Abstract Syntax and Variable Binding

(Extended Abstract)

Marcelo Fiore
COGS
Univ. of Sussex

Gordon Plotkin
LFCS
Univ. of Edinburgh

Daniele Turi*
LFCS
Univ. of Edinburgh

Abstract

We develop a theory of abstract syntax with variable binding. To every binding signature we associate a category of models consisting of variable sets *endowed with compatible algebra and substitution structures. The syntax generated by the signature is the initial model. This gives a notion of initial algebra semantics encompassing the traditional one; besides compositionality, it automatically verifies the semantic substitution lemma.*

Introduction

It has long been recognised that the essential syntactical structure of programming languages is not that given by their concrete or surface syntax—as expressed, say, by a language description in BNF oriented to parsing (there the parse trees contain much information useless for language processing). Rather, the deep structure of a phrase should reflect its semantic import.

McCarthy [24] coined the term *abstract syntax* for such structure, which is typically given as a tree with its top node labelled by the main semantic constituent, or, equivalently, by a term of first-order logic. Abstract syntax has both *synthetic* and *analytic* aspects: the former concerns the *constructors* needed to form phrases, the latter the *destructors* (*predicates* and *selectors*) needed to take them apart [21]. Burstall [6] contributed *structural recursion*—a generalised form of primitive recursion—to analytic syntax, with an associated principle of *structural induction*.

The *algebraic* point of view of the ADJ group [14] (see also [32]) regards abstract syntax as the initial algebra (of the constructors) and semantics as the unique homomorphism to a semantic algebra (the *model*). Structural recursion then arises from initiality. The *categorical* view regards the algebras as those of an associated signature functor: the initial algebra has an isomorphism as structure map and its inverse is analytic syntax. Finally, in the *recursive type* view [32], the initial algebra is obtained as the solution to a recursive (set) equation; this leads one to a treatment of syntax within programming languages as exemplified in the ML/LCF approach [15].

The first-order view can be problematic. In particular, when dealing with context-sensitive aspects of syntax, it does not account satisfactorily for *variable binding,* with its allied notions of scope, free and bound occurrences, α-equivalence, production of fresh variables, and substitution (*e.g.*, in λ-calculus, CCS with value passing, π-calculus, logical quantifiers, and derivative and integral expressions). Useful context-sensitive extensions of BNF certainly exist, such as attribute grammars and van Wijngaarden grammars; however, they do not in themselves give an account of deep structure.

One proposal for an abstract treatment, originating with Church [7], is that of *higher-order abstract syntax*; there one uses the binding apparatus provided by the typed λ-calculus to express all other forms of binding—see [29, 27, 30, 28]. Normal forms (of ground type) play the rôle of first-order terms, but with first-order signatures being replaced by second-order ones. (The binding analogue of trees is provided by the *term graphs* first introduced by Wadsworth—see, *e.g.*, [17].) A form of analytic syntax is given by matching or unification in associated logic programming languages [27, 30]. In implementations, De Bruijn's terms [9] are used to provide "α-equivalence normal forms".

Unfortunately, in the higher-order abstract syntax approach, many of the desirable properties mentioned above, such as accounts of structural recursion and induction, and recursive equations for abstract syntax, are missing, or at least not fully developed (see [26, 25, 10]).

In this paper, we provide a (categorical) algebraic view of syntax with variable binding. The analogue to universal algebra is the theory of binding algebras originating in the work of Aczel [1]—see also [20, 31, 37]. We replace algebras over sets by *binding algebras* over *variable sets*. Formally, variable sets are (covariant) *presheaves* and the funda-

[*] Research supported by the EU TMR research programme.

mental idea is to turn contexts into the "index category" of the presheaves.

We obtain a notion of *binding signature* in which binding is again expressed by second-order types, but now using a special presheaf of variables or, equivalently, by a first-order signature extended with a notion of *differentiation* (*cf.* [16]). Our models, the *binding algebras*, are then presheaves endowed with both an algebra structure (corresponding to the operations in the signature) and a substitution structure compatible with each other.

Abstract syntax is the initial such model, with the algebra structure obtained as the solution to a recursive (presheaf) equation and substitution defined by an associated structural recursion. The unique homomorphism from the syntax to another model (*initial algebra semantics*) preserves the constructors (*i.e.*, is compositional) and the extra substitution structure (*i.e.*, verifies the semantic substitution lemma).

Organisation of the paper. We start in Section 1 by advocating the use of the category of (covariant) presheaves $\mathbf{Set}^{\mathbb{F}}$ as a suitable mathematical universe in which to deal with syntax with variable binding. The index category $\mathbb{F}$ is a skeleton of the category of finite sets and functions; it provides a notion of cartesian context allowing for exchange, weakening, and contraction. The structure of $\mathbf{Set}^{\mathbb{F}}$ relevant to modelling variable binding is studied in detail.

The study of signatures with variable binding in our framework is carried out in Section 2. Our main result here is that the presheaf of terms (with binders) associated to a (binding) signature has an abstract universal characterisation as a free algebra over a presheaf of variables. This result is exploited in two directions: to give implementations of abstract syntax (with variable binding) à la De Bruijn, and to provide semantics by initiality. These two applications are exemplified using the λ-calculus.

Section 3 is devoted to substitution. We treat both single-variable and simultaneous substitution. The former is handled by introducing the notion of *substitution algebra*; the latter first by the elementary *abstract clones* and then, more abstractly, as certain *monoids* in the category of presheaves. The three presentations are shown to be equivalent. We also define two "categorical programs" for substitution by structural recursion.

In Section 4, we define the category of models of a binding signature; the presheaf of terms is the initial such model. We exemplify the corresponding initial algebra semantics again using the λ-calculus. This semantics, besides being *compositional*, automatically verifies the *semantic substitution lemma*.

Future work. Various directions for further work are possible; we mention but a few here. First, the syntactic counterpart of our treatment of substitution by *categorical* structural recursion (*i.e.*, parameterised initiality) will be worked out. In particular, we envisage a type theory based on the internal language of our semantic universe for manipulating abstract syntax with binding. Again, structural induction principles for reasoning about abstract syntax with binding should be available within our framework.

Second, the investigation of more sophisticated syntax in our setting will be pursued. Multi-sorted binding signatures (like the simply typed λ-calculus) can be easily accommodated; various linear settings (*cf.* operads [22]), in contrast to the cartesian one explored here, also seem to fit; type theories with dependent types are yet to be tackled. Connections between our approach and the general theory of substitution provided by *clubs* [18, 19] will also be investigated.

Third, theories of operational semantics with binding will be developed along the lines of [36]. Preliminary results indicate that some interesting syntactic formats of well-behaved operational rules for languages with variable binding can be obtained.

1. The universe of types

We present the universe of types within which we work. Our intent is to consider a notion of type broad enough to encompass *syntax with variable binding* (and, more generally, algebras for binding signatures) in a framework with rich type structure (*i.e.*, type constructors and operations on types). To motivate our choice we start by examining the structure of the set of λ-terms.

