Logical Methods in Computer Science
Volume 20, Issue 1, 2024, pp. 18:1–18:49
https://lmcs.episciences.org/

Submitted Sep. 07, 2022
Published Mar. 01, 2024

VARIABLE BINDING AND SUBSTITUTION FOR (NAMELESS) DUMMIES

ANDRÉ HIRSCHOWITZ [a], TOM HIRSCHOWITZ [b], AMBROISE LAFONT [c], AND MARCO MAGGESI [d]

[a] Univ. Côte d'Azur, CNRS, LJAD, 06103, Nice, France

[b] Univ. Savoie Mont Blanc, CNRS, LAMA, 73000, Chambéry, France

[c] LIX, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

[d] Università degli Studi di Firenze, Italy

Abstract. By abstracting over well-known properties of De Bruijn's representation with nameless dummies, we design a new theory of syntax with variable binding and capture-avoiding substitution. We propose it as a simpler alternative to Fiore, Plotkin, and Turi's approach, with which we establish a strong formal link. We also show that our theory easily incorporates simple types and equations between terms.

1. Introduction

In this paper we propose a new initial-algebra semantics [GT74] for syntax and substitution in the presence of variable binding, which gives a new perspective on the status of the well-known De Bruijn encoding [DB72].

Given a so-called binding signature [Plo90] (which we suppose untyped in this introduction), De Bruijn's encoding provides an explicit definition of the desired syntax; it consists of a (single) set of terms, equipped with a suitable operation of "substitution". The salient feature of De Bruijn's encoding is that variables are represented by natural numbers, which he termed "nameless dummies", hence the title of the present paper. The idea is that any occurrence of 0 refers to the binder just above it (in the abstract syntax tree), if any, while 1 refers to the next one up, and so on. E.g., $\lambda x.\lambda y.(x\ y)$ is represented by $\lambda.\lambda.(1\ 0)$. See [FPT99, Shu21] for more recent analyses. This encoding is generally considered "good for machine implementations, but not [...] for machine-assisted human reasoning" [GP99] (see also [ABF⁺05, BU07]).

Our initial-algebra semantics provides an alternative to the above *explicit* definition, by offering an *implicit* one:

- We design a category of "models" of the considered signature.

Key words and phrases: syntax and variable binding and substitution and category theory.
* Extended abstract (FoSSaCS 2022).

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.46298/LMCS-20(1:18)2024

- We define the desired syntax (up to unique isomorphism) as the initial object in this category.

One may then reason about syntax independently of any chosen initial object, since initiality provides a convenient induction principle.

Of course, we have to prove that such an initial object exists, and the natural witness in this proof is precisely De Bruijn's encoding. It thus acquires the new status of initiality witness, and hence may be forgotten, to some extent.

We know of two initial-algebra semantics for syntax with substitution in the presence of variable binding. A mainstream one is by Fiore et al. [FPT99, Fio08], while the second one, which also handles linear syntax, is due to Power [Pow07]. Both approaches consider terms indexed by the number of (potential) free variables. By contrast, ours involves a single (infinite and implicit) context. It is thus simpler, at least in the sense that it can naturally be implemented in a proof assistant without dependent types. We demonstrate this by implementing our framework in HOL Light. We also provide a Coq implementation for comparison.

Let us emphasise that our initial-algebra semantics optimises the usual layering into (1) syntax, (2) variable renaming, and (3) substitution. Indeed, we show that the second layer is unnecessary, and directly give the implicit definition of syntax with substitution in (unindexed) sets.

A consequence is that our mechanisations offer a very different **trusted computing base**[1] from what one usually gets with an explicit definition.

- With an explicit definition, the trusted computing base typically consists of
  - the inductive type defining the syntax,
  - the recursive definition of renaming, and
  - the recursive definition of substitution.
- By contrast, in our mechanisations, the trusted computing base consists of
  - the definition of the category of models, and
  - the initiality statement.

As the authors have experienced, the pros and cons can be discussed *ad libitum*. We refrain from doing so in this paper.

1.1. **Overview.**

Let us now present our contribution in a bit more detail, for which we should start by recalling binding signatures.

Definition 1.1. A **binding arity** is a sequence of natural numbers. A **binding signature** is a set O (of "operations"), together with a map $O \to \mathbb{N}^*$, which associates a binding arity to each operation.

The idea is that an operation of binding arity $b = (n_1, \ldots, n_p)$ has p arguments, with the ith argument binding n_i variables, for all $i \in \{1, \ldots, p\}$.

Example 1.2. In pure λ-calculus, the binding arity for application is $(0, 0)$: it has two arguments, binding no variables. Abstraction, on the other hand, has one argument which binds one variable. Its binding arity thus is the singleton sequence (1).

[1] I.e., the part of the development that needs to be read in order to check that the definitions and statements are correct.

We should now answer the question: where do operations of a given binding arity live, and what are they? To the first question, we answer that they live in a De Bruijn monad, whose definition we now sketch.

Definition 1.3. A **De Bruijn monad** is a set X, equipped with

- a **variables** map $v\colon \mathbb{N} \to X$, and
- a **substitution** map $s\colon X \times X^{\mathbb{N}} \to X$, which takes an element $x \in X$ and an assignment $\sigma\colon \mathbb{N} \to X$, and returns an element $s(x, \sigma)$, which we denote by $x[\sigma]$ when s is clear from context,

satisfying three simple axioms (see Definition 2.3 below).

Remark 1.4. The use of the word "monad" is justified by the fact that De Bruijn monads are in fact relative monads [ACU15], see Corollary 3.12 below.

To the second question, what is an operation of a given binding arity in a De Bruijn monad (X, v, s), we answer as follows.

Definition 1.5. An **operation of binding arity** $b = (n_1, \ldots, n_p)$ is a map $o\colon X^p \to X$ satisfying the following **binding condition**: for all $e_1, \ldots, e_p \in X$, and $\sigma\colon \mathbb{N} \to X$,

$$o(e_1, \ldots, e_p)[\sigma] = o(e_1[\Uparrow^{n_1} \sigma], \ldots, e_p[\Uparrow^{n_p} \sigma]), \tag{1.1}$$

where $\Uparrow$ is a unary operation defined on $X^{\mathbb{N}}$ by

$$(\Uparrow\sigma)(0) = v(0) \tag{1.2}$$

$$(\Uparrow\sigma)(n+1) = \sigma(n)[p \mapsto v(p+1)]. \tag{1.3}$$

To explain the idea behind this definition, let us consider the simplest non-trivial binding operation, abstraction in the pure λ-calculus, which has arity (1). The main idea of De Bruijn indices is that, under an abstraction:

(a) the bound variable is 0, and
(b) any outer variable k should be referred to as $1 + k$.

By the binding condition, for any operation $\lambda\colon X \to X$ of binding arity (1), we have $\lambda(e)[\sigma] = \lambda(e[\Uparrow\sigma])$. By definition, $\Uparrow\sigma$ leaves the variable 0 unchanged, which complies with (a) above. Furthermore, by definition, any reference $1 + k$ to some outer variable is mapped by $\Uparrow\sigma$ to $\sigma(k)[p \mapsto v(p+1)]$. That is, the intended element $\sigma(k)$, whose free variables are shifted by one to comply with (b).

From here, we straightforwardly define models of a given binding signature S to be De Bruijn monads equipped with operations of the specified binding arities. We call such models **De Bruijn S-algebras**, and organise them into a category S-**DBAlg**.

Finally, we prove that S-**DBAlg** admits an initial object (Theorem 3.16). For this, we follow (the standard modern variant of) De Bruijn's construction:

- We extract from S a first-order signature $|S|$, by mapping binding arities $(n_1, \ldots, n_p)$ to their lengths p, and construct the free $|S|$-algebra DB_S over the set $\mathbb{N}$ of variables in the usual, first-order way.
- We prove that DB_S admits a unique substitution map satisfying both the binding conditions and the De Bruijn monad axioms. This is not entirely trivial, because we cannot directly take (1.1)–(1.3) as a recursive definition. Indeed, the recursive call in (1.3) would not be decreasing, at least in any standard proof assistant's sense! We thus resort to the usual, two-phase construction:

- We first define a renaming map $\mathrm{DB}_S \times \mathbb{N}^{\mathbb{N}} \to \mathrm{DB}_S$, by adapting (1.1)–(1.3) to the renaming case.
- We then define the substitution map by (1.1)–(1.3), except that we replace the problematic recursive call in (1.3) by $\sigma(n)[p \mapsto p+1]$, which is a renaming, hence non recursive.

We finally prove that this uniquely equips DB_S with De Bruijn S-algebra structure, and that the obtaind De Bruijn S-algebra is initial.

Once this initial-algebra semantics is in place, we investigate the link with the above-mentioned mainstream framework of Fiore, Plotkin, and Turi. We find that both categories of models may include pathological objects, in the sense that we do not see any loss in ruling them out. When we do so, we obtain equivalent categories (Theorem 4.25).

Next, we devote two sections to investigating the status of binding signatures and the binding conditions. Indeed, binding signatures are combinatorial objects, and the binding conditions may seem somewhat arbitrary. We provide two categorical interpretations of binding signatures and binding conditions.

- We first recast binding signatures within Borthelle et al.'s framework [BHL20], which is a generalisation of Fiore's [Fio08]. After recalling the notion of **structurally strong** endofunctor (on **Set**), and the category Σ - **Mon** of models of such an endofunctor Σ, we show that any binding signature S gives rise to such an endofunctor Σ_S, and exhibit an isomorphism S - **DBAlg** $\cong \Sigma_S$ - **Mon** of categories over **DBMnd**.
- We then recast our initial-algebra semantics within the module-based approach to syntax with variable binding and substitution [HM07, HM10]. For this, we need to adapt the notion of parametric module over monads to De Bruijn monads, thus introducing **parametric De Bruijn modules**. We further define the category M - **MAlg** (for "modular algebras") of models of any such parametric De Bruijn module M. Finally, we show that any binding signature S gives rise to a parametric De Bruijn module M_S, and exhibit an isomorphism S - **DBAlg** $\cong M_S$ - **MAlg** of categories over **DBMnd**.

