be proved by ordinary structural induction for arities $\sigma$. The induction step for atom-binding arities uses the corresponding property (28) of $=_\alpha$. □

We can now complete the proof of Theorem 23. We are given subsets $(S_{\mathsf{s}} \in P_{\mathrm{fs}}(\mathsf{T}_\alpha(\Sigma)_{\mathsf{s}}) \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ supported by the finite set of atoms $A$ and satisfying $(\mathrm{IH}_K)$ for each $K \in \Sigma_{\mathrm{C}}$. Define $(S_\sigma \in P_{\mathrm{fs}}(\mathsf{T}_\alpha(\Sigma)_\sigma) \mid \sigma \in Ar(\Sigma))$ as in the right-hand column in Figure 3. It is not hard to see that all of these subsets are also supported by $A$ and hence so is their union $S \triangleq \bigcup_{\sigma \in Ar(\Sigma)} S_\sigma$. If $S = \mathsf{T}_\alpha(\Sigma)$, then $S_{\mathsf{s}} = \mathsf{T}_\alpha(\Sigma)_{\mathsf{s}}$ for each $\mathsf{s} \in \Sigma_{\mathrm{D}}$; and to prove $S = \mathsf{T}_\alpha(\Sigma)$ we just have to prove that this $S$ satisfies the conditions (40)–(44) of Theorem 21. Conditions (40) and (42)–(44) are immediate from the definition of $(S_\sigma \mid \sigma \in Ar(\Sigma))$ in Figure 3. For condition (41), given $(K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}}$ and $e \in S_\sigma$, we have to show that $K\,e \in S_{\mathsf{s}}$. First note that by applying Lemma 35(ii) to the subset $\{\bar{a} \in \mathbb{A}^\sigma \mid (\forall \bar{e} \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|})\ \bar{a} \sharp_\sigma \bar{e}\ \&\ \bar{e} \in S^{|\sigma|} \ \Rightarrow\ K\,\bar{a}.\,\bar{e} \in S_{\mathsf{s}}\}$, which is supported by $A$, to see that $(\mathrm{IH}_K)$ is equivalent to

$$\begin{aligned}&(\forall \bar{a} \in \mathbb{A}^\sigma)\ \bar{a} \mathrel{\#} A \ \Rightarrow\ (\forall \bar{e} \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|})\ \bar{a} \sharp_\sigma \bar{e}\ \&\ \bar{e} \in S^{|\sigma|} \ \Rightarrow \\ &\qquad\qquad\qquad\qquad\qquad\qquad K\,\bar{a}.\,\bar{e} \in S_{\mathsf{s}}\,.\end{aligned} \tag{128}$$

[17] Since it may not be the case that $\bar{a} \mathrel{\#} \bar{a}'$, we cannot use $\tau_{\bar{a},\bar{a}'}$; so (unlike in the proof of Theorem 14) we resort to swapping via an intermediate, fresh $\bar{a}''$.

By Lemma 33, $e = \bar{a}.\,\bar{e}$ with $\bar{a} \in \mathbb{A}^{\sigma}$, $\bar{e} \in S^{|\sigma|}$, $\bar{a} \mathrel{\#} A$ and $\bar{a} \mathrel{\sharp_{\sigma}} \bar{e}$; so (128) gives $K\,e \in S_{\mathsf{s}}$. Thus we can indeed apply Theorem 21 to conclude that $S$ is the whole of $\mathsf{T}_{\alpha}(\Sigma)$ and hence in particular that each $S_{\mathsf{s}}$ is equal to the whole of $\mathsf{T}_{\alpha}(\Sigma)_{\mathsf{s}}$, completing the proof of Theorem 23.

## B Proof of Second $\alpha$-Structural Recursion Theorem

We will establish the second $\alpha$-structural recursion principle (Theorem 22) via a common strategy for reducing recursion to induction: we first construct relations (using a rule-based inductive definition) that would be the graphs of the required functions $\hat{f}_{\mathsf{s}}$ were they single-valued and total relations—and then prove they are so by applying Theorem 23. A further application of Theorem 23 is needed to show that the functions $\hat{f}_{\mathsf{s}}$ are unique with the stated properties.

So suppose we are given a family of nominal sets $X = (X_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ indexed by the data-sorts of a nominal signature $\Sigma$, and functions $(f_K \in X^{(\sigma)} \to_{\mathrm{fs}} X^{(\mathsf{s})} \mid (K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}})$ all of which are supported by a finite set $A$ of atoms and satisfy (FCB$_K$).