Λ. In the course of our discussion we will consider the set of (untyped) λ-terms Λ_{Var} given by the following grammar.

$$\begin{array}{rcl} x \in \mathrm{Var} & ::= & x_i \quad (i \in \mathbb{N}^+) \quad , \\ t \in \Lambda_{\mathrm{Var}} & ::= & x \mid \lambda x.t' \mid t_1 t_2 \quad . \end{array}$$

In untyped settings, the treatment of the operator λ as a *binder* is typically dealt with by introducing the notion of free/bound variable. However, as is well-known and commonly used in typed settings, this information may be presented by judgements, consisting of a term together with a context, subject to a well-formedness condition. To simplify the exposition we will consider the following well-formedness rules which provide canonical representatives for α-equivalence classes of λ-terms by the method of *De Bruijn levels* [9].

$$\begin{array}{cc} \dfrac{1 \leq i \leq n}{x_1, \ldots, x_n \vdash x_i} & \dfrac{x_1, \ldots, x_n, x_{n+1} \vdash t}{x_1, \ldots, x_n \vdash \lambda x_{n+1}.t} \\ \multicolumn{2}{c}{\dfrac{x_1, \ldots, x_n \vdash t_1 \qquad x_1, \ldots, x_n \vdash t_2}{x_1, \ldots, x_n \vdash t_1 t_2}} \end{array} \tag{1}$$

Conceptually, the passage from the approach based on free/bound variables to the one based on contexts consists in turning the free-variable function $\mathrm{FV} : \Lambda_{\mathrm{Var}} \longrightarrow \mathcal{P}(\mathrm{Var})$

into extra structure on terms. As we will see, the latter viewpoint is important for bringing out the structure of the *type* Λ of λ-terms (modulo α-equivalence). As a first step in this direction notice that contexts *stratify* λ-terms. Indeed, for all $n \in \mathbb{N}$, we have a bijective correspondence

$$\Lambda_\ell(n) \cong \Lambda_\alpha(n) : t \mapsto [t]_\alpha \qquad (2)$$

where

$$\Lambda_\ell(n) \overset{\text{def}}{=} \{\ t \in \Lambda_{\text{Var}} \mid x_1, \ldots, x_n \vdash t\ \} \ ,$$

$$\Lambda_\alpha(n) \overset{\text{def}}{=} \{\ [t]_\alpha \mid t \in \Lambda_{\text{Var}} \wedge \text{FV}(t) \subseteq \{\ x_1, \ldots, x_n\ \}\ \} \ .$$

Next, note that the well-formedness rules (1) induce (and in fact correspond to) the following bijection: for all $n \in \mathbb{N}$,

$$\begin{array}{rcl} n + \Lambda_\ell(n+1) + \Lambda_\ell(n) \times \Lambda_\ell(n) & \xrightarrow{\cong} & \Lambda_\ell(n) \\ i \in n & \mapsto & x_i \\ t \in \Lambda_\ell(n+1) & \mapsto & \lambda x_{n+1}.t \\ (t_1, t_2) \in \Lambda_\ell(n) \times \Lambda_\ell(n) & \mapsto & t_1 t_2 \end{array} \qquad (3)$$

where, by abuse of notation, we write n for the set $\{\ 1, \ldots, n\ \}$.

To conclude the analysis of the structure of Λ we first need to examine the structure of contexts.

The structure of contexts. The notion of context relevant to this paper is that of (untyped) *cartesian context*. This is reflected in the operations which we allow for context manipulation: *exchange*, *weakening*, and *contraction*. These operations, when closed under composition, yield all functions between contexts. Thus we take the category of cartesian contexts $\mathbb{F}$ to be the full subcategory of $\mathbf{Set}$ with objects $\{\ 1, \ldots, n\ \}$ $(n \in \mathbb{N})$ representing generic abstract contexts with n variables. (Note that $\mathbb{F}$ is isomorphic to the category of finite cardinals and functions.)

A conceptual description of the category $\mathbb{F}$ is as the free cocartesian category on one object (see *e.g.* [23, § VIII.4, Lemma 1]). As such, it may be understood as being generated from an initial object (the generic abstract empty context) by an operation of context extension, $(-) + 1$, with a generic abstract context with one variable, 1. Following this viewpoint, we will henceforth consider the category $\mathbb{F}$ as equipped with a chosen coproduct structure

$$n \xrightarrow{\mathsf{old}_n} n + 1 \xleftarrow{\mathsf{new}_n} 1 \quad .$$

In particular, we have *atomic* operations of exchange, weakening, and contraction, respectively given by:

$$\begin{array}{rcl} \mathsf{s} & \overset{\text{def}}{=} & [\mathsf{new}_1, \mathsf{old}_1] : 2 \to 2 \quad , \\ \mathsf{w} & \overset{\text{def}}{=} & \mathsf{old}_0 : 0 \to 1 \quad , \\ \mathsf{c} & \overset{\text{def}}{=} & [\text{id}_1, \text{id}_1] : 2 \to 1 \quad . \end{array} \qquad (4)$$

We are now in a position to spell out the structure of Λ.

The structure of Λ. Since contexts stratify terms, it follows that the operations on contexts *act* on them. Indeed, every function $\rho : m \to n$ in $\mathbb{F}$ (thought of as a renaming of variables) induces an action

$$\begin{tikzcd} \Lambda_\ell(m) \arrow[r, "\Lambda_\ell(\rho)"] \arrow[d, "\cong"'] \arrow[dr, phantom, "\text{def}"] & \Lambda_\ell(n) \\ \Lambda_\alpha(m) \arrow[r, "\Lambda_\alpha(\rho)"'] & \Lambda_\alpha(n) \arrow[u, "\cong"'] \end{tikzcd}$$

where $\Lambda_\alpha(\rho) : t \mapsto t\{x_{\rho 1}/x_1, \ldots, x_{\rho m}/x_m\}$, which is *functorial* in the sense that

$$\Lambda_\ell(\text{id}_n) = \text{id}_{\Lambda_\ell(n)} \quad , \quad \Lambda_\ell(\rho' \circ \rho) = \Lambda_\ell(\rho') \circ \Lambda_\ell(\rho) \quad ,$$

for all $n \in \mathbb{F}$ and for all $\rho : n \to n'$ and $\rho' : n' \to n''$ in $\mathbb{F}$. That is, Λ_ℓ is an object of the presheaf category

$$\mathcal{F} \overset{\text{def}}{=} \mathbf{Set}^{\mathbb{F}} \quad ,$$

which we take to be our *universe of types*. Following common usage, these *types* are referred to as **presheaves**, see [23]. Clearly, Λ_α is also a presheaf in $\mathcal{F}$. The presheaf of variables Var is given, for $n \in \mathbb{F}$ and ρ in $\mathbb{F}$, by

$$\text{Var}(n) = \{\ x_1, \ldots, x_n\ \} \ , \quad \text{Var}(\rho) : x_i \mapsto x_{\rho i} \ .$$

For a slightly more involved example consider the presheaf $\text{L} : \mathbb{F} \to \mathbf{Set}$ with $\text{L}(n)$ given by the left hand side of the bijection (3) and equipped with the functorial action

$$\text{L}(\rho) \overset{\text{def}}{=} \rho + \Lambda_\ell(\rho + \text{id}_1) + \Lambda_\ell(\rho) \times \Lambda_\ell(\rho) \qquad (5)$$

for every ρ in $\mathbb{F}$.

An important non-syntactic example is provided by the *presheaf of operations* (from A to B) $\langle A, B \rangle$ for objects A, B in a cartesian category $\mathcal{C}$:

$$\begin{array}{l} \langle A, B \rangle(n) = \mathcal{C}(A^n, B) \\ \langle A, B \rangle(\rho) : f \mapsto f \circ \langle \pi_{\rho 1}, \ldots, \pi_{\rho_m} \rangle \end{array} \qquad (6)$$

for $n \in \mathbb{F}$ and $\rho : m \to n$ in $\mathbb{F}$. In particular, the presheaf $\langle A, A \rangle$ is the so-called **clone of operations** on A.

Recall that a map $f : A \to B$ between presheaves A and B in $\mathcal{F}$ is a natural transformation, *i.e.*, an indexed family of functions $\{\ f_n : A(n) \to B(n)$ in $\mathbf{Set}\ \}_{n \in \mathbb{F}}$ subject to the following naturality condition: for all $\rho : m \to n$ in $\mathbb{F}$, $f_n \circ A(\rho) = B(\rho) \circ f_m$. The bijection (2) yields an isomorphism of presheaves $\Lambda_\ell \cong \Lambda_\alpha$ in $\mathcal{F}$. More interestingly, notice that the particular implementation of λ-terms (modulo α-equivalence) adopted in our exposition by the method of De Bruijn levels is reflected in the mathematical structures under consideration. Indeed, the bijection (3) yields a natural isomorphism of presheaves $\text{L} \cong \Lambda_\ell$ in $\mathcal{F}$ if and only if the chosen coproduct structure on $\mathbb{F}$ is taken to be the one with $\mathsf{old}_n(i) = i \; (1 \leq i \leq n)$ and $\mathsf{new}_n = n+1$ (*cf.* the rule for λ-introduction in (1)). We will see in the next section that an implementation of λ-terms (modulo α-equivalence) by the method of *De Bruijn indices* [9] is also available in our framework.