Our next two contributions extend the initial-algebra semantics in two different directions.

- We first propose a simply-typed generalisation, which is parameterised over a given set of types. We adopt a standard simply-typed variant of binding signatures [FH10], and prove a corresponding initiality result (Theorem 7.27). The strength-based and module-based recastings that we just mentioned could be extended to this setting, but we refrain from doing so for simplicity.
- Then, we consider equations. We introduce a notion of **De Bruijn equational theory**, and prove a corresponding initiality result (Theorem 8.7), whose witness is a straightforward quotient of De Bruijn's encoding.

Finally, in §9, we provide two mechanised versions of our framework: the first one is in Coq, while the second one is in HOL Light, a proof assistant which does not support dependent types, thus illustrating the simplicity of our theory.

1.2. **Plan of the paper.** In §2, we introduce De Bruijn monads, De Bruijn S-algebras, and the De Bruijn S-algebra DB_S. We furthermore prove (Theorem 2.21) that DB_S admits a unique substitution map satisfying the binding conditions with the desired behaviour on variables. In §3, we organise De Bruijn monads as a category, which we prove equivalent to categories of relative monads and of monoids. For any binding signature S, we then organise De Bruijn S-algebras into a category S - **DBAlg**, wherein we prove that DB_S is an initial object. In §4, we establish the announced link with the presheaf-based approach. In §5 and 6, we introduce our interpretations of binding signatures and binding conditions in terms of structurally strong endofunctors and modules, respectively. We enrich the framework with simple types in §7, and with equations in §8. In §9, we briefly describe our mechanisations in HOL Light and Coq. Finally, we conclude in §10.

1.3. **Related work.**

Abstract frameworks for variable binding. We have already mentioned the tight link with the presheaf-based approach [FPT99]. This link could probably be extended to variants such as [HM07, HM10, AM21, FS22].

In recent work, Allais et al. [AAC+18] introduce a universe of syntaxes, which essentially corresponds to a simply-typed version of binding signatures. Their framework is designed to facilitate the definition of so-called **traversals**, i.e., functions defined by structural induction, "traversing" their argument. In a similar spirit, let us mention the recent work of Gheri and Popescu [GP20], which presents a theory of syntax with binding, mechanised in Isabelle/HOL. Potential links between these frameworks and our approach remain unclear to us at the time of writing.

The categories of "intersectional" objects obtained in §4 are technically very close to nominal sets [GP99]: finite supports appear in the "action-based" presentation of nominal sets (and in our §4.2), while pullback preservation appears in their sheaf-based presentation (and in our §4.1). And indeed, any intersectional presheaf yields a nominal set, and so does any finitary De Bruijn monad. However, these links are not entirely satisfactory, because they do not account for substitution. The reason is that the only categorical theory of substitution that we know of for nominal sets, by Power [Pow07], is operadic rather than monadic, so we do not immediately see how to state a correspondence.

Finally, Pitts [Pit23] recently introduced semantics for the locally nameless approach to syntax, where bound variables are De Bruijn indices and free variables are chosen in a fixed infinite set of atoms. In some sense, his locally nameless sets are the counterpart of our finitary De Bruijn monads, in the untyped case. Beyond the difference between the locally nameless approach and the crude De Bruijn encoding we focus on, while only single-variable renamings are available in locally nameless sets, simultaneous substitution is built-in in De Bruijn monads. This enables us to define a notion of model (for a binding signature) with explicit compatibility conditions about substitution, resulting in a recursion principle which is compatible with substitution.

Proof assistant libraries. Allais et al. [AAC+18] and Gheri and Popescu [GP20] mechanise their approach in Agda and Isabelle/HOL, respectively. In the same spirit, the presheaf-based approach was recently formalised [FS22].

De Bruijn representation benefits from well-developed proof assistant libraries, in particular Autosubst [STS15, SSK19]. Such libraries are somewhat complementary to our work. Their main goal is to automate part of the reasoning about substitution in the proof assistant, while we provide an initial-algebra semantics. In particular, it could be useful to adapt the decision procedure of Autosubst to our Coq library.

1.4. **General notation.** We denote by $A^* = \sum_{n \in \mathbb{N}} A^n$ the set of finite sequences of elements of A, for any set A. In any category $\mathbf{C}$, we tend to write $[C, D]$ for the hom-set $\mathbf{C}(C, D)$ between any two objects C and D. Finally, for any endofunctor F, F - **alg** denotes the usual category of F-algebras and morphisms between them, and $\mu F = \mu X.F(X)$ will be its least fixed point. Finally, **CAT** denotes the large category of locally small categories.

2. De Bruijn monads

In this section, we start by introducing De Bruijn monads in an untyped setting. Then, we define assignment lifting, the binding conditions, and the models of a binding signature S in De Bruijn monads, De Bruijn S-algebras. Finally, we construct the term De Bruijn S-algebra DB_S.

2.1. **Definition of De Bruijn monads.** We start by fixing some terminology and notation, and then give the definition.

Definition 2.1. Given a set X, an **X-assignment** is a map $\mathbb{N} \to X$. We sometimes merely use "assignment" when X is clear from context.

Notation 2.2. Consider any map $s \colon X \times Y^{\mathbb{N}} \to Z$.

- For all $x \in X$ and $g \colon \mathbb{N} \to Y$, we write $x[g]_s$ for $s(x, g)$, or $x[g]_X$ when s is clear from context, or even $x[g]$ when s and X are clear from context.
- Furthermore, s gives rise to the map

$$\begin{array}{rcl} X^{\mathbb{N}} \times Y^{\mathbb{N}} & \to & Z^{\mathbb{N}} \\ (f, g) & \mapsto & n \mapsto s(f(n), g). \end{array}$$

  We use similar notation for this map, i.e., $f[g](n) := f(n)[g]_s$.

Definition 2.3. A **De Bruijn monad** is a set X, equipped with

- a **substitution** map $s \colon X \times X^{\mathbb{N}} \to X$, which takes an element $x \in X$ and an assignment $f \colon \mathbb{N} \to X$, and returns an element $x[f]$, and
- a **variables** map $v \colon \mathbb{N} \to X$,

satisfying, for all $x \in X$, and $f, g \colon \mathbb{N} \to X$:

- **associativity**: $x[f][g] = x[f[g]]$,
- **left unitality**: $v(n)[f] = f(n)$, and
- **right unitality**: $x[v] = x$.

Example 2.4. The set $\mathbb{N}$ itself is clearly a De Bruijn monad, with variables given by the identity and substitution $\mathbb{N} \times \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ given by evaluation. This is in fact the initial De Bruijn monad, as should be clear from the development below.

Example 2.5. The set $\Lambda := \mu X.\mathbb{N} + X + X^2$ of λ-terms forms a De Bruijn monad with well-known structure, which we now recall for completeness. Elements of Λ are generated by the following grammar, where n ranges over $\mathbb{N}$.

$$e ::= n \mid \lambda(e) \mid e\ e$$

The variables map $\mathbb{N} \to \Lambda$ sends any n to itself, i.e., the leaf labelled n. For substitution, we want it to satisfy the following mutually recursive equations:

$$\begin{aligned} v(n)[\sigma] &= \sigma(n) \\ (e_1\ e_2)[\sigma] &= e_1[\sigma]\ e_2[\sigma] \\ \lambda(e)[\sigma] &= \lambda(e[\Uparrow\sigma]) \end{aligned}$$

$$\begin{aligned} (\Uparrow\sigma)(0) &= v(0) \\ (\Uparrow\sigma)(n+1) &= \sigma(n)[v \circ \mathrm{succ}], \end{aligned}$$

where $\mathrm{succ}\colon \mathbb{N} \to \mathbb{N}$ denotes the successor map. However, the very last recursive call to substitution is not clearly decreasing in any way, so we cannot take this as a definition. Instead, we take it as a specification, and prove that there exist unique substitution and lifting maps satisfying the above equations.

For this, we use a standard technique, based on the observation that the problematic recursive call $(\sigma(n)[v \circ \mathrm{succ}])$ does not involve a general assignment but the mere renaming $v \circ \mathrm{succ}$. We replace this recursive call with $\sigma(n)\{\mathrm{succ}\}$, where $-\{-\}\colon \Lambda \times \mathbb{N}^{\mathbb{N}} \to \Lambda$ denotes a **renaming** map, easily defined by recursion as follows:

$$\begin{aligned} v(n)\{f\} &= v(f(n)) \\ (e_1\ e_2)\{f\} &= e_1\{f\}\ e_2\{f\} \\ \lambda(e)\{f\} &= \lambda(e\{\uparrow f\}), \end{aligned}$$

$$\text{where} \qquad \begin{aligned} (\uparrow f)(0) &= 0 \\ (\uparrow f)(n+1) &= f(n) + 1. \end{aligned}$$

(Because f is a mere renaming, the definition of $\uparrow$ is not recursive.)

It is then straightforward to prove that the original equations are (uniquely) satisfied.

In Example 3.17, as an application of Theorem 3.16, we will characterise the obtained De Bruijn monad by a universal property. In fact, the set $\Lambda := \mu X.\mathbb{N} + X + X^2$ has infinitely many De Bruijn monad structures, as many as there are binding arities with underlying endofunctors $X \mapsto X$ and $X \mapsto X^2$, in the sense defined below. But only one of these structures models λ-calculus substitution.