### Existence of the functions $\hat{f}_{\mathsf{s}} \in \mathsf{T}_{\alpha}(\Sigma)_{\mathsf{s}} \to_{\mathrm{fs}} X_{\mathsf{s}}$

Consider the subsets $F_{\mathsf{s}} \subseteq \mathsf{T}_{\alpha}(\Sigma)_{\mathsf{s}} \times X_{\mathsf{s}}$ (for $\mathsf{s} \in \Sigma_{\mathrm{D}}$) that are inductively defined by the following rule

$$\frac{\begin{array}{cccc} K : \sigma \to \mathsf{s} & \bar{a} \in \mathbb{A}^{\sigma} & \bar{e} \in \mathsf{T}_{\alpha}(\Sigma)^{|\sigma|} & \\ \bar{a} \mathrel{\#} A & \bar{a} \mathrel{\sharp_{\sigma}} \bar{e} & (\bar{e}, \bar{x}) \in F^{|\sigma|} & \bar{a} \mathrel{\sharp_{\sigma}} \bar{x} \end{array}}{(K\,\bar{a}.\,\bar{e}, f_K(\bar{a}, \bar{x})_{\sigma}) \in F_{\mathsf{s}}} \tag{129}$$

where $F^{|\sigma|} \subseteq \mathsf{T}_{\alpha}(\Sigma)^{|\sigma|} \times X^{|\sigma|}$ is defined from any family $(F_{\mathsf{s}} \subseteq \mathsf{T}_{\alpha}(\Sigma)_{\mathsf{s}} \times X_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ by ordinary recursion on the structure of arities as follows.

| $\sigma$ | $F^{\lvert\sigma\rvert} \subseteq \mathsf{T}_{\alpha}(\Sigma)^{\lvert\sigma\rvert} \times X^{\lvert\sigma\rvert}$ |
|---|---|
| $\mathsf{a} \in \Sigma_{\mathrm{A}}$ | $\{(a, a) \mid a \in \mathbb{A}_{\mathsf{a}}\}$ |
| $\mathsf{s} \in \Sigma_{\mathrm{D}}$ | $F_{\mathsf{s}}$ |
| $1$ | $\{((), ())\}$ |
| $\sigma_1 * \sigma_2$ | $\{((\bar{e}_1, \bar{e}_2), (\bar{x}_1, \bar{x}_2)) \mid (\bar{e}_1, \bar{x}_1) \in F^{\lvert\sigma_1\rvert}$ & $(\bar{e}_2, \bar{x}_2) \in F^{\lvert\sigma_2\rvert}\}$ |
| «a»$\sigma_1$ | $F^{\lvert\sigma_1\rvert}$ |

It is not hard to see that the set of rules determined by (129) is supported by $A$. Hence by Theorem 12, so are all the subsets $F_{\mathsf{s}}$ (and the subsets $F^{|\sigma|}$ defined from them). For the existence part of Theorem 22 it suffices to show that each $F_{\mathsf{s}}$ is the graph of a function $\hat{f}_{\mathsf{s}}$ from $\mathsf{T}_{\alpha}(\Sigma)_{\mathsf{s}}$ to $X_{\mathsf{s}}$. For then:

- each $\hat{f}_{\mathsf{s}}$ and the functions $\hat{f}^{|\sigma|}$ defined from them as in Figure 2 are supported by $A$, because $F_{\mathsf{s}}$ is;

– the graphs of the functions $\hat{f}^{|\sigma|}$ are the relations $F^{|\sigma|}$, because of the above definition of $F^{|\sigma|}$;

– if $\bar{a} \# A$ and $\bar{a} \natural_\sigma \bar{e}$, then $\bar{a} \natural_\sigma \hat{f}^{|\sigma|}\bar{e}$, because of the definition of $\hat{f}^{|\sigma|}$;

– hence $(\hat{f}_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ satisfies the recursion property (46): for if $\bar{a} \# A$ and $\bar{a} \natural_\sigma \bar{e}$, then $\bar{x} \triangleq \hat{f}^{|\sigma|}\bar{e}$ satisfies $(\bar{e}, \bar{x}) \in F^{|\sigma|}$ and $\bar{a} \natural_\sigma \bar{x}$; so $(K\,\bar{a}.\,\bar{e}, f_K(\bar{a}, \bar{x})_\sigma) \in F_{\mathsf{s}}$ by rule (129) and hence $\hat{f}_{\mathsf{s}}(K\,\bar{a}.\,\bar{e}) = f_K(\bar{a}, \bar{x})_\sigma = f_K(\bar{a}, \hat{f}^{|\sigma|}\bar{e})_\sigma$.