We conclude with a description of the type constructors and the operations on types (*cf.* [11]) in the universe $\mathcal{F}$ that will be needed in the rest of the paper. As an application, we will show how type constructors may be used to provide a structural definition of the presheaf L of (5).

The structure of $\mathcal{F}$. The category $\mathcal{F}$ is a well-known and interesting topos (see [23, § VIII.4]). Many of the constructions that follow in this and later sections can be cast in the language of topos theory. However, we do not emphasise this viewpoint here; rather we adopt a presentation that generalises to other *contexts*.

Sums, products, and exponentials: $\mathcal{F}$ is a complete and cocomplete cartesian closed category, with limits and colimits computed pointwise (see [23]).

V: The presheaf (of *abstract variables*) $\mathrm{V} \in \mathcal{F}$ is obtained by embedding the generic abstract context with one variable in $\mathbb{F}$ into $\mathcal{F}$ via Yoneda. Explicitly, V is the embedding of $\mathbb{F}$ in $\mathbf{Set}$ given by

$$\mathrm{V}(n) = n \quad (n \in \mathbb{F}) \; ; \quad \mathrm{V}(\rho) = \rho \quad (\rho \text{ in } \mathbb{F}) \; .$$

δ: The type constructor (for *context extension*) $\delta : \mathcal{F} \longrightarrow \mathcal{F}$ is obtained from the operation of context extension $(-){+}1 : \mathbb{F} \longrightarrow \mathbb{F}$ by precomposition:

$$\delta(=) \stackrel{\mathrm{def}}{=} (=) \circ ((-){+}1) \quad .$$

In elementary terms, for $A \in \mathcal{F}$, the presheaf δA is given, for $n \in \mathbb{F}$ and ρ in $\mathbb{F}$, by

$$(\delta A)(n) = A(n+1) \; , \quad (\delta A)(\rho) = A(\rho + \mathrm{id}_1) \quad ;$$

and, for $f : A \longrightarrow B$ in $\mathcal{F}$, the map $\delta f : \delta A \longrightarrow \delta B$ is given by

$$(\delta f)_n = f_{n+1} : A(n+1) \longrightarrow B(n+1) \quad (n \in \mathbb{F}) \quad .$$

Thus, intuitively, an element of type δA in the context n is an element of type A in the extended context $n+1$.

The operations on contexts extend from $\mathbb{F}$ to $\mathcal{F}$ in the same vein. For instance, the operations in (4) respectively give rise to the natural transformations $\mathsf{swap} : \delta^2 \overset{.}{\longrightarrow} \delta^2$, $\mathsf{up} : \mathrm{Id} \overset{.}{\longrightarrow} \delta$, $\mathsf{contract} : \delta^2 \overset{.}{\longrightarrow} \delta$ with the following explicit descriptions: for $A \in \mathcal{F}$ and $n \in \mathbb{F}$,

$$\begin{array}{rcl} \mathsf{swap}_{A,n} & = & A(\mathrm{id}_n + \mathsf{s}) : A(n+2) \longrightarrow A(n+2) \;\; , \\ \mathsf{up}_{A,n} & = & A(\mathrm{id}_n + \mathsf{w}) : A(n) \longrightarrow A(n+1) \;\; , \\ \mathsf{contract}_{A,n} & = & A(\mathrm{id}_n + \mathsf{c}) : A(n+2) \longrightarrow A(n+1) \;\; . \end{array}$$

Note that, as the passage from $\mathbb{F}$ to $\mathcal{F}$ is given by precomposition, it preserves equational structures. In particular, the monad $((-){+}1, \{ \; \mathrm{id}_n + \mathsf{w} \; \}, \{ \; \mathrm{id}_n + \mathsf{c} \; \})$ on $\mathbb{F}$ yields the monad $(\delta, \mathsf{up}, \mathsf{contract})$ on $\mathcal{F}$.

We examine some properties of δ. First we note that, by construction, δ has both a left and a right adjoint; hence it preserves both limits and colimits (as a simple calculation will also show). These adjoints are given by the following natural bijective correspondences

$$\frac{X \longrightarrow \delta A}{\overline{X \times \mathrm{V} \longrightarrow A}} \qquad\qquad \frac{\delta A \longrightarrow Y}{\overline{A \longrightarrow \langle \mathrm{V}+1, Y \rangle}} \tag{7}$$

Second, observe that the first correspondence above shows that

$$\delta(-) \quad \dot{\cong} \quad (-)^{\mathrm{V}} \quad , \tag{8}$$

and states that the elements of type δA in the context X are the elements of type A in the extended context $X \times \mathrm{V}$. Finally, we note the important fact that the diagram

$$\mathrm{V} \xrightarrow{\mathsf{old}} \delta \mathrm{V} \xleftarrow{\mathsf{new}} 1 \tag{9}$$

is a coproduct in $\mathcal{F}$.

It should be clear that the type L of (5) equals the structured type $\mathrm{V} + \delta\Lambda_\ell + \Lambda_\ell \times \Lambda_\ell$. We will show in the next section that the inductive type $\mu X.\mathrm{V} + \delta X + X \times X$ characterises the presheaf Λ of λ-terms (modulo α-equivalence).

2. Binding signatures and their algebras

We show that the universe $\mathcal{F}$ provides a suitable setting for modelling binding signatures and their algebras. In particular, we obtain a characterisation of syntax with variable binding by initiality, which generalises the well-known result for the first-order case [14]. This yields a notion of *abstract syntax with variable binding* for which an initial algebra semantics is available.

Syntax with variable binding. A **binding signature** [31] $\Sigma = (O, a)$ consists of a set of *operations* O equipped with an *arity* function $a : O \longrightarrow \mathbb{N}^*$. An operator of arity $\langle n_1, \ldots, n_k \rangle$ has k arguments and binds n_i variables in the i-th argument ($1 \leq i \leq k$). For instance, the signature of the λ-calculus has an operator of arity $\langle 1 \rangle$, *viz.* λ-abstraction with one argument and binding one variable, and an operator of arity $\langle 0, 0 \rangle$, *viz.* application with two arguments and binding no variables.

The terms associated to a binding signature over a set of variables (ranged over by x) are given by the following grammar.

$$t \in \mathrm{T}_\Sigma ::= \; x \; \mid \; o((x_1, \ldots, x_{n_1}).t_1, \ldots, (x'_1, \ldots, x'_{n_k}).t_k)$$

where o is an operator of arity $\langle n_1,\ldots,n_k\rangle$. Obvious definitions for free/bound variables and α-equivalence apply to these terms.

Analogous to the case of the λ-calculus, for any binding signature, there is a presheaf of terms (up to α-equivalence) $\mathrm{TV}_\alpha \in \mathcal{F}$ given by

$$\begin{array}{l} \mathrm{TV}_\alpha(n) \stackrel{\mathrm{def}}{=} \{\, [t]_\alpha \mid \mathrm{FV}(t) \subseteq \{\, x_1,\ldots,x_n \,\}\,\} \\ \mathrm{TV}_\alpha(\rho) : t \longmapsto t\{x_{\rho 1}/x_1,\ldots,x_{\rho m}/x_m\} \end{array}$$

for every $n \in \mathbb{F}$ and $\rho : m \rightarrow n$ in $\mathbb{F}$.