2.2. **Lifting assignments.** In preparation for introducing the binding conditions, given a De Bruijn monad M, we now define an operation called **lifting** on its set of assignments $\mathbb{N} \to M$. It is convenient to stress that only part of the structure of a De Bruijn monad is needed for this definition.

Definition 2.6. Consider any set M, equipped with maps $s\colon M \times M^{\mathbb{N}} \to M$ and $v\colon \mathbb{N} \to M$. For any assignment $\sigma\colon \mathbb{N} \to M$, we define the assignment $\Uparrow\sigma\colon \mathbb{N} \to M$ by

$$\begin{aligned} (\Uparrow\sigma)(0) &= v(0) \\ (\Uparrow\sigma)(n+1) &= \sigma(n)[\uparrow], \end{aligned}$$

where $\uparrow\colon \mathbb{N} \to X$ maps any n to $v(n+1)$.

Remark 2.7. Both $\Uparrow$ and $\uparrow$ depend on M and (part of) (s, v). Here, and in other similar situations below, we abuse notation and omit such dependencies for readability.

Of course we may iterate lifting:

Definition 2.8. Let $\Uparrow^0 A = A$, and $\Uparrow^{n+1} A = \Uparrow(\Uparrow^n A)$.

2.3. **Binding arities and binding conditions.** Our treatment of binding arities reflects the separation between the first-order part of the arity, namely its length, which concerns the syntax, and the binding information, namely the binding numbers, which concerns the compatibility with substitution.

Definition 2.9.

- A **first-order arity** is a natural number.
- A **binding arity** is a sequence $(n_1, \ldots, n_p)$ of natural numbers, i.e., an element of $\mathbb{N}^*$.
- The first-order arity $|a|$ associated with a binding arity $a = (n_1, \ldots, n_p)$ is its length p.

Let us now axiomatise what we call an operation of a given binding arity.

Definition 2.10. Let $a = (n_1, \ldots, n_p)$ be any binding arity, M be any set, $s\colon M \times M^{\mathbb{N}} \to M$, and $v\colon \mathbb{N} \to M$ be any maps. An operation **of binding arity** a is a map $o\colon M^p \to M$ satisfying the following a**-binding condition** w.r.t. (s, v):

$$\forall \sigma\colon \mathbb{N} \to M, x_1, \ldots, x_p \in M, \quad o(x_1, \ldots, x_p)[\sigma] = o(x_1[\Uparrow^{n_1}\sigma], \ldots, x_p[\Uparrow^{n_p}\sigma]). \tag{2.1}$$

Remark 2.11. Let us emphasise the dependency of this definition on v and s – which is hidden in the notation for substitution and lifting.

2.4. **Binding signatures and algebras.** In this section, we recall the standard notions of first-order (resp. binding) signatures, and adapt the definition of algebras to our De Bruijn context. Let us first briefly recall the former.

Definition 2.12. A **first-order signature** consists of a set O of **operations**, equipped with an **arity** map $ar\colon O \to \mathbb{N}$.

Definition 2.13. For any first-order signature $S := (O, ar)$, an S**-algebra** is a set X, together with, for each operation $o \in O$, a map $o_X\colon X^{ar(o)} \to X$.

Let us now generalise this to binding signatures.

Definition 2.14.

- A **binding signature** [Plo90] consists of a set O of **operations**, equipped with an **arity** map $ar\colon O \to \mathbb{N}^*$. Intuitively, the arity of an operation specifies the number of bound variables in each argument.
- The first-order signature $|S|$ associated with a binding signature $S := (O, ar)$ is $|S| := (O, |ar|)$, where $|ar|\colon O \to \mathbb{N}$ maps any $o \in O$ to the length $|ar(o)|$ of $ar(o)$.

Example 2.15. As we saw in Example 1.2, the binding signature for λ-calculus has two operations, abstraction and application, of respective arities (1) and $(0, 0)$. The associated first-order signature has two operations of respective arities 1 and 2.

Let us now present the notion of De Bruijn S-algebra:

Definition 2.16. For any binding signature $S := (O, ar)$, a **De Bruijn** S**-algebra** is a De Bruijn monad (X, s, v) equipped with an operation of binding arity $ar(o)$, for all $o \in O$.

In order to state our characterisation of the term model, we associate to any binding signature an endofunctor on sets, as follows.

Definition 2.17. The endofunctor Σ_S associated to a binding signature $S = (O, ar)$ is defined by $\Sigma_S(X) = \sum_{o \in O} X^{|ar(o)|}$.

Remark 2.18. The induced endofunctor merely depends on the underlying first-order signature.

Definition 2.19. For any binding signature $S = (O, ar)$ and Σ_S-algebra $a \colon \Sigma_S(X) \to X$, we call the composite $X^{|ar(o)|} \xrightarrow{in_o} \Sigma_S(X) \xrightarrow{a} X$ the **interpretation** of o in X.

Remark 2.20. As is well known, for any binding signature, the initial $(\mathbb{N} + \Sigma_S)$-algebra is the desired syntax; it has as carrier the least fixed point $\mu A.\mathbb{N} + \Sigma_S(A)$.

The following theorem defines the term model of a binding signature.

Theorem 2.21. *Consider any binding signature $S = (O, ar)$, and let DB_S denote the initial $(\mathbb{N} + \Sigma_S)$-algebra, with structure maps $v \colon \mathbb{N} \to \mathrm{DB}_S$ and $a \colon \Sigma_S(\mathrm{DB}_S) \to \mathrm{DB}_S$. Then,*

(i) *There exists a unique map $s \colon \mathrm{DB}_S \times {\mathrm{DB}_S}^{\mathbb{N}} \to \mathrm{DB}_S$ such that*
- *for all $n \in \mathbb{N}$ and $f \colon \mathbb{N} \to \mathrm{DB}_S$, $s(v(n), f) = f(n)$, and*
- *the interpretation of each $o \in O$ in DB_S satisfies the $ar(o)$-binding condition w.r.t. (s, v).*

(ii) *This map turns (DB_S, v, s, a) into a De Bruijn S-algebra.*

Proof. We have proved the result in both HOL Light [Mag22] and Coq [Laf22a], see §9. □

Remark 2.22. Point (i) may be viewed as an abstract form of recursive definition for substitution in the term model. The theorem thus allows us to construct the term model of a signature in two steps: first the underlying set, constructed as the inductive datatype $\mu Z.\mathbb{N} + \Sigma_S(Z)$, and then substitution, defined by the binding conditions viewed as recursive equations.

3. Initial-algebra semantics of binding signatures in De Bruijn monads

In this section, for any binding signature S, we organise De Bruijn S-algebras into a category, S-**DBAlg**, and prove that the term De Bruijn S-algebra DB_S is initial therein.

3.1. A category of De Bruijn monads.

Let us start by organising general De Bruijn monads into a category:

Definition 3.1. A morphism $(X, s, v) \to (Y, t, w)$ between De Bruijn monads is a set-map $f \colon X \to Y$ commuting with substitution and variables, in the sense that for all $x \in X$ and $\sigma \colon \mathbb{N} \to X$ we have $f(x[\sigma]) = f(x)[f \circ \sigma]$ and $f \circ v = w$.

Remark 3.2. More explicitly, the first axiom says: $f(s(x, \sigma)) = t(f(x), f \circ \sigma)$.

Notation 3.3. De Bruijn monads and morphisms between them form a category, which we denote by **DBMnd**.

3.2. **De Bruijn monads as relative monads and as monoids.** In this subsection, we briefly mention an alternative presentation of De Bruijn monads for the categorically-minded reader, in terms of **relative** monads. Namely, we show that they are monads relative to the functor $1 \to \mathbf{Set}$ picking $\mathbb{N}$. Then, following Altenkirch et al. [ACU15], we explain a companion presentation in terms of monoids in **Set**, for a suitable **skew monoidal** structure [ACU15, Szl12].

Remark 3.4. Altenkirch et al. have similarly shown that Fiore, Plotkin, and Turi's approach may be understood in terms of monads relative to the canonical embedding from finite sets into sets (and hence also in terms of monoids in a corresponding monoidal category).

Let us first briefly recall relative monads, which were introduced by Altenkirch et al. [ACU15].

Definition 3.5. For any set $\mathbf{E}$, category $\mathbf{C}$, and map $J \colon \mathbf{E} \to \mathbf{ob}(\mathbf{C})$, a ***J*-relative monad**, or **monad relative to** J, consists of

- an object mapping $T \colon \mathbf{E} \to \mathbf{ob}(\mathbf{C})$, together with
- **unit** morphisms $\eta_X \colon J(X) \to T(X)$, for all $X \in \mathbf{E}$, and
- for each morphism $f \colon J(X) \to T(Y)$, an **extension** $f^\dagger \colon T(X) \to T(Y)$,

such that the following diagrams commute for all $X, Y, Z \in \mathbf{E}$, $f \colon J(X) \to T(Y)$, and $g \colon J(Y) \to T(Z)$.

$$
\begin{array}{ccc}
J(X) & \xrightarrow{\eta_X} & T(X) \\
& {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle f^\dagger} & \\
& T(Y) &
\end{array}
\qquad
T(X) \underset{\quad}{\overset{\eta_X^\dagger}{\rightrightarrows}} T(X)
\qquad
\begin{array}{ccc}
T(X) & \xrightarrow{f^\dagger} & T(Y) \\
& {\scriptstyle (g^\dagger \circ f)^\dagger}\searrow \quad \swarrow{\scriptstyle g^\dagger} & \\
& T(Z) &
\end{array}
$$

A morphism $(T, \eta, (-)^\dagger) \to (T', \eta', (-)^{\dagger'})$ of J-relative monads consists of morphisms $\alpha_X \colon T(X) \to T'(X)$, for all $X \in \mathbf{E}$, making the following diagrams commute

$$
\begin{array}{ccc}
& J(X) & \\
{\scriptstyle \eta_X}\swarrow & & \searrow{\scriptstyle \eta'_X} \\
T(X) & \xrightarrow[\alpha_X]{} & T'(X)
\end{array}
\qquad
\begin{array}{ccc}
T(X) & \xrightarrow{f^\dagger} & T(Y) \\
{\scriptstyle \alpha_X}\downarrow & & \downarrow{\scriptstyle \alpha_Y} \\
T'(X) & \xrightarrow[(\alpha_Y \circ f)^{\dagger'}]{} & T'(Y)
\end{array}
$$

for all $X, Y \in \mathbf{E}$, and $f \colon J(X) \to T(X)$. Monads relative to J and morphisms between them form a category.