To prove that each $F_{\mathsf{s}}$ is the graph of a function, i.e. that

$$S_{\mathsf{s}} \triangleq \{e \in \mathsf{T}_\alpha(\Sigma)_{\mathsf{s}} \mid (\exists!\, x \in X_{\mathsf{s}})\ (e, x) \in F_{\mathsf{s}}\}$$

is the whole of $\mathsf{T}_\alpha(\Sigma)_{\mathsf{s}}$, we apply Theorem 23. Note that from the way it is defined, each $S_{\mathsf{s}}$ is supported by $A$, because $F_{\mathsf{s}}$ is. So we just have to prove that $(S_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ satisfies ($\mathrm{IH}_K$) for each $(K : \sigma \to \mathsf{s}) \in \Sigma_{\mathrm{C}}$. By Lemma 35(i), there is some $\bar{a} \in \mathbb{A}^\sigma$ with $\bar{a} \# A$. We prove ($\mathrm{IH}_K$) for this $\bar{a}$ by showing that for each $\bar{e} \in S^{|\sigma|}$ with $\bar{a} \natural_\sigma \bar{e}$ it is the case that $K\,\bar{a}.\,\bar{e} \in S_{\mathsf{s}}$.

First note that by the definitions of $S^{|\sigma|}$ from $(S_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$ (in Figure 3) and $F^{|\sigma|}$ from $(F_{\mathsf{s}} \mid \mathsf{s} \in \Sigma_{\mathrm{D}})$, we have

$$(\forall \bar{e} \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|})\ \bar{e} \in S^{|\sigma|} \ \Rightarrow\ (\exists!\, \bar{x} \in X^{|\sigma|})\ (\bar{e}, \bar{x}) \in F^{|\sigma|}\ . \tag{130}$$

In particular it follows that if $\bar{e} \in S^{|\sigma|}$ and $(\bar{e}, \bar{x}) \in F^{|\sigma|}$, then $\mathit{supp}(\bar{x}) \subseteq A \cup \mathit{supp}(\bar{e})$ (since $A$ supports $F^{|\sigma|}$); and this in turn implies (by induction on the structure of arities $\sigma$) that

$$(\forall \bar{a} \in \mathbb{A}^\sigma, \bar{e} \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|}, \bar{x} \in X^{|\sigma|})\ \bar{a} \natural_\sigma \bar{e}\ \ \&\ \ \bar{e} \in S^{|\sigma|}\ \ \&\ \ (\bar{e}, \bar{x}) \in F^{|\sigma|} \ \Rightarrow \\ \bar{a} \natural_\sigma \bar{x}\ . \tag{131}$$

So by rule (129), if $\bar{e} \in S^{|\sigma|}$ with $\bar{a} \natural_\sigma \bar{e}$, then there is some $\bar{x} \in X^{|\sigma|}$ with $(K\,\bar{a}.\,\bar{e}, f_K(\bar{e}, \bar{x})_\sigma) \in F_{\mathsf{s}}$. Thus to see that $K\,\bar{a}.\,\bar{e} \in S_{\mathsf{s}}$, it just remains to show that if $(K\,\bar{a}.\,\bar{e}, x) \in F_{\mathsf{s}}$, then $x = f_K(\bar{e}, \bar{x})_\sigma$. But if $(K\,\bar{a}.\,\bar{e}, x) \in F_{\mathsf{s}}$ holds, it must have been deduced by an application of rule (129) to

$$\bar{a}' \# A\ \ \&\ \ \bar{a}' \natural_\sigma \bar{e}'\ \ \&\ \ (\bar{e}', \bar{x}') \in F^{|\sigma|}\ \ \&\ \ \bar{a}' \natural_\sigma \bar{x}'$$