Abstract syntax with variable binding. To give the abstract characterisation of TV_α we consider algebras of binding signatures. Recalling that an operator of arity $\langle n_i\rangle_{1\le i\le k}$ binds n_i variables in the i-th argument and that δ is a type constructor for context extension it is natural to interpret an operation of arity $\langle n_1,\ldots,n_k\rangle$ on a presheaf $A \in \mathcal{F}$ as a map $\delta^{n_1}(A)\times\ldots\times\delta^{n_k}(A) \longrightarrow A$, and hence to define a Σ-algebra over a presheaf $A\in\mathcal{F}$ as a map

$$\coprod_{\substack{o\in O\\ a(o)=\langle n_i\rangle_{1\le i\le k}}} \delta^{n_1}(A)\times\ldots\times\delta^{n_k}(A) \longrightarrow A \quad .$$

Thus, to a binding signature $\Sigma = (O,a)$ we associate the functor $\Sigma : \mathcal{F} \longrightarrow \mathcal{F}$ given by

$$\Sigma(X) \stackrel{\mathrm{def}}{=} \coprod_{\substack{o\in O\\ a(o)=\langle n_i\rangle_{1\le i\le k}}} \prod_{1\le i\le k} \delta^{n_i}(X) \quad , \tag{10}$$

and define the category of algebras associated to the signature Σ as the category Σ-**Alg**, with objects given by **algebras** $h : \Sigma A \longrightarrow A$ and morphisms $f : (A,h) \longrightarrow (A',h')$ given by maps $f : A \longrightarrow A'$ that are *homomorphic* in the sense that $f\circ h = h'\circ\Sigma(f)$. This approach fits into the paradigm of categorical algebra [4, 5]. However we remark that the general theory allows for the treatment of more sophisticated notions of signature (incorporating equational theories and thus enabling us to deal with notions such as λ-models) which will not be considered in this extended abstract.

As is well-known (see *e.g.* [4]), the forgetful functor $\Sigma\text{-Alg} \longrightarrow \mathcal{F} : (A,h) \longmapsto A$ has a left adjoint providing the free Σ-algebra on a presheaf; which, for a presheaf $X\in\mathcal{F}$, is an initial $(X+\Sigma)$-algebra.

The presheaf of terms TV_α of a binding signature $\Sigma = (O,a)$ has a **syntactic algebra** structure $[\tau^{(o)}]_{o\in O}$ given, at stage n, by the mapping sending the tuple of terms $\langle t_i\rangle_{1\le i\le k}$ to the term

$$\begin{array}{l} o(\ (x_{new(1)},\ldots,x_{new(n_i)}).t_i \\ \qquad\qquad \{x_1/x_{old(1)},\ldots,x_n/x_{old(n)}\}\ )_{1\le i\le k}\ , \end{array}$$

where $new(j) = \mathsf{new}_{n+j-1}$ ($1\le j\le n_i$) and $old(j) = \mathsf{old}_n(j)$ ($1\le j\le n$).

Theorem 2.1 The presheaf of terms TV_α associated to a binding signature Σ (equipped with the syntactic algebra structure) is a free Σ-algebra on the presheaf of variables V.

We show how the above general result may be used to implement abstract syntax. To this end recall that the underlying presheaf of a free Σ-algebra on a presheaf X may be computed as the union of the chain

$$0 \subseteq X+\Sigma(0) \subseteq X+\Sigma(X+\Sigma(0)) \subseteq \cdots$$

obtained by iterating the functor $X+\Sigma$ on the empty presheaf 0.

In the particular case of the free algebra Λ on the presheaf of variables V for the signature of the λ-calculus $\Sigma_\lambda(X) = \delta X + X\times X$ this calculation amounts to the following inductive definitions: $\Lambda(n) = \{\ t \mid n\vdash t\ \}$ ($n\in\mathbb{F}$) where

$$\frac{1\le i\le n}{n\vdash \mathsf{var}(i)} \qquad \frac{n+1\vdash t}{n\vdash \mathsf{lam}(t)} \qquad \frac{n\vdash t_1 \qquad n\vdash t_2}{n\vdash \mathsf{app}(t_1,t_2)}$$

and, for $\rho : m\rightarrow n$ in $\mathbb{F}$,

$$\begin{array}{lll} \Lambda(\rho)(t) = \mathsf{case}\ t\ \mathsf{of} & & \\ \quad \mathsf{var}(i) & \Rightarrow & \mathsf{var}(\ \rho i\ ) \\ \quad \mathsf{lam}(t') & \Rightarrow & \mathsf{lam}(\ \Lambda(\rho+\mathrm{id}_1)(t)\ ) \\ \quad \mathsf{app}(t1,t2) & \Rightarrow & \mathsf{app}(\ \Lambda(\rho)(t_1),\Lambda(\rho)(t_2)\ )\ . \end{array}$$

This abstract view yields particular implementations of λ-terms according to different choices of the coproduct structure on the category $\mathbb{F}$. For instance, if $\mathsf{old}_n(i) = i+1$ ($1\le i\le n$) and $\mathsf{new}_n = 1$ ($n\in\mathbb{N}$) then the presheaf Λ implements λ-terms by the method of De Bruijn indices; as one can notice, for example, from the fact that, for $\rho : m\rightarrow n$ in $\mathbb{F}$,

$$\Lambda(\rho)(\mathsf{lam}(\mathsf{var}\ i)) \quad = \quad \begin{cases} \mathsf{lam}(\mathsf{var}\ 1) & \text{, if } i=1\ . \\ \mathsf{lam}(\mathsf{var}\ \rho i) & \text{, otherwise}\ . \end{cases}$$

(The reader may wish to consider examples involving swap and up.) Of course, the implementation of λ-terms by the method of De Bruijn levels is obtained by choosing $\mathsf{old}_n(i) = i$ ($1\le i\le n$) and $\mathsf{new}_n = n+1$ ($n\in\mathbb{N}$).

A glance at initial algebra semantics. To consider interpretations of the λ-calculus let $fold : D^D \lhd D : unfold$ be a retraction in a cartesian closed category $\mathcal{C}$ [33].

The clone of operations $\langle D,D\rangle \in \mathcal{F}$ (see (6)) admits a canonical interpretation of variables $\iota : \mathrm{V} \longrightarrow \langle D,D\rangle$ given by

$$\begin{array}{rcl} n & \longrightarrow & \mathcal{C}(D^n,D) \\ i & \longmapsto & \pi_i \qquad , \end{array}$$

and may be equipped with an algebra structure $\delta\langle D, D\rangle + \langle D, D\rangle \times \langle D, D\rangle \longrightarrow \langle D, D\rangle$ as follows

$$\begin{array}{ccc} \mathcal{C}(D^{n+1}, D) & \longrightarrow & \mathcal{C}(D^n, D) \\ f & \longmapsto & \mathit{fold} \circ \boldsymbol{\lambda}(f) \end{array} \quad ,$$

$$\begin{array}{ccc} \mathcal{C}(D^n, D) \times \mathcal{C}(D^n, D) & \longrightarrow & \mathcal{C}(D^n, D) \\ (f, x) & \longmapsto & \mathrm{ev} \circ \langle \mathit{unfold} \circ f, x\rangle \end{array} .$$

As Λ is the free Σ_λ-algebra on V, it follows from Theorem 2.1 that $\iota : \mathrm{V} \longrightarrow \langle D, D\rangle$ has a unique *homomorphic* extension $[\![-]\!] : \Lambda \longrightarrow \langle D, D\rangle$ characterised as

$$\begin{tikzcd} \mathrm{V} + \delta\Lambda + \Lambda \times \Lambda \arrow[r, "{[\mathsf{var},\mathsf{lam},\mathsf{app}]}", "\cong"'] \arrow[d, "\mathrm{V}+\delta[\![-]\!]+[\![-]\!]\times[\![-]\!]"'] & \Lambda \arrow[d, "{[\![-]\!]}"] \\ \mathrm{V} + \delta\langle D, D\rangle + \langle D, D\rangle \times \langle D, D\rangle \arrow[r] & \langle D, D\rangle \end{tikzcd} ,$$

and which can be easily shown to be the traditional *compositional* interpretation function $\Lambda(n) \longrightarrow \mathcal{C}(D^n, D)$ $(n \in \mathbb{N})$ of λ-terms [33].

3. Substitution

A program for substitution. To motivate the more abstract development to follow, and to link our approach to programming, we start by writing a (categorical) recursive program for substituting λ-terms in the type theory of $\mathcal{F}$.