Remark 3.6. This definition is slightly different from, but equivalent to the original.

Proposition 3.7. *The category* **DBMnd** *is canonically isomorphic to the category of monads relative to the map* $1 \to \mathbf{Set}$ *picking* $\mathbb{N}$.

Remark 3.8. Canonicity here means that the isomorphism lies over the canonical isomorphism $[1, \mathbf{Set}] \cong \mathbf{Set}$.

Proof. By mere definition unfolding:

- An object mapping $T \colon 1 \to \mathbf{Set}$ amounts to a choice of object X in **Set**.
- A unit $\eta \colon \mathbb{N} \to X$ amounts to a choice of variables map.
- The assignment of an extension $f^\dagger \colon X \to X$ to each $f \colon \mathbb{N} \to X$ amounts to a map $X^{\mathbb{N}} \to X^X$, which is equivalent by uncurrying to a choice of substitution map $X \times X^{\mathbb{N}} \to X$. Notationally, $f^\dagger(x)$ thus corresponds to $x[f]$.

We then check that the axioms match:

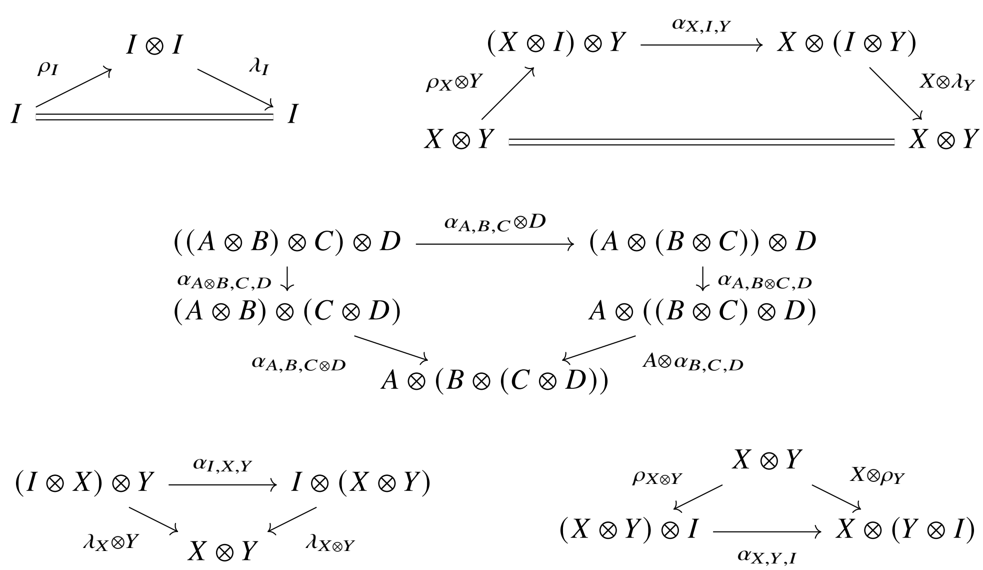
- Right unitality $x[\eta] = x$ corresponds to $\eta^\dagger(x) = x$, i.e., $\eta^\dagger = \mathrm{id}_X$.
- Left unitality $\eta(n)[f] = f(n)$ corresponds to $f^\dagger(\eta(n)) = f(n)$, i.e., $f^\dagger \circ \eta = f$.
- For associativity $x[f][g] = x[f[g]]$, by definition $f[g](n) = f(n)[g]$, so $f[g]$ corresponds to $n \mapsto g^\dagger(f(n))$, i.e., $g^\dagger \circ f$, and the axiom becomes $g^\dagger \circ f^\dagger = (g^\dagger \circ f)^\dagger$, as desired. □

Following Altenkirch et al., let us now give a further alternative characterisation of the category **DBMnd** of De Bruijn monads in terms of skew monoidal categories, which we now recall.

Definition 3.9 (Szlachányi [Szl12])**.** A **skew monoidal category** is a category $\mathbf{C}$ equipped with a **tensor product** functor $\otimes \colon \mathbf{C}^2 \to \mathbf{C}$, written in infix notation, and a **unit** object $I \in \mathbf{C}$, together with

- an **associator** natural transformation $\alpha_{X,Y,Z} \colon (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$,
- a **right unitor** natural transformation $\rho_X \colon X \to X \otimes I$, and
- a **left unitor** natural transformation $\lambda_X \colon I \otimes X \to X$,

satisfying the following coherence conditions.



We will now show that De Bruijn monads are equivalently monoids for some suitable skew monoidal category structure on **Set**. For this, we introduce the following terminology.

Notation 3.10. For a functor $J \colon \mathbf{E} \to \mathbf{C}$, "$J$-relative monad" means monad relative to the object mapping $\mathbf{ob}(\mathbf{E}) \to \mathbf{ob}(\mathbf{C})$ of J.

Proposition 3.11. *For any small category* $\mathbf{E}$*, cocomplete category* $\mathbf{C}$*, and functor* $J \colon \mathbf{E} \to \mathbf{C}$*, the functor category* $[\mathbf{E}, \mathbf{C}]$ *is skew monoidal, with tensor* $G \otimes F = \sum_J(G) \circ F$ *and unit* $I = J$*, where* $\sum_J$ *denotes left Kan extension along* J*. Furthermore, monoids in* $[\mathbf{E}, \mathbf{C}]$ *are in one-to-one correspondence with monads relative to* J*.*

Proof. By [ACU15, Theorems 4 and 5]. □

Applying this to the functor $J \colon 1 \to \mathbf{Set}$ picking $\mathbb{N}$ and transferring across the isomorphism $[1, \mathbf{Set}] \cong \mathbf{Set}$, we obtain a skew monoidal structure on sets, and Proposition 3.11 gives:

Corollary 3.12. *The tensor product* $X \otimes Y := X \times Y^{\mathbb{N}}$ *extends to a skew monoidal structure on* **Set**, *with:*

- *unit* $\mathbb{N}$,
- *right unitor* $\rho_X \colon X \to X \otimes \mathbb{N}$ *given by* $\rho_X(x) = (x, \mathrm{id}_{\mathbb{N}})$,
- *left unitor* $\lambda_X \colon \mathbb{N} \otimes X \to X$ *given by evaluation* $\lambda_X(n, \sigma) = \sigma(n)$, *and*
- *associator* $\alpha_{X,Y,Z} \colon (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$ *given by* $\alpha_{X,Y,Z}((x, \upsilon), \zeta) = (x, (n \mapsto (\upsilon(n), \zeta)))$.

Furthermore, **DBMnd** *is precisely the category of monoids therein.*

Proof. By the standard formula for left Kan extension, we have

$$\begin{aligned} \sum_J (X)(U) &\cong \int^{\star \in 1} \mathbf{Set}(J(\star), U) \times X(\star) \\ &\cong \mathbf{Set}(\mathbb{N}, U) \times X(\star) \\ &\cong U^{\mathbb{N}} \times X(\star). \end{aligned}$$

□

Remark 3.13. By Notation 3.10, if two functors $J \colon \mathbf{E} \to \mathbf{C}$ and $J' \colon \mathbf{E}' \to \mathbf{C}$ have the same object mapping up to isomorphism (hence in particular $\mathbf{ob}(\mathbf{E}) \cong \mathbf{ob}(\mathbf{E}')$), then J-relative monads are isomorphic to J'-relative monads, and both are isomorphic to monoids in $[\mathbf{E}, \mathbf{C}]$, resp. in $[\mathbf{E}', \mathbf{C}]$ (under the assumptions of Proposition 3.11).

In particular, the functor $1 \to \mathbf{Set}$ picking $\mathbb{N}$ factors as

$$1 \xrightarrow{I} \mathbf{B}[\mathbb{N}, \mathbb{N}] \xrightarrow{K} \mathbf{Set},$$

where $\mathbf{B}[\mathbb{N}, \mathbb{N}]$ denotes the full subcategory spanned by $\mathbb{N}$. Since the object mapping of K is the same as that of $1 \to \mathbf{Set}$, De Bruijn monads are equivalently monoids in the category $[\mathbf{B}[\mathbb{N}, \mathbb{N}], \mathbf{Set}]$. Remarkably, unlike **Set** with the skew monoidal structure of Corollary 3.12, $[\mathbf{B}[\mathbb{N}, \mathbb{N}], \mathbf{Set}]$ is in fact monoidal.

3.3. Categories of De Bruijn algebras.

In this section, for any binding signature S, we organise De Bruijn S-algebras into a category S - **DBAlg**.

Let us start by recalling the category of S-algebras for a first-order S:

Definition 3.14. For any first-order signature S, a morphism $X \to Y$ of S-algebras is a map between underlying sets commuting with operations, in the sense that for each $o \in O$, letting $p := ar(o)$, we have $f(o_X(x_1, \ldots, x_p)) = o_Y(f(x_1), \ldots, f(x_p))$.