with $K\,\bar{a}.\,\bar{e} = K\,\bar{a}'.\,\bar{e}'$ and $x = f_K(\bar{a}', \bar{x}')_\sigma$. So $\bar{a}.\,\bar{e} = \bar{a}'.\,\bar{e}'$ and by Lemma 36, $\tau_{\bar{a},\bar{a}''} \cdot \bar{e} = \tau_{\bar{a}',\bar{a}''} \cdot \bar{e}'$, for some $\bar{a}'' \# (A, \bar{a}, \bar{e}, \bar{a}', \bar{e}')$. By (121), the atom-permutations $\tau_{\bar{a},\bar{a}''}$ and $\tau_{\bar{a}',\bar{a}''}$ have their support disjoint from $A$ and hence from the support of $F^{|\sigma|}$ and $S^{|\sigma|}$. Therefore

$$\begin{gathered}(\tau_{\bar{a},\bar{a}''} \cdot \bar{e}, \tau_{\bar{a},\bar{a}''} \cdot \bar{x}) = \tau_{\bar{a},\bar{a}''} \cdot (\bar{e}, \bar{x}) \in \tau_{\bar{a},\bar{a}''} \cdot F^{|\sigma|} = F^{|\sigma|}\ , \\ (\tau_{\bar{a}',\bar{a}''} \cdot \bar{e}', \tau_{\bar{a}',\bar{a}''} \cdot \bar{x}') = \tau_{\bar{a}',\bar{a}''} \cdot (\bar{e}', \bar{x}') \in \tau_{\bar{a}',\bar{a}''} \cdot F^{|\sigma|} = F^{|\sigma|}\ , \\ \tau_{\bar{a}',\bar{a}''} \cdot \bar{e}' = \tau_{\bar{a},\bar{a}''} \cdot \bar{e} \in \tau_{\bar{a},\bar{a}''} \cdot S^{|\sigma|} = S^{|\sigma|}\end{gathered}$$

and hence by (130), $\tau_{\bar a,\bar a''} \cdot \bar x = \tau_{\bar a',\bar a''} \cdot \bar x'$. Note that from (FCB$_K$), by Lemma 35(ii) we have

$$(\forall \bar a \in \mathbb{A}^\sigma)\ \bar a \mathrel{\#} A \ \Rightarrow\ (\forall \bar x \in X^{|\sigma|})\ \bar a \mathrel{\natural_\sigma} \bar x \ \Rightarrow\ \bar a \mathrel{\#} f_K(\bar a, \bar x)_\sigma\ . \tag{132}$$

So we have $\bar a \mathrel{\#} f_K(\bar a,\bar x)_\sigma$ and $\bar a' \mathrel{\#} f_K(\bar a',\bar x')_\sigma$; and by choice of $\bar a''$ we also have $\bar a'' \mathrel{\#} (f_K(\bar a,\bar x)_\sigma, f_K(\bar a',\bar x')_\sigma)$. Since the atom-permutations $\tau_{\bar a,\bar a''}$ and $\tau_{\bar a',\bar a''}$ have their supports in $A$ and hence disjoint from $f_K$, $f_K(\bar a,\bar x)_\sigma$ and $f_K(\bar a',\bar x')_\sigma$, we have

$$\begin{aligned} f_K(\bar a,\bar x)_\sigma &= \tau_{\bar a,\bar a''} \cdot f_K(\bar a,\bar x)_\sigma = f_K(\tau_{\bar a,\bar a''} \cdot \bar a, \tau_{\bar a,\bar a''} \cdot \bar x)_\sigma \\ f_K(\bar a',\bar x')_\sigma &= \tau_{\bar a',\bar a''} \cdot f_K(\bar a',\bar x')_\sigma = f_K(\tau_{\bar a',\bar a''} \cdot \bar a', \tau_{\bar a',\bar a''} \cdot \bar x')_\sigma\ . \end{aligned}$$

From above $\tau_{\bar a,\bar a''} \cdot \bar e = \tau_{\bar a',\bar a''} \cdot \bar e'$; and by (124), $\tau_{\bar a,\bar a''} \cdot \bar a = \bar a'' = \tau_{\bar a',\bar a''} \cdot \bar a'$. So $f_K(\bar a,\bar x)_\sigma = f_K(\bar a',\bar x')_\sigma = x$, as required.