Let Λ (the presheaf of λ-terms) be the free algebra on the presheaf of variables V for the signature of the λ-calculus Σ_λ (see Theorem 2.1 and the discussion after it). We aim at defining an operation

$$\sigma : \delta\Lambda \times \Lambda \longrightarrow \Lambda$$

that, roughly speaking, given a pair (t, u) consisting of a term t with a *new* variable (*i.e.*, a term in an extended context) and a term u, substitutes u for the *new* variable in t. Using that

$$\delta\Lambda = \delta\mathrm{V} + \delta\delta\Lambda + \delta\Lambda \times \delta\Lambda$$

(since $\Lambda = \mu X.\mathrm{V} + \delta X + X \times X$ and δ preserves sums and products) we can define a recursive program for σ by case analysis on its first argument. The definition is as follows

$$\begin{array}{lll} \multicolumn{3}{l}{\sigma(t, u) = \mathsf{case}\ t\ \mathsf{of}} \\ \quad x : \delta\mathrm{V} & \Rightarrow & \beta(x, u) \\ \quad t' : \delta\delta\Lambda & \Rightarrow & \mathsf{lam}(\ \delta\sigma(\mathsf{swap}\ t', \mathsf{up}\ u)\ ) \\ \quad t_1, t_2 : \delta\Lambda & \Rightarrow & \mathsf{app}(\ \sigma(t_1, u), \sigma(t_2, u)\ ) \end{array} \tag{11}$$

where, using that $\delta\mathrm{V} = \mathrm{V} + 1$ (see (9)), the *basic substitution* $\beta : \delta\mathrm{V} \times \Lambda \longrightarrow \Lambda$ is defined as

$$\begin{array}{lllll} \beta(x, u) = \mathsf{case}\ x\ \mathsf{of} & \mathsf{old}(y) & \Rightarrow & \mathsf{var}(y) & \\ & \mathsf{new} & \Rightarrow & u & . \end{array} \tag{12}$$

Put in elementary terms, the natural family $\sigma_n : \Lambda(n+1) \times \Lambda(n) \longrightarrow \Lambda(n)$ $(n \in \mathbb{F})$ is given by

$$\begin{array}{lllll} \multicolumn{5}{l}{\sigma_n(t, u) = \mathsf{case}\ t\ \mathsf{of}} \\ \quad \mathsf{var}(i) & \Rightarrow & \mathsf{case}\ i\ \mathsf{of}\ \ \mathsf{old}_n(j) & \Rightarrow & \mathsf{var}(j) \\ & & \qquad\qquad\ \ \mathsf{new}_n & \Rightarrow & u \\ \quad \mathsf{lam}(t') & \Rightarrow & \multicolumn{3}{l}{\mathsf{lam}(\ \sigma_{n+1}(\mathsf{swap}_n\ t', \mathsf{up}_n\ u)\ )} \\ \quad \mathsf{app}(t_1, t_2) & \Rightarrow & \multicolumn{3}{l}{\mathsf{app}(\ \sigma_n(t_1, u), \sigma_n(t_2, u)\ )} \end{array} .$$

Note that the substitution $\sigma_n(\mathsf{lam}(t'), u)$ proceeds by *swapping* (the indices for) the binding and the new variables in $t' \in \Lambda(n+2)$, and by subsequently using the substitution operation σ_{n+1} with the *weakened* argument $\mathsf{up}_n(u)$ (where indices are shifted appropriately).

Interestingly, using that $\delta(X) = X^{\mathrm{V}}$ (see (8)), we have that the definition (11) corresponds to the following one

$$\begin{array}{lllll} \multicolumn{5}{l}{\sigma(t, u) = \mathsf{case}\ t\ \mathsf{of}} \\ \quad x : \mathrm{V} + 1 & \Rightarrow & \mathsf{case}\ x\ \mathsf{of}\ \ \mathsf{old}(y) & \Rightarrow & \mathsf{var}(y) \\ & & \qquad\qquad\ \ \mathsf{new} & \Rightarrow & u \\ \quad t' : (\Lambda^{\mathrm{V}})^{\mathrm{V}} & \Rightarrow & \multicolumn{3}{l}{\mathsf{lam}(\ \lambda y : \mathrm{V}.\, \sigma(\ \lambda x : \mathrm{V}.\, t'xy\ ,\ u\ )\ )} \\ \quad t_1, t_2 : \Lambda^{\mathrm{V}} & \Rightarrow & \multicolumn{3}{l}{\mathsf{app}(\ \sigma(t_1, u), \sigma(t_2, u)\ )} \end{array}$$

which resembles the traditional definition of substitution. For instance, note that by construction the equality

$$\begin{array}{l} t' : (\Lambda^{\mathrm{V}})^{\mathrm{V}} \vdash \sigma(\ \lambda x : \mathrm{V}.\, \mathsf{lam}(\ \lambda y : \mathrm{V}.\, t'xy\ )\ ,\ u\ ) \\ \qquad\qquad\qquad = \mathsf{lam}(\ \lambda y : \mathrm{V}.\, \sigma(\ \lambda x : \mathrm{V}.\, t'xy\ ,\ u\ )\ ) \end{array}$$

holds.

As we show below, the above definition of substitution amounts to a definition by *structural recursion*. Hence in our approach, unlike in the traditional one (see, *e.g.*, [3]), the well-definedness of the substitution operation need not be established separately.

Definition of substitution by structural recursion. For a binding signature Σ, let $\phi_X : \Sigma\mathrm{T}X \longrightarrow \mathrm{T}X$ be a free Σ-algebra over the presheaf X.

To define a substitution operation

$$\sigma : \delta(\mathrm{TV}) \times \mathrm{TV} \longrightarrow \mathrm{TV}$$

for the presheaf of terms TV by structural recursion we proceed as follows. First, with the aid of the following *exchange* natural isomorphisms

$$\delta^n(\delta X) \cong (X^{\mathrm{V}})^{\mathrm{V}\times\cdots\times\mathrm{V}} \cong (X^{\mathrm{V}\times\cdots\times\mathrm{V}})^{\mathrm{V}} \cong \delta(\delta^n X)$$

we define a distributive law

$$\psi : \Sigma\delta \overset{.}{\cong} \delta\Sigma \tag{13}$$

of the endofunctor Σ over the monad δ in an obvious way. Second, we observe that this construction yields a natural isomorphism $(\delta\mathrm{V} + \Sigma) \circ \delta \overset{.}{\cong} \delta \circ (\mathrm{V} + \Sigma)$; from which, by the *uniformity property* of the fixed-point operator (see [12, Theorem 7.3.12 (6)]), it follows that the Σ-algebra

$$\Sigma\delta(\mathrm{TV}) \overset{\psi_{\mathrm{TV}}}{\cong} \delta\Sigma(\mathrm{TV}) \overset{\delta\phi_{\mathrm{V}}}{\longrightarrow} \delta(\mathrm{TV})$$

is free over $\delta\mathrm{V}$. Finally, using that Σ has a *strength* $str : \Sigma(A) \times X \longrightarrow \Sigma(A \times X)$, we let σ be the unique homomorphic extension of a basic substitution β as in (12). That is, we define σ to be the unique map such that the diagram

$$\begin{tikzcd}
\Sigma(\delta(\mathrm{TV})) \times \mathrm{TV} \arrow[r, "str"] \arrow[d, "\psi_{\mathrm{TV}} \times \mathrm{id}"', "\cong"] & \Sigma(\delta(\mathrm{TV}) \times \mathrm{TV}) \arrow[r, "\Sigma\sigma"] & \Sigma\mathrm{TV} \arrow[dd, "\phi_{\mathrm{V}}"] \\
\delta\Sigma(\mathrm{TV}) \times \mathrm{TV} \arrow[d, "\delta\phi_{\mathrm{V}} \times \mathrm{id}"'] & & \\
\delta(\mathrm{TV}) \times \mathrm{TV} \arrow[rr, dashed, "\sigma"] & & \mathrm{TV} \\
\delta\mathrm{V} \times \mathrm{TV} \arrow[u, "\delta\eta_{\mathrm{V}} \times \mathrm{id}"] \arrow[urr, "\beta"'] & &
\end{tikzcd}$$

commutes, where the map $\eta_{\mathrm{V}} : \mathrm{V} \longrightarrow \mathrm{TV}$, coercing variables into terms, is the universal arrow associated to the free algebra TV.