We denote by S - **alg** the category of S-algebras and morphisms between them.

We now exploit this to define morphisms between De Bruijn S-algebras:

Definition 3.15. For any binding signature S, a morphism of De Bruijn S-algebras is a map $f \colon X \to Y$ between underlying sets, which is a morphism both of De Bruijn monads and of $|S|$-algebras. We denote by S - **DBAlg** the category of De Bruijn S-algebras and morphisms between them.

Theorem 3.16. *Consider any binding signature* $S = (O, ar)$, *and let* DB_S *denote the initial* $(\mathbb{N} + \Sigma_S)$*-algebra. Then, the De Bruijn* S*-algebra structure of Theorem 2.21 on* DB_S *makes it initial in* S - **DBAlg**.

Proof. We have proved the result in both HOL Light [Mag22] and Coq [Laf22a], see §9. □

Example 3.17. For the binding signature of λ-calculus (Example 2.15), the carrier of the initial model is $\mu Z.\mathbb{N} + Z + Z^2$.

4. Relation to presheaf-based models

The classical initial-algebra semantics introduced in [FPT99, Fio08] associates in particular to each binding signature S a category, say $\mathbf{\Phi}_S$ - **Mon** of models, while we have proposed in §3 an alternative category of models S - **DBAlg**. In this section, we are interested in comparing both categories of models.

In fact, we find that both may include pathological models, in the sense that we do not see any loss in ruling them out. And when we do so, we obtain equivalent categories.

4.1. **Trimming down presheaf-based models.**

First of all, in this subsection, let us recall the mainstream approach we want to relate to, and exclude some pathological objects from it.

4.1.1. *Presheaf-based models.* We start by recalling the presheaf-based approach. The ambient category is the category of functors $[\mathbb{F}, \mathbf{Set}]$, where $\mathbb{F}$ denotes the category of finite ordinals, and all maps between them.

Definition 4.1. Let $[\mathbf{Set}, \mathbf{Set}]_f$ denote the full subcategory of $[\mathbf{Set}, \mathbf{Set}]_f$ spanning **finitary** functors, i.e., those preserving directed colimits (= colimits of directed posets).

Proposition 4.2. *The restriction functor* $[\mathbf{Set}, \mathbf{Set}]_f \to [\mathbb{F}, \mathbf{Set}]$ *is an equivalence.*

Proof. The category of sets is ω-accessible, so by [AR94, Theorem 2.26, (i) $\Leftrightarrow$ (ii)] and [AR94, Remark 2.26(1)], it is a free cocompletion of its full subcategory of finitely presentable objects under directed colimits. Equivalently, it is a free cocompletion of $\mathbb{F}$ under directed colimits. Thus, by taking $\mathcal{B}$ to be **Set** in [AR94, Definition 2.25], we obtain that the restriction functor $[\mathbf{Set}, \mathbf{Set}]_f \to [\mathbb{F}, \mathbf{Set}]$ is an equivalence. □

Definition 4.3. Let $(\otimes, I)$ denote the monoidal structure on $[\mathbb{F}, \mathbf{Set}]$ inherited from the composition monoidal structure on $[\mathbf{Set}, \mathbf{Set}]_f$ through the equivalence of Proposition 4.2.

By construction, monoids in $[\mathbb{F}, \mathbf{Set}]$ are thus equivalent to finitary monads on sets.

The idea is then to interpret binding signatures S as endofunctors $\mathbf{\Phi}_S$ on $[\mathbb{F}, \mathbf{Set}]$, and to define models as monoids equipped with $\mathbf{\Phi}_S$-algebra structure, satisfying a suitable compatibility condition.

The definition of $\mathbf{\Phi}_S$ relies on an operation called derivation:

Definition 4.4 (Endofunctor associated to a binding signature)**.**

- Let the **derivative** X' of any functor $X \colon \mathbb{F} \to \mathbf{Set}$ be defined by $X'(n) = X(n+1)$.
- Furthermore, let $X^{(0)} = X$, and $X^{(n+1)} = (X^{(n)})'$.
- For any binding arity $a = (n_1, \ldots, n_p)$, let $\mathbf{\Phi}_a(X) = X^{(n_1)} \times \ldots \times X^{(n_p)}$.
- For any binding signature $S = (O, ar)$, let $\mathbf{\Phi}_S = \sum_{o \in O} \mathbf{\Phi}_{ar(o)}$.

Proposition 4.5. *Through the equivalence with finitary functors, derivation becomes* $F'(A) = F(A+1)$, *for any finitary* $F \colon \mathbf{Set} \to \mathbf{Set}$ *and* $A \in \mathbf{Set}$.

Example 4.6. On the binding signature for λ-calculus, say S_λ, which we saw in Example 2.15, we get

$$\Phi_{S_\lambda}(X)(n) = X(n)^2 + X(n+1).$$

Next, we want to express the relevant compatibility condition between algebra and monoid structure. For this, let us briefly recall the notion of pointed strength, see [FPT99, Fio08] for details.

Definition 4.7. A **pointed strength** on an endofunctor $F\colon \mathbf{C} \to \mathbf{C}$ on a monoidal category $(\mathbf{C}, \otimes, I, \alpha, \lambda, \rho)$ is a family of morphisms $st_{C,(D,v)}\colon F(C) \otimes D \to F(C \otimes D)$, natural in $C \in \mathbf{C}$ and $(D, v\colon I \to D) \in I/\mathbf{C}$, the coslice category under the tensor unit I, making the following diagrams commute,

$$\begin{array}{ccc} & F(A) & \\ {\scriptstyle \rho_{F(A)}} \swarrow & & \searrow {\scriptstyle F(\rho_A)} \\ F(A) \otimes I & \xrightarrow[st_{A,(I,\mathrm{id})}]{} & F(A \otimes I) \end{array}$$

$$\begin{array}{ccccc} (F(A) \otimes X) \otimes Y & \xrightarrow{st_{A,(X,v_X)} \otimes Y} & F(A \otimes X) \otimes Y & \xrightarrow{st_{A\otimes X,(Y,v_Y)}} & F((A \otimes X) \otimes Y) \\ {\scriptstyle \alpha_{F(A),X,Y}} \downarrow & & & & \downarrow {\scriptstyle F(\alpha_{A,X,Y})} \\ F(A) \otimes (X \otimes Y) & & \xrightarrow[st_{A,(X\otimes Y, v_{X\otimes Y})}]{} & & F(A \otimes (X \otimes Y)) \end{array}$$

for all objects A, X, Y, and morphisms $v_X\colon I \to X$ and $v_Y\colon I \to Y$, where $v_{X\otimes Y}$ denotes the composite

$$I \xrightarrow{\rho_I} I \otimes I \xrightarrow{v_X \otimes v_Y} X \otimes Y.$$

The next step is to observe that binding signatures generate pointed strong endofunctors.

Definition 4.8. The derivation endofunctor $X \mapsto X'$ on $[\mathbb{F}, \mathbf{Set}]$ has a pointed strength, defined through the equivalence with finitary functors by

$$G(F(X)+1) \xrightarrow{G(F(X)+v_1)} G(F(X)+F(1)) \xrightarrow{G[F(in_1),F(in_2)]} G(F(X+1)).$$

Product, coproduct, and composition of endofunctors lift to pointed strong endofunctors, which yields:

Corollary 4.9 [FPT99, Fio08]**.** *For all binding signatures S, Φ_S is canonically pointed strong.*

At last, we arrive at the definition of models.

Definition 4.10. For any pointed strong endofunctor F on a monoidal category $(\mathbf{C}, \otimes, I, \alpha, \lambda, \rho)$, an **$F$-monoid** is an object X equipped with F-algebra and monoid structure, say $a\colon F(X) \to X$, $s\colon X \otimes X \to X$, and $v\colon I \to X$, such that the following pentagon commutes.

$$\begin{array}{ccccc} F(X) \otimes X & \xrightarrow{st_{X,(X,v)}} & F(X \otimes X) & \xrightarrow{F(s)} & F(X) \\ {\scriptstyle a\otimes X} \downarrow & & & & \downarrow {\scriptstyle a} \\ X \otimes X & & \xrightarrow[s]{} & & X \end{array}$$

A morphism of F-monoids is a morphism in $\mathbf{C}$ which is a morphism both of F-algebras and of monoids. We let $F\text{-}\mathbf{Mon}$ denote the category of F-monoids and morphisms between them.

Example 4.11. For the binding signature S_λ of Example 2.15, a Φ_{S_λ}-monoid is an object X, equipped with maps $X' \to X$ and $X^2 \to X$, and with compatible monoid structure. Compatibility describes how substitution should be pushed down through abstractions and applications.

4.1.2. *Intersectional presheaves.* The pathology we want to rule out only concerns the underlying functor of a model, so we just have to define well-behaved functors in $[\mathbb{F}, \mathbf{Set}]$.

Well-behavedness for a functor $T\colon \mathbb{F} \to \mathbf{Set}$ is about getting closed terms right. More precisely, for some finite sets m and n, an element of $T(m+n)$ which both exists in $T(m)$ and $T(n)$ should also exist in $T(\emptyset)$, and uniquely so. This says exactly that T should preserve the pullback

$$\begin{array}{ccc} \emptyset & \longrightarrow & n \\ \downarrow & \lrcorner & \downarrow \\ m & \longrightarrow & m+n. \end{array} \tag{4.1}$$

Taking **intersection** to mean pullback of two monomorphisms, the following known result shows that all non-empty intersections are automatically preserved.