### Uniqueness of the functions $\hat f_{\mathsf s} \in \mathsf{T}_\alpha(\Sigma)_{\mathsf s} \to_{\mathrm{fs}} X_{\mathsf s}$

Suppose $(\hat f'_{\mathsf s} \in \mathsf{T}_\alpha(\Sigma)_{\mathsf s} \to_{\mathrm{fs}} X_{\mathsf s} \mid \mathsf s \in \Sigma_{\mathrm D})$ is also supported by $A$ and satisfies property (46). To see that $\hat f'_{\mathsf s} = \hat f_{\mathsf s}$ if suffices to show that $S_{\mathsf s} \triangleq \{e \in \mathsf{T}_\alpha(\Sigma)_{\mathsf s} \mid \hat f_{\mathsf s}\, e = \hat f'_{\mathsf s}\, e\}$ is the whole of $\mathsf{T}_\alpha(\Sigma)$ for each $\mathsf s \in \Sigma_{\mathrm D}$. This follows almost immediately by Theorem 23; one just has to check that the subsets $S^{|\sigma|}$ defined from this $(S_{\mathsf s} \mid \mathsf s \in \Sigma_{\mathrm D})$ satisfy $S^{|\sigma|} \subseteq \{\bar e \in \mathsf{T}_\alpha(\Sigma)^{|\sigma|} \mid \hat f^{|\sigma|}\, \bar e = \hat f'^{|\sigma|}\, \bar e\}$.

## References

[1] B. E. Aydemir, A Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. E. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. Mechanised metatheory for the masses: The POPLMARK challenge. In J. Hurd and T. Melham, editors, *18th International Conference on Theorem Proving in Higher Order Logics: TPHOLs 2005*, volume 3603 of *Lecture Notes in Computer Science*, pages ?–? Springer-Verlag, 2005. `www.cis.upenn.edu/group/proj/plclub/mmm/`.

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

[3] U. Berger and H. Schwichtenberg. An inverse of the evaluation functional for typed $\lambda$-calculus. In *6th Annual Symposium on Logic in Computer Science*, pages 203–211. IEEE Computer Society Press, Washington, 1991.

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

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

[6] 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.

[7] P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2), 2000.

[8] P. Dybjer and A. Filinski. Normalization and partial evaluation. In G. Barthe, P. Dybjer, and J. Saraiva, editors, *Applied Semantics, Advanced Lectures*, volume 2395 of *Lecture Notes in Computer Science, Tutorial*, pages 137–192. Springer-Verlag, 2002. International Summer School, APPSEM 2000, Caminha, Portugal, September 9–15, 2000.

[9] M. P. Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. In *Proceedings of the 4th iNternational ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, October 6-8, (PPDP 2002), Pittsburgh, PA, USA*, pages 26–37. ACM Press, October 2002.

[10] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *14th Annual Symposium on Logic in Computer Science*, pages 193–202. IEEE Computer Society Press, Washington, 1999.

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

[12] 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.

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

[14] A. D. Gordon and T. Melham. Five axioms of alpha-conversion. In *Theorem Proving in Higher Order Logics, 9th International Conference*, volume 1125 of *Lecture Notes in Computer Science*, pages 173–191. Springer-Verlag, 1996.

[15] T. G. Griffin. Notational definition — a formal account. In *3rd Annual Symposium on Logic in Computer Science*, pages 372–383. IEEE Computer Society Press, 1988.

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

[17] F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on nominal algebras in HOAS. In F. Orejas, P. G. Spirakis, and

J. van Leeuwen, editors, *28th International Colloquium on Automata, Languages and Programming, ICALP 2001, Crete, Greece, July 2001. Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 963–978. Springer-Verlag, Heidelberg, 2001.

[18] M. Norrish. Recursive function definition for types with binders. In *Theorem Proving in Higher Order Logics, 17th International Conference*, volume 3223 of *Lecture Notes in Computer Science*, pages 241–256. Springer-Verlag, 2004.

[19] F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proc. ACM-SIGPLAN Conference on Programming Language Design and Implementation*, pages 199–208. ACM Press, 1988.

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

[21] A. M. Pitts. Alpha-structural recursion and induction (extended abstract). In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford UK, August 2005, Proceedings*, volume 3603 of *Lecture Notes in Computer Science*, pages 17–34. Springer-Verlag, 2005.

[22] 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.

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

[24] F. Pottier. An overview of C$\alpha$ml. Submitted, June 2005.

[25] D. Sangiorgi and D. Walker. *The $\pi$-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

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

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

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

[29] C. Urban and C. Tasson. Nominal techniques in Isabelle/HOL. In *20th International Conference on Automated Deduction, CADE-20, Tallinn, Estonia, July 2005*, volume ???? of *Lecture Notes in Computer Science*, pages ?–? Springer-Verlag, 2005.