Substitution algebras. We show that the operations $\sigma : \delta TV \times TV \longrightarrow TV$ (obtained as above) and $\ulcorner\eta_{\mathrm{V}}\urcorner = (\delta\eta_V) \circ \mathsf{new} : 1 \longrightarrow \delta TV$ obey the laws of substitution. To this end we introduce an axiomatisation of single-variable substitution, whose justification is provided by Theorem 3.3 below.

Definition 3.1 A **substitution algebra** $X = (X, \varsigma, \nu)$ consists of a presheaf $X \in \mathcal{F}$ equipped with two operations $\varsigma : \delta X \times X \longrightarrow X$ (a *substitution*) and $\nu : 1 \longrightarrow \delta X$ (a *generic new variable*) such that:

1. $u : X \ \vdash \ \varsigma(\nu, u) = u$.

2. $t, u : X \ \vdash \ \varsigma(\mathsf{up}(t), u) = t$.

3. $t : \delta^2 X \ \vdash \ \delta\varsigma(t, \nu) = \mathsf{contract}(t)$.

4. $t : \delta^2 X \ , \ u : \delta X \ , \ v : X \ \vdash$
$\varsigma(\delta\varsigma(t, u), v) = \varsigma(\delta\varsigma(\mathsf{swap}(t), \mathsf{up}(v)), \varsigma(u, v))$.

The axioms have the following intuitive reading. Axiom 1 says that substituting for the generic new variable has the expected result. Axiom 2 says that substituting for a variable that is not in a term does not affect the term. Axiom 3 says that substituting the generic new variable in a term is like performing a contraction. Axiom 4 is a version of the substitution lemma.

Theorem 3.2 For every binding signature Σ, the structure $(\mathrm{TV}, \sigma, \ulcorner\eta_{\mathrm{V}}\urcorner)$ is a substitution algebra.

Clones. Substitution algebras axiomatise single-variable substitution. Here we show that they are equivalent to the following axiomatisation of simultaneous substitution by abstract clones (familiar, in the concrete case, from universal algebra—*cf.* [8, page 132]).

An (*abstract*) **clone** $X = (X, \mu, \iota)$ consists of a family $X = \{X_n\}_{n\in\mathbb{N}}$ of sets, a family $\iota = \{\iota_i^{(n)} \in X_n \mid 1 \le i \le n\}_{n\in\mathbb{N}}$ of distinguished elements, and a family

$$\mu = \{\mu_m^{(n)} : X_n \times (X_m)^n \longrightarrow X_m\}_{n,m\in\mathbb{N}}$$

of operations such that, for every element t of X_n, every n-tuple $\vec{u} = (u_1, \ldots, u_n)$ of elements of X_m, and every m-tuple $\vec{v}$ of elements of X_l, the following three axioms hold:

$$\begin{aligned} &\mu_m(\iota_i; \vec{u}) = u_i \quad , \quad \mu_n(t; \iota_1, \ldots, \iota_n) = t \quad , \\ &\mu_l(\mu_m(t; \vec{u}); \vec{v}) = \mu_l(t; \mu_l(u_1; \vec{v}), \ldots, \mu_l(u_n; \vec{v})) \end{aligned} \tag{14}$$

An example of a clone is given by taking: for X_n the set $\mathrm{TV}(n)$ of terms (in a context of n variables) with respect to a given signature; for $\iota_i^{(n)}$ the variable x_i in $\mathrm{TV}(n)$; and for $\mu_m^{(n)}$ the simultaneous substitution of terms

$$\sigma_m^{(n)} : \mathrm{TV}(n) \times \mathrm{TV}(m)^n \longrightarrow \mathrm{TV}(m) \quad .$$

We write $\sigma_m^{(n)}(t; \vec{u})$ in infix notation and with no indices as $t\,[\vec{u}]$. Then the three axioms in (14) amount to the following familiar properties of substitution:

$$\begin{aligned} &x_i[\vec{u}] = u_i \quad , \qquad t\,[x_1, \ldots, x_n] = t \quad , \\ &t\,[\vec{u}][\vec{v}] = t\,[u_1[\vec{v}], \ldots, u_n[\vec{v}]] \quad . \end{aligned}$$

(The last identity is the *syntactic substitution lemma*—*cf.* [3, page 27].)

For every object C of a cartesian category $\mathcal{C}$, the clone of operations $\langle C, C\rangle$ on C as defined after (6) yields another example of an (abstract) clone: $X_n = \mathcal{C}(C^n, C)$ is then the set of operations on C of arity n, ι_i is the i-th projection π_i, and μ_m is given by composition.

We remark that, just like clones of operations, abstract clones $X = (X, \mu, \iota)$ are presheaves, with $X(n) \overset{\mathrm{def}}{=} X_n$ and with action on renamings $\rho : n \to m$ given by $X(\rho)(t) \overset{\mathrm{def}}{=} \mu_m(t; \iota_{\rho 1}, \ldots, \iota_{\rho n})$.

Both clones and substitution algebras, together with the evident homomorphisms, form categories.

Theorem 3.3 The categories of substitution algebras and of clones are equivalent.

Monoids in $\mathcal{F}$. There are several equivalent categorical formulations of clones; *e.g.*, as *Lawvere theories*, as *finitary monads*, or as *one-object cartesian multicategories*.

(See [13] for an elementary presentation of the connection between substitution and Lawvere theories.) Here we recall that clones (and thus also substitution algebras) are equivalent to monoids in $\mathcal{F}$ with respect to a suitable monoidal structure. This compact and abstract presentation is important for reasoning about the structure of substitution and its interplay with Σ-algebras. In particular, it allows us to define the simultaneous substitution of terms by structural recursion.

The monoidal structure we consider is given by a (highly non-symmetric) tensor '$\bullet$' of presheaves with unit V. For every presheaf Y, the functor $\_ \bullet Y$ is left adjoint to the endofunctor $\langle Y, \_\rangle$ on $\mathcal{F}$:

$$\begin{array}{c} X \bullet Y \longrightarrow Z \\ \hline\hline X \longrightarrow \langle Y, Z\rangle \end{array} \tag{15}$$

An explicit description of this tensor is given by the following *coend* formula: for presheaves X and Y,

$$(X \bullet Y)(m) \overset{\text{def}}{=} (\textstyle\coprod_{n\in\mathbb{N}} X(n) \times Y(m)^n)_{/\approx}$$
$$= \{(t;\vec{u}) \mid n \in \mathbb{N},\, t \in X(n),\, \vec{u} \in Y(m)^n\}_{/\approx}$$

where $\approx$ is the equivalence relation generated by

$$(t; u_1, \ldots, u_n) \sim (t'; u'_1, \ldots, u'_{n'})$$

iff there exists a map $\rho : n \to n'$ such that $X(\rho)(t) = t'$ and $u_i = u'_{\rho i}$. For example, for $X = Y = \Lambda$,

$$\begin{aligned} &(x_1x_2; u_1, u_2) \sim (x_2x_3; u, u_1, u_2) \\ &\quad \sim (x_1x_3; u_1, u, u_2) \sim (x_2x_1; u_2, u_1) \end{aligned}$$

and $(x_1x_1; u) \sim (x_1x_2; u, u)$, for all u_1, u_2, and u in $\Lambda(m)$.

A **monoid** $X = (X, \mu, \iota)$ in $\mathcal{F} = (\mathcal{F}, \bullet, \mathrm{V})$ consists of a presheaf X, a *unit* $\iota : \mathrm{V} \longrightarrow X$, and a *multiplication* $\mu : X \bullet X \longrightarrow X$ such that the following diagrams commute.

$$\begin{tikzcd}
\mathrm{V} \bullet X \arrow[r, "\iota\bullet\mathrm{id}"] \arrow[dr, "\cong"'] & X \bullet X \arrow[d, "\mu"] & X \bullet \mathrm{V} \arrow[l, "\mathrm{id}\bullet\iota"'] \arrow[dl, "\cong"] \\
& X &
\end{tikzcd}$$

$$\begin{tikzcd}
(X \bullet X) \bullet X \arrow[rr, "\cong"] \arrow[d, "\mu\bullet\mathrm{id}"'] & & X \bullet (X \bullet X) \arrow[d, "\mathrm{id}\bullet\mu"] \\
X \bullet X \arrow[r, "\mu"'] & X & X \bullet X \arrow[l, "\mu"]
\end{tikzcd}$$

The three isomorphisms in the above diagrams act as follows:

$$(i; \vec{u}) \longmapsto u_i \quad , \qquad (t; 1, \ldots, n) \longmapsto t \quad ,$$
$$((t; \vec{u}); \vec{v}) \longmapsto (t; (u_1; \vec{v}), \ldots, (u_n; \vec{v})) \quad .$$

(Note that every equivalence class of $X \bullet V$ contains a tuple of the form $(t; 1, \ldots, n)$.)