Proposition 4.12 [AGT10, Proposition 2.2]**.** *All endofunctors of sets preserve non-empty intersections.*

Thus, by Proposition 4.2, all functors $\mathbb{F} \to \mathbf{Set}$ preserve non-empty intersections, and we have:

Corollary 4.13. *A functor $\mathbb{F} \to \mathbf{Set}$ preserves (binary) intersections iff it preserves empty (binary) intersections.*

Lemma 4.14. *A functor T from $\mathbf{Set}$ (or $\mathbb{F}$) to $\mathbf{Set}$ preserves empty binary intersections if and only if it preserves the following pullback.*

$$\begin{array}{ccc} 0 & \longrightarrow & 1 \\ \downarrow & \lrcorner & \downarrow{\scriptstyle 1} \\ 1 & \xrightarrow[0]{} & 2 \end{array}$$

Proof. By Proposition 4.2, it is enough to reason on an endofunctor T on $\mathbf{Set}$. If T preserves empty binary intersections, then it preserves the above pullback as a particular case. Conversely, assume that it preserves the above pullback. Then, the following diagram is an equaliser.

$$T(0) \longrightarrow T(1) \underset{T(1)}{\overset{T(0)}{\rightrightarrows}} T(2).$$

Therefore, T coincides with its so-called Trnková closure and thus by [AMBL12, Corollary VII.2], it preserves finite intersections. ☐

Definition 4.15.

- A functor $\mathbb{F} \to \mathbf{Set}$ is **intersectional** iff it preserves binary intersections, or equivalently empty binary intersections. Let $[\mathbb{F}, \mathbf{Set}]_{int}$ denote the full subcategory spanned by intersectional functors.

- A monoid in $[\mathbb{F}, \mathbf{Set}]$, (resp., for any binding signature S, an object of Φ_S - **Mon**) is **intersectional** iff the underlying functor is. Let Φ_S - $\mathbf{Mon}_{int}$ denote the full subcategory spanned by intersectional objects.

Example 4.16. As an example of a non intersectional finitary monad, first consider the monad L of λ-calculus, so that $L(X)$ is set of λ-terms taking free variables in X. This monad is intersectional, but now consider the monad L' agreeing with L on any non-empty set, and such that $L'(\emptyset) = \emptyset$. Then, L' is not intersectional.

The important result for comparing the presheaf-based approach with ours is the following.

Proposition 4.17. *The subcategory* Φ_S - $\mathbf{Mon}_{int}$ *includes the initial object.*

Proof. Roughly, closed terms are isomorphic to terms in two free variables that use neither the first, nor the second. □

Let us conclude this subsection with the following observation, that for a wide class of signatures all models are in fact well behaved.

Proposition 4.18. *If the initial object* DB'_S *of* Φ_S - **Mon** *has at least one closed term (i.e.,* $\mathrm{DB}'_S(\emptyset) \neq \emptyset$*), then* Φ_S - $\mathbf{Mon}_{int} = \Phi_S$ - **Mon**.

Proof. If T is a Φ_S-monoid, then by initiality there is a morphism $\mathrm{DB}'_S \to T$, and in particular a map $\mathrm{DB}'_S(\emptyset) \to T(\emptyset)$. Since $\mathrm{DB}'_S(\emptyset)$ is non-empty by assumption, $T(\emptyset)$ cannot be empty. The result then follows from [AMBL12, Proposition VII.7]: a monad T on **Set** either preserves the initial object, or is intersectional. □

Remark 4.19. The binding signatures for which the initial model has at least one closed term are those specifying at least a constant or an operation binding (at least) one variable in each argument.

4.2. Trimming down De Bruijn monads.

Let us now turn to well-behaved De Bruijn algebras. Here well-behavedness is about finitariness. However, it may not be immediately clear how to define finitariness of a De Bruijn monad.

Definition 4.20. A De Bruijn monad (X, s, v) is finitary iff each of its elements $x \in X$ has a (finite) support $N_x \in \mathbb{N}$, in the sense that for all $f : \mathbb{N} \to \mathbb{N}$ fixing the first N_x numbers, the corresponding renaming $v \circ f$ fixes x, i.e., $x[v \circ f] = x$.

Example 4.21. By Proposition 4.24 below, the initial S-algebra is finitary, for any binding signature S. For an example of infinitary De Bruijn monad, consider the greatest fixed point $\nu A.\mathbb{N} + \Sigma_S(A)$, for any S with at least one operation with more than one argument. E.g., if S has an operation of binding arity $(0, 0)$, like application in λ-calculus, then the term $v(0)\ (v(1)\ (v(2)\ \ldots))$ does not have finite support.

Proposition 4.22. *Let* $x \in X$ *be an element of a De Bruijn monad* (X, s, v)*. The following are equivalent:*

(1) x *has support* N*;*
(2) *given any pair of assignments* $f_1, f_2 : \mathbb{N} \to X$ *which coincide on the first* N *numbers,* $x[f_1] = x[f_2]$.

(3) *for any assignment* $f: \mathbb{N} \to X$ *fixing the first* N *variables (in the sense that* $f(n) = v(n)$ *for any* $n < N$*),* $x[f] = x$*;*

Proof. (1) ⇒ (2) Suppose given two assignments $f_1, f_2: \mathbb{N} \to X$ such that $f_1(n) = f_2(n)$ for any $n < N$. Let us fix some bijection $s: \mathbb{N} \to \mathbb{N} + \mathbb{N}$ such that$s(n) = in_1(n)$ for all $n < N$. For $i \in \{1, 2\}$, let $h_i: \mathbb{N} \to \mathbb{N}$ fix the first N numbers and map any $n \geq N$ to $s^{-1}(in_i(n))$, where $in_1, in_2: \mathbb{N} \to \mathbb{N} + \mathbb{N}$ are the coproduct injections. Furthermore, let $u: \mathbb{N} \to X$ map any $n < N$ to $f_1(n) = f_2(n)$ and any $n \geq N$ to $[f_1, f_2](s(n))$, where $[f_1, f_2]: \mathbb{N} + \mathbb{N} \to X$ denotes the copairing of f_1 and f_2. We then have $f_i = u \circ h_i$, for $i \in \{1, 2\}$. Indeed, for each i:

- For any $n < N$, we have $u(h_i(n)) = u(n) = f_i(n)$ by definition.
- For any $n \geq N$, we have $h_i(n) = s^{-1}(in_i(n))$, hence $s(h_i(n)) = in_i(n)$. But we know that $h_i(n) \geq N$, because otherwise we would have $s(h_i(n)) = in_1(k)$ for some $k < N$, which does not hold. We thus obtain $u(h_i(n)) = [f_1, f_2](s(h_i(n)) = [f_1, f_2](in_i(n)) = f_i(n)$.

Thus, $x[f_i] = x[u \circ h_i] = x[v \circ h_i][u]$. Since x has support N, $x[v \circ h_i] = x$. Hence, $x[f_1] = x[u] = x[f_2]$.

(2) ⇒ (3) Suppose given an assignment $f: \mathbb{N} \to X$ fixing the first N variables. Then, f coincides with v on the first N variables. Thus, $x[f] = x[v] = x$.

(3) ⇒ (1) Let $f: \mathbb{N} \to \mathbb{N}$ fixing the first N numbers. Then, $v \circ f$, as an assignment, also does. Thus, $x[v \circ f] = x$. □

Definition 4.23. For any binding signature S, let $S\text{-}\mathbf{DBAlg}_{fin}$ denote the full subcategory spanning De Bruijn S-algebras whose underlying De Bruijn monad is finitary.

Proposition 4.24. *The subcategory* $S\text{-}\mathbf{DBAlg}_{fin}$ *includes the initial object.*

Proof. One can define by induction the greatest free variable N of a term x (or 0 if x is closed). Then, x has support $N + 1$. □

4.3. **Bridging the gap.** We may at last state the relationship between initial-algebra semantics of binding signatures in presheaves and in De Bruijn monads:

Theorem 4.25. *Consider any binding signature* S*. The subcategories* $\Phi_S\text{-}\mathbf{Mon}_{int}$ *and* $S\text{-}\mathbf{DBAlg}_{fin}$ *are equivalent.*

Remark 4.26. The moral of this is that, if one removes pathological objects from both $\Phi_S\text{-}\mathbf{Mon}$ and $S\text{-}\mathbf{DBAlg}$, then one obtains equivalent categories, which both retain the initial object. Thus, up to equivalence, the two approaches to initial-algebra semantics of binding signatures differ only marginally.

Restricting attention to well-behaved objects, we may thus benefit from the strengths of both approaches. Typically, in De Bruijn monads, free variables need to be computed explicitly, while presheaves come with intrinsic scoping, as terms are indexed by sets of potential free variables. Conversely, in some settings, observational equivalence may relate programs with different sets of free variables [SW01]. In such cases, it is useful to have all terms collected in one single set. This needs to be computed (and involves non-trivial quotienting) in presheaves, while it is direct in De Bruijn monads.

The remainder of this section is devoted to sketching the proof of Theorem 4.25, and may be skipped on a first reading as it relies on the module-based interpretation of the binding conditions described later in §6.

We start by proving that both De Bruijn monads and finitary monads are monoids in monoidal, full subcategories of $[\mathbf{Set}, \mathbf{Set}]$. Let us first treat the easy case of finitary monads:

Lemma 4.27. *The category* $\mathbf{Mon}[\mathbb{F}, \mathbf{Set}]$ *of monoids in* $[\mathbb{F}, \mathbf{Set}]$ *for the monoidal structure of Definition 4.3, is equivalent to the category* $\mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_f$ *of monoids in* $[\mathbf{Set}, \mathbf{Set}]_f$*.*

Proof. By definition of the monoidal structure on $[\mathbb{F}, \mathbf{Set}]$. □

Now for De Bruijn monads:

Definition 4.28. Let $[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}$ denote the full subcategory spanned by $\aleph_1$-accessible endofunctors which preserve empty intersections.