Monoids in $\mathcal{F}$, with maps $f : (X, \mu, \iota) \longrightarrow (X', \mu', \iota')$ given by morphisms $f : X \longrightarrow X'$ such that $f \circ \iota = \iota'$ and $f \circ \mu = \mu' \circ (f \bullet f)$, form a category **Mon**($\mathcal{F}$) with initial object $\mathrm{V} = (\mathrm{V}, \mathrm{V} \bullet \mathrm{V} \xrightarrow{\cong} \mathrm{V}, \mathrm{id}_{\mathrm{V}})$. Similarly, one defines the category **Mon**($\mathcal{C}$) of monoids in any monoidal category $\mathcal{C} = (\mathcal{C}, \otimes, I)$.

Proposition 3.4 The categories of clones and of monoids in $\mathcal{F} = (\mathcal{F}, \bullet, \mathrm{V})$ are equivalent.

A program for simultaneous substitution. Let F be an endofunctor on a monoidal closed category $\mathcal{C} = (\mathcal{C}, \otimes, I)$. If F has a strength $st_{X,Y} : F(X) \otimes Y \longrightarrow F(X \otimes Y)$, then its free algebras are also *parametrically* free with respect to $\otimes$. In particular, if F has a free algebra $\phi_I : FTI \longrightarrow TI$ over I, then there is a unique map $\sigma : TI \otimes TI \longrightarrow TI$ making the following diagram commutative

$$\begin{tikzcd}
F(TI) \otimes TI \arrow[r, "st_{TI,TI}"] \arrow[d, "\phi_I\otimes\mathrm{id}"'] & F(TI \otimes TI) \arrow[r, "F\sigma"] & FTI \arrow[d, "\phi_I"] \\
TI \otimes TI \arrow[rr, dashed, "\sigma"'] & & TI \\
I \otimes TI \arrow[u, "\eta_I\otimes\mathrm{id}"] \arrow[urr, "\cong"'] & &
\end{tikzcd}$$

where η_I is the universal arrow corresponding to the free algebra TI. That is, σ is the unique (parametric) homomorphic extension of $I \otimes TI \xrightarrow{\cong} TI$.

Proposition 3.5 Under the above hypotheses, $TI = (TI, \sigma, \eta_I)$ is a monoid in $\mathcal{C}$.

The adjunction (15) shows that $\mathcal{F} = (\mathcal{F}, \bullet, \mathrm{V})$ is closed. Moreover, endofunctors corresponding to *first-order* signatures are strong with respect to $\bullet$ in the obvious way; for instance, for binary operations, the strength maps $(t_1, t_2; \vec{u})$ to $((t_1; \vec{u}), (t_2; \vec{u}))$. Moving on to binding signatures, one has to prove δ strong. This holds in the category $\mathrm{V}/\mathcal{F}$ of pointed presheaves; indeed, δ restricts to an endofunctor on $\mathrm{V}/\mathcal{F}$ (via $\mathsf{up} : \mathrm{Id} \longrightarrow \delta$) and it has a strength $st_{X,Y}(m) : (\delta X \bullet Y)(m) \longrightarrow \delta(X \bullet Y)(m)$ which acts as follows,

$$(t; \vec{u}) \longmapsto (t; [\mathsf{up}_m(\vec{u}), \iota_{m+1}(\mathsf{new}_m)]) \quad ,$$

where $\iota : \mathrm{V} \longrightarrow Y$ is the point of Y. In particular, the free Σ-algebra TV over V has a point $\eta_V \overset{\text{def}}{=} \mathsf{var} : \mathrm{V} \longrightarrow \mathrm{TV}$ given by the insertion of the variables (mapping i to x_i). The above proposition then yields the following:

Corollary 3.6 Let Σ be a binding signature and let TV be its free algebra over V. Then $(\mathrm{TV}, \sigma, \eta_{\mathrm{V}})$ is a monoid in $\mathcal{F}$, where $\sigma : \mathrm{TV} \bullet \mathrm{TV} \longrightarrow \mathrm{TV}$ is the unique homomorphic extension of $\mathrm{V} \bullet \mathrm{TV} \xrightarrow{\cong} \mathrm{TV}$.

The above σ is defined by *structural recursion*. For instance, for the λ-calculus $\sigma : \Lambda \bullet \Lambda \longrightarrow \Lambda$ is defined as follows: for all $(t; \vec{u})$ in $(\Lambda \bullet \Lambda)(m)$,

$$\begin{array}{lll} \multicolumn{3}{l}{\sigma_m(t;\vec{u}) = \mathsf{case}\ t\ \mathsf{of}} \\ \quad \mathsf{var}(i) & \Rightarrow & u_i \\ \quad \mathsf{lam}(t') & \Rightarrow & \mathsf{lam}(\ \sigma_{m+1}(t'; [\mathsf{up}_m(\vec{u}), \mathsf{var}(\mathsf{new}_m)]\ ) \\ \quad \mathsf{app}(t_1, t_2) & \Rightarrow & \mathsf{app}(\ \sigma_m(t_1; \vec{u}), \sigma_m(t_2; \vec{u})\ ) \end{array}$$

Taking $\mathsf{old}_n(i) = i + 1$ $(1 \leq i \leq n)$ and $\mathsf{new}_n = 1$ we get De Bruijn's definition of substitution for indices, while $\mathsf{old}_n(i) = i$ $(1 \leq i \leq n)$ and $\mathsf{new}_n = n + 1$ give the definition for levels. (*Cf.* [9, 34].)

4. Initial algebra semantics

The key to initial algebra semantics for syntax with variable binding is the definition of a category of *binding algebras* (consisting of compatible algebra and substitution structures) in which the syntactic algebra equipped with the usual substitution operation is characterised as an initial object. In this section we consider two equivalent formulations of the concept of binding algebra (*viz.* as Σ-monoids and as Σ-substitution algebras) and establish the required property (*i.e.*, that the free Σ-algebra TV with the monoid structure given by Corollary 3.6 is an initial object in the category of Σ-monoids).

For any strong endofunctor F on $\mathcal{C} = (\mathcal{C}, \otimes, I)$, let F-**Mon**$(\mathcal{C})$ be the category with objects given by F-**monoids**, *i.e.*, quadruples $X = (X, \mu, \iota, h)$, where (X, μ, ι) is a monoid in $\mathcal{C}$ and (X, h) is an F-algebra such that

$$\begin{tikzcd} F(X) \otimes X \arrow[r, "st_{X,X}"] \arrow[d, "h\otimes\mathrm{id}"'] & F(X \otimes X) \arrow[r, "F\mu"] & FX \arrow[d, "h"] \\ X \otimes X \arrow[rr, "\mu"'] & & X \end{tikzcd} \tag{16}$$

commutes; morphisms are maps of $\mathcal{C}$ which are both F-algebra and monoid homomorphisms.

Theorem 4.1 Let F, TI, σ, η_I, and ϕ_I be as in Proposition 3.5. Then $TI = (TI, \sigma, \eta_I, \phi_I)$ is an initial F-monoid.

This result, together with Corollary 3.6, ensures that we can take Σ-monoids as our category of models for a binding signature Σ; the type of terms TV is then the initial such model. We remark that Σ-monoids and the commutativity of diagram (16) specialise, respectively, to the models and the *uniformity condition* given in [2].