Lemma 4.29. *Evaluation at* $\mathbb{N}$ *induces an equivalence between the category* $\mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}$ *of monoids in* $[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}$ *and the category* $\mathbf{DBMnd}$ *of De Bruijn monads.*

Note that any monad T on $\mathbf{Set}$ induces a De Bruijn monad $T(\mathbb{N})$ by restricting the monadic bind and unit. This induces a functor whose restriction to $\mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}$ underlies the above claimed equivalence.

Proof sketch. De Bruijn monads are equivalently monads relative to the embedding $\mathbf{B}[\mathbb{N}, \mathbb{N}] \to \mathbf{Set}$ of the full subcategory on $\mathbb{N} \in \mathbf{Set}$. Now, presheaves on a category are equivalent to presheaves on its Cauchy completion, and we prove that the Cauchy completion of $\mathbf{B}[\mathbb{N}, \mathbb{N}]$, i.e., the category of idempotent maps $\mathbb{N} \to \mathbb{N}$, is equivalent to the full subcategory $\bar{\mathbb{F}}^+$ of $\mathbf{Set}$ spanned by non-empty, finite ordinals and $\mathbb{N}$. De Bruijn monads are thus equivalent to monads relative to the embedding $J^+ \colon \bar{\mathbb{F}}^+ \hookrightarrow \mathbf{Set}$. Now, because the embedding is full, functors $\bar{\mathbb{F}}^+ \hookrightarrow \mathbf{Set}$ are equivalent to functors $\mathbf{Set} \to \mathbf{Set}$ which preserve the initial object and are $\aleph_1$-accessible. Letting $[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, 0}$ denote the category of such functors, we thus obtain an equivalence

$$[\mathbf{B}[\mathbb{N}, \mathbb{N}], \mathbf{Set}] \simeq [\mathbf{Set}, \mathbf{Set}]_{\aleph_1, 0},$$

which is monoidal. We thus obtain an equivalence

$$\mathbf{DBMnd} \simeq \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, 0}.$$

It remains to make the link with $\mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}$. At this point, there is a difficulty. Indeed, the functor $\mathbf{pin} \colon [\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0} \to [\mathbf{Set}, \mathbf{Set}]_{\aleph_1, 0}$ defined by

$$\mathbf{pin}(F)(X) = \begin{cases} \emptyset & \text{if } X = \emptyset \\ F(X) & \text{otherwise} \end{cases}$$

is an equivalence preserving the identity endofunctor, but it is however not monoidal: e.g., letting $F = \emptyset$ and $G = 1$, we have

$$\mathbf{pin}(G \circ F)(1) = G(F(1)) = G(\emptyset) = 1$$

while

$$(\mathbf{pin}(G) \circ \mathbf{pin}(F))(1) = \mathbf{pin}(G)(F(1)) = \mathbf{pin}(G)(\emptyset) = \emptyset.$$

Still, we have:

Lemma 4.30. *For any monoid* $F \in [\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}$ *and any* $G \in [\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}$*, we have* $\mathbf{pin}(G \circ F) = \mathbf{pin}(G) \circ \mathbf{pin}(F)$*.*

Proof. We prove the more general fact that, if $F(X) \neq \emptyset$ at any $X \neq \emptyset$, then $\mathbf{pin}(G) \circ \mathbf{pin}(F) = \mathbf{pin}(G \circ F)$:

- at $X = \emptyset$, we have
$$(\mathbf{pin}(G) \circ \mathbf{pin}(F))(\emptyset) = \emptyset = \mathbf{pin}(G \circ F)(\emptyset),$$
- at any $X \neq \emptyset$, we have
$$(\mathbf{pin}(G) \circ \mathbf{pin}(F))(X) = \mathbf{pin}(G)(F(X)) = G(F(X)) = \mathbf{pin}(G \circ F)(X).$$

But at any $X \neq \emptyset$, any monoid F is equipped with a unit component $X \to F(X)$, so $F(X) \neq \emptyset$, hence the result. □

We thus obtain an equivalence
$$\mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0} \simeq \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, 0}$$
over **pin**, hence the result. □

We then characterise well-behavedness in both contexts, as follows.

Definition 4.31. Let $[\mathbf{Set}, \mathbf{Set}]_{f, int_0}$ denote the full subcategory of $[\mathbf{Set}, \mathbf{Set}]$ spanned by finitary endofunctors preserving empty intersections.

Lemma 4.32. *The following squares commute,*
$$\begin{array}{ccc} \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{f, int_0} & \longrightarrow & \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0} \\ {\scriptstyle\simeq}\downarrow & & \downarrow{\scriptstyle\simeq} \\ \mathbf{DBMnd}_{fin} & \longrightarrow & \mathbf{DBMnd} \end{array} \qquad \begin{array}{ccc} \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{f, int_0} & \longrightarrow & \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{f} \\ {\scriptstyle\simeq}\downarrow & & \downarrow{\scriptstyle\simeq} \\ \mathbf{Mon}[\mathbb{F}, \mathbf{Set}]_{int} & \longrightarrow & \mathbf{Mon}[\mathbb{F}, \mathbf{Set}] \end{array}$$
and all vertical functors are equivalences.

Proof. For De Bruijn monads, well-behavedness is finitarity. For presheaves, well-behavedness is preservation of empty intersections. □

Corollary 4.33. *We obtain a chain of equivalences*
$$\mathbf{DBMnd}_{fin} \simeq \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{f, int_0} \simeq \mathbf{Mon}[\mathbb{F}, \mathbf{Set}]_{int}. \tag{4.2}$$

The point is now to prove that this chain of equivalences lifts to one between S - $\mathbf{DBAlg}_{fin}$ and Φ_S - $\mathbf{Mon}_{int}$, for any binding signature S.

For this, we adopt the viewpoint of modules over monads [HM10] (see §6 below for the module-based interpretation of the binding conditions). Let $S = (O, ar)$ denote any binding signature. We first introduce the analogue of the endofunctor Φ_S induced by a binding signature (Definition 4.4) in the context of $[\mathbf{Set}, \mathbf{Set}]$:

Definition 4.34. We define $\mathcal{F}_S \colon [\mathbf{Set}, \mathbf{Set}] \to [\mathbf{Set}, \mathbf{Set}]$ by
$$\mathcal{F}_S(F)(X) = \sum_{o \in O} \prod_{i \in |ar(o)|} F(X + ar(o)_i).$$

We then show that this functor restricts to the relevant subcategories.

Proposition 4.35. *For any* $\mathbf{C} \in \{[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}, [\mathbf{Set}, \mathbf{Set}]_f, [\mathbf{Set}, \mathbf{Set}]_{f, int_0}\}$*, the functor* $\mathcal{F}_S$ *restricts to a functor* $\mathcal{F}_S^{\mathbf{C}} \colon \mathbf{C} \to \mathbf{C}$ *making the following square commute.*
$$\begin{array}{ccc} \mathbf{C} & \xrightarrow{\mathcal{F}_S^{\mathbf{C}}} & \mathbf{C} \\ \downarrow & & \downarrow \\ [\mathbf{Set}, \mathbf{Set}] & \xrightarrow[\mathcal{F}_S]{} & [\mathbf{Set}, \mathbf{Set}] \end{array}$$

Furthermore, for any monoid $T \in \mathbf{C}$, $\mathcal{F}_S^{\mathbf{C}}(T)$ forms a T-module, with action given at each $o \in O$ with $ar(o) = (n_1, \dots, n_p)$ by

$$\prod_{i \in p} T(T(X) + n_i) \to \prod_{i \in p} T(T(X + n_i)) \xrightarrow{\prod_i \mu} \prod_{i \in p} T(X + n_i).$$

Proof. For the various restrictions of $\mathcal{F}_S$, one checks that each of the conditions (finitarity, $\aleph_1$-accessibility, preservation of empty intesections) is closed under coproducts, finite products, and shift, i.e., any $T \mapsto T(- + n)$. For the second statement, it holds in $[\mathbf{Set}, \mathbf{Set}]$, and all considered subcategories are full. □

Definition 4.36. For any $\mathbf{C} \in \{[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}, [\mathbf{Set}, \mathbf{Set}]_f, [\mathbf{Set}, \mathbf{Set}]_{f, int_0}\}$, an $S^{\mathbf{C}}$**-algebra** is a monoid T in $\mathbf{C}$ equipped with a module morphism $\mathcal{F}_S^{\mathbf{C}}(T) \to T$. A morphism of $S^{\mathbf{C}}$-algebras is a monoid morphism commuting with action. We denote by $S^{\mathbf{C}}\text{-}\mathbf{alg}$ the category of $S^{\mathbf{C}}$-algebras.

We next prove that in the case of $[\mathbf{Set}, \mathbf{Set}]_f$ and $[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0}$ this interpretation of S corresponds to its interpretations in presheaves and De Bruijn monads through the equivalences of Lemmas 4.27 and 4.29.