We call the unique morphism $\mathrm{TV} \longrightarrow \mathfrak{M}$ from the initial Σ-monoid TV to a Σ-monoid $\mathfrak{M}$ the *initial algebra semantics* corresponding to $\mathfrak{M}$. By definition of morphism in Σ-**Mon**$(\mathcal{F})$, the initial algebra semantics is *compositional*, it *preserves the variables* and it always satisfies the *semantic substitution lemma* [35, Lemma 4.6].

Consider, for example, the λ-calculus. For $\mathfrak{M} = (\mathfrak{M}, \mu, \iota, [abs, \cdot])$ to be a Σ_λ-monoid the following should hold: for all d in $\mathfrak{M}(n)$ and $\vec{e}$ in $\mathfrak{M}(m)^n$,

$$\begin{aligned} \mu_m(abs(d); \vec{e}) &= abs(\mu_{m+1}(d; [\mathsf{up}_m(\vec{e}), \iota_{m+1}(\mathsf{new}_m)])) \\ \mu_m(d_1 \cdot d_2; \vec{e}) &= \mu_m(d_1; \vec{e}) \cdot \mu_m(d_2; \vec{e}) \quad . \end{aligned}$$

The initial algebra semantics of the λ-calculus with respect to such a model $\mathfrak{M}$ is the unique morphism $[\![\_]\!] : \Lambda \longrightarrow \mathfrak{M}$ such that:

$$\begin{aligned} & [\![\lambda x.t]\!] = abs[\![t]\!] \quad , \qquad [\![tu]\!] = [\![t]\!] \cdot [\![u]\!] \quad , \\ & [\![x_i]\!] = [\![\iota_n(i)]\!] \quad , \\ & [\![t\,[u_1, \ldots, u_n]]\!] = \mu_m([\![t]\!]; [\![u_1]\!], \ldots, [\![u_n]\!]) \quad . \end{aligned}$$

In particular, one can easily verify that the model $\langle D, D \rangle$ for the λ-calculus defined at the end of Section 2 is a Σ_λ-monoid and that the corresponding initial algebra semantics is the desired one.

Using substitution algebras rather than monoids we define an equivalent category of Σ-**substitution algebras** as follows: objects are quadruples $X = (X, \varsigma, \nu, h)$ consisting of a substitution algebra (X, ς, ν) and a Σ-algebra (X, h) that are compatible in the sense that the diagram

$$\begin{tikzcd} \Sigma(\delta(X)) \times X \arrow[r, "str_{\delta X,X}"] \arrow[d, "\psi_X \times \mathrm{id}"', "\cong"] & \Sigma(\delta(X) \times X) \arrow[r, "\Sigma\varsigma"] & \Sigma X \arrow[dd, "h"] \\ \delta\Sigma(X) \times X \arrow[d, "\delta h \times \mathrm{id}"'] & & \\ \delta(X) \times X \arrow[rr, "\varsigma"'] & & X \end{tikzcd}$$

commutes (where ψ is the distributive law of (13)); morphisms are maps in $\mathcal{F}$ that are both Σ-algebra and substitution-algebra homomorphisms.

Theorem 4.2 The categories of Σ-substitution algebras and of Σ-monoids in $\mathcal{F}$ are equivalent.

Acknowledgements. We are grateful to John Power especially for his suggestion of working with algebras over presheaves.

References

[1] P. Aczel. A general Church-Rosser theorem. Unpublished manuscript, 1978.

[2] P. Aczel. Frege structures and the notions of proposition, truth and set. In J. Barwise et al., editors, *The Kleene Symposium*, pages 31–60. North-Holland, 1980.

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

[4] M. Barr and C. Wells. *Toposes, Triples and Theories*. Springer-Verlag, 1985.

[5] F. Borceux. *Handbook of Categorial Algebra*. Cambridge University Press, Cambridge, 1994.

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

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

[8] P. Cohn. *Universal Algebra*. Harper & Row, 1965.

[9] N. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–391, 1972.

[10] J. Despeyroux, F. Pfenning, and C. Schürmann. Primitive recursion for higher-order abstract syntax. In R. Hindley, editor, *Proc. TLCA'97 Conf.*, volume 1210 of *LNCS*. Springer-Verlag, 1997.

[11] M. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus. In *Proc. 11th LICS Conf.* IEEE, Computer Society Press, 1996.

[12] M. P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Distinguished Dissertations Series. Cambridge University Press, 1996.

[13] J. Goguen. What is unification? In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Equations, Vol. 1, Algebraic Techniques*. Academic Press, 1989.

[14] J. Goguen, J. Thatcher, and E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology*, volume IV, pages 80–149. Prentice Hall, 1978.

[15] M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF*, volume 78 of *LNCS*. Springer Verlag, 1979.

[16] A. Joyal. Foncteurs analytiques et espèces de structures. In *Combinatoire énumerative*, volume 1234 of *LNM*, pages 126–159. Springer Verlag, 1987.

[17] W. Kahl. Relational treatment of term graphs with bound variables. *Logic Journal of the IGPL*, 6(2):259–303, 1998.

[18] G. Kelly. An abstract approach to coherence. In *Coherence in Categories*, volume 281 of *LNM*, pages 106–147. Springer-Verlag, 1972.

[19] G. Kelly. Clubs and data-type constructors. In M. P. Fourman et al., editors, *Applications of Categories in Computer Science*, volume 177 of *London Mathematical Society Lecture Note Series*, pages 163–190. CUP, 1992.

[20] J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, University of Utrecht, 1980. Published as Mathematical Center Tract 129.

[21] P. Landin. A λ-calculus approach. In *Advances in Programming and Non-Numerical Computation*, pages 97–141. Pergammon Press, 1966.

[22] J.-L. Loday, J. Stasheff, and A. Voronov, editors. *Operads: Proceedings of Renaissance Conference*. Am. Math. Soc., 1996.

[23] S. Mac Lane and I. Moerdijk. *Sheaves in geometry and logic: A First Introduction to Topos Theory*. Springer-Verlag, 1992.

[24] J. McCarthy. Towards a mathematical science of computation. In *IFIP Congress 1962*. North-Holland, 1963.

[25] R. McDowell and D. Miller. A logic for reasoning with higher-order abstract syntax. In *Proc. 12th LICS Conf.*, pages 434–445. IEEE Computer Society Press, 1997.

[26] D. Miller. An extension to ML to handle bound variables in data structures: Preliminary report. In *Informal Proceedings of the Logical Frameworks BRA Workshop*, June 1990. Available as UPenn CIS technical report MS-CIS-90-59.

[27] D. Miller and G. Nadathur. A logic programming approach to manipulating formulas and programs. In S. Haridi, editor, *IEEE Symposium on Logic Programming*, pages 379–388, San Francisco, Sept. 1987.

[28] T. Nipkow. Higher-order critical pairs. In *Proc. 6th LICS Conf.*, pages 342–349. IEEE, Computer Society Press, 1991.

[29] B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf type theory: An Introduction*. Clarendon, 1990.

[30] F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proc. SIGPLAN '88 Conf.* ACM Press, 1988.

[31] G. Plotkin. An illative theory of relations. In R. Cooper et al., editors, *Situation Theory and its Applications*, number 22 in CSLI Lecture Notes, pages 133–146. Stanford Univ., 1990.

[32] D. Scott. The lattice of flow diagrams. In E. Engeler, editor, *Symposium on Semantics of Algorithmic Languages*, volume 188 of *LNM*, pages 311–366. Springer, 1971.

[33] D. Scott. Relating theories of the λ-calculus. In J. Hindley and J. Seldin, editors, *To H.B. Curry: Essays on combinatory logic, lambda calculus and formalism*, pages 403–450. Academic Press, 1980.

[34] A. Stoughton. Substitution revisited. *Theoretical Computer Science*, 59(3):317–325, Aug. 1988.

[35] R. Tennent. *Semantics of Programming Languages*. Prentice Hall, New York, 1991.

[36] D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Proc. 12$^{\rm th}$ LICS Conf.* IEEE, Computer Society Press, 1997.

[37] S. Yong. *A Framework for Binding Operators*. PhD thesis, LFCS, Edinburgh, 1992.