Lemma 4.37. *We have commuting squares*

$$\begin{array}{ccc} S^{[\mathbf{Set},\mathbf{Set}]_f}\text{-}\mathbf{alg} & \xrightarrow{\simeq} & \Phi_S\text{-}\mathbf{Mon} \\ \downarrow & & \downarrow \\ \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_f & \xrightarrow[\simeq]{} & \mathbf{Mon}[\mathbb{F}, \mathbf{Set}] \end{array} \qquad \begin{array}{ccc} S^{[\mathbf{Set},\mathbf{Set}]_{\aleph_1, int_0}}\text{-}\mathbf{alg} & \xrightarrow{\simeq} & S\text{-}\mathbf{DBAlg} \\ \downarrow & & \downarrow \\ \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0} & \xrightarrow[\simeq]{} & \mathbf{DBMnd} \end{array} \tag{4.3}$$

Proof. The first square is easy. The second is a tedious verification that the binding conditions correspond to the definition of module morphisms $\mathcal{F}_S^{[\mathbf{Set},\mathbf{Set}]_{\aleph_1, int_0}}(T) \to T$. □

Finally, we show that the restrictions of $\Phi_S\text{-}\mathbf{Mon}$ and $S\text{-}\mathbf{DBAlg}$ to well-behaved objects are equivalent to $S^{[\mathbf{Set},\mathbf{Set}]_{f, int_0}}\text{-}\mathbf{alg}$.

Indeed, by definition, we have pullback squares

$$\begin{array}{ccc} \Phi_S\text{-}\mathbf{Mon}_{int} & \longrightarrow & \Phi_S\text{-}\mathbf{Mon} \\ \downarrow & \lrcorner & \downarrow \\ \mathbf{Mon}[\mathbb{F}, \mathbf{Set}]_{int} & \longrightarrow & \mathbf{Mon}[\mathbb{F}, \mathbf{Set}] \end{array} \qquad \begin{array}{ccc} S\text{-}\mathbf{DBAlg}_{fin} & \longrightarrow & S\text{-}\mathbf{DBAlg} \\ \downarrow & \lrcorner & \downarrow \\ \mathbf{DBMnd}_{fin} & \hookrightarrow & \mathbf{DBMnd} \end{array} \tag{4.4}$$

so by the equivalences (4.3) and (4.2) the theorem follows from the next result.

Proposition 4.38. *We have the following pullback squares.*

$$\begin{array}{ccccc} S^{[\mathbf{Set},\mathbf{Set}]_f}\text{-}\mathbf{alg} & \longleftarrow & S^{[\mathbf{Set},\mathbf{Set}]_{f, int_0}}\text{-}\mathbf{alg} & \longrightarrow & S^{[\mathbf{Set},\mathbf{Set}]_{\aleph_1, int_0}}\text{-}\mathbf{alg} \\ \downarrow & & \llcorner \downarrow \lrcorner & & \downarrow \\ \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_f & \longleftarrow & \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{f, int_0} & \longrightarrow & \mathbf{Mon}[\mathbf{Set}, \mathbf{Set}]_{\aleph_1, int_0} \end{array}$$

5. Strength-based interpretation of the binding conditions

In the previous section, we have compared the category S - **DBAlg** of models of a binding signature S in De Bruijn monads with the usual category of Φ_S-monoids [FPT99]. In fact, the latter approach is much more general, in the sense that it does not only work for binding signatures but for so-called **pointed strong** endofunctors [FPT99], and in fact also for the more general **structurally strong** endofunctors introduced by Borthelle et al. [BHL20].

In this section, we show that De Bruijn algebras also generalise from binding signatures to structurally strong endofunctors, in the following sense. To any binding signature S, we associate such an endofunctor, say Σ_S, such that Σ_S - **Mon** $\cong S$ - **DBAlg**, where Σ_S - **Mon** is as defined for any structurally strong endofunctor by Borthelle et al.

This way we give a categorical status to binding signatures, as particular structurally strong endofunctors on **Set**.

Remark 5.1. We do not (yet) prove existence of an initial Σ-De Bruijn algebras for any larger class of endofunctors than those of the form Σ_S.

Remark 5.2. We resort to structurally strong endofunctors because pointed strong endofunctors live on monoidal categories [FPT99, Fio08], while we have seen in Corollary 3.12 that our tensor product merely equips **Set** with skew monoidal structure. (The very purpose of structurally strong endofunctors is to generalise pointed strong endofunctors to the skew monoidal case.) Following up on Remark 3.13, we could equivalently work with the monoidal category $[\mathbf{B}[\mathbb{N}, \mathbb{N}], \mathbf{Set}]$, in which the machinery of pointed strong endofunctors applies.

Remark 5.3. In fact, the isomorphism Σ_S - **Mon** $\cong S$ - **DBAlg** is almost an equality, since the only difference lies in the difference between a family $(M^{|ar(o)|} \to M)_{o \in O}$ of operations and its cotupling $\sum_{o \in O} M^{|ar(o)|} \to M$: one could easily adjust the presentation to get an exact match.

The starting point is that the endofunctor Σ_S associated to any given binding signature S may be equipped with a family of maps

$$\mathbf{dbs}_S \colon \Sigma_S(X) \otimes Y \to \Sigma_S(X \otimes Y)$$

that will be used to specify how substitution commutes with the operations of S. However, in order for such a map to be well-defined for binding operations, we need to assume that Y features variables and renaming, i.e., that it is a **pointed** $\mathbb{N}$**-module**. Moreover, this map should satisfy some compatibility laws. These definitions and conditions are detailed in §5.1, where we furthermore recall structurally strong endofunctors. In §5.2, we interpret binding signatures as such endofunctors, we recall the category of Σ-monoids, for any structurally strong endofunctor Σ, and establish the announced isomorphism of categories.

5.1. Structural strengths.

We start by introducing a notion of set equipped with variables and renamings, in Definition 5.7 below. Recalling from Example 2.4 that $\mathbb{N}$ forms a De Bruijn monad, we have:

Definition 5.4. An $\mathbb{N}$**-module** is a set X equipped with an action of the monoid $\mathbb{N}^{\mathbb{N}}$, namely a map $r \colon X \times \mathbb{N}^{\mathbb{N}} = X \otimes \mathbb{N} \to X$, making the following diagrams commute.

$$
\begin{array}{ccccc}
(X \otimes \mathbb{N}) \otimes \mathbb{N} & \xrightarrow{\alpha_{X,\mathbb{N},\mathbb{N}}} & X \otimes (\mathbb{N} \otimes \mathbb{N}) \\
\downarrow{\scriptstyle r \otimes \mathbb{N}} & & \downarrow{\scriptstyle X \otimes \lambda_{\mathbb{N}}} \\
X \otimes \mathbb{N} & & X \otimes \mathbb{N} \\
& \searrow^{r} \quad X \quad {}^{r}\swarrow &
\end{array}
\qquad
\begin{array}{ccc}
X & \xrightarrow{\rho_X} & X \otimes \mathbb{N} \\
& \searrow^{=} \quad X \quad {}^{r}\swarrow &
\end{array}
$$

A **morphism of $\mathbb{N}$-modules** $(X, r) \to (Y, s)$ is a map $f : X \to Y$ between underlying sets commuting with action, i.e., making the following square commute.

$$
\begin{CD}
X \otimes \mathbb{N} @>{f \otimes \mathbb{N}}>> Y \otimes \mathbb{N} \\
@V{r}VV @VV{s}V \\
X @>>{f}> Y
\end{CD}
$$

Finally, $\mathbb{N}$-modules and morphisms between them form a category, which we denote by $\mathbb{N}\text{-}\mathbf{Mod}$.

Notation 5.5. We generally denote $r(x, f)$ by $x[f]_r$, or merely $x[f]$ when r is clear from context.

Example 5.6. Any De Bruijn monad (X, s, v) (in particular $(\mathbb{N}, \lambda, \mathrm{id})$ itself) has a canonical structure of $\mathbb{N}$-module given by $r(x, f) = x[v \circ f]_s$.

Definition 5.7.

- A **pointed $\mathbb{N}$-module** is an $\mathbb{N}$-module (X, r), equipped with a map $v : \mathbb{N} \to X$ which is a morphism of $\mathbb{N}$-modules, i.e., such that the following square commutes.

$$
\begin{CD}
\mathbb{N} \otimes \mathbb{N} @>{v \otimes \mathbb{N}}>> X \otimes \mathbb{N} \\
@V{\lambda_{\mathbb{N}}}VV @VV{r}V \\
\mathbb{N} @>>{v}> X
\end{CD}
$$

- A morphism of pointed $\mathbb{N}$-modules $(X, r, v) \to (Y, s, w)$ is a morphism of $\mathbb{N}$-modules $f : (X, r) \to (Y, s)$ commuting with point, i.e., such that the following triangle commutes.

$$
\begin{array}{ccc}
& \mathbb{N} & \\
{}^{v}\swarrow & & \searrow^{w} \\
X & \xrightarrow[f]{} & Y
\end{array}
$$

- Let $\mathbb{N}\text{-}\mathbf{Mod}_{\mathbb{N}}$ denote the category of pointed $\mathbb{N}$-modules.

Remark 5.8. Equivalently, $\mathbb{N}\text{-}\mathbf{Mod}_{\mathbb{N}}$ is the coslice category $\mathbb{N}/(\mathbb{N}\text{-}\mathbf{Mod})$.

Example 5.9. The canonical $\mathbb{N}$-module structure of any De Bruijn monad (X, s, v) (in particular $(\mathbb{N}, \lambda, \mathrm{id})$ itself), described in Example 5.6, is in fact pointed, by the map $v : \mathbb{N} \to X$.

We now define a tensor product on (pointed) $\mathbb{N}$-modules, following [LS14, (8.1)].

Definition 5.10. Given an $\mathbb{N}$-module (X, r) and a set Y, let $X \boxtimes Y$ denote the following coequaliser in $\mathbf{Set}$.

$$
(X \otimes \mathbb{N}) \otimes Y \overset{(X \otimes \lambda_Y) \circ \alpha_{X,\mathbb{N},Y}}{\underset{r \otimes Y}{\rightrightarrows}} X \otimes Y \xrightarrow{\kappa_{X,Y}} X \boxtimes Y \tag{5.1}
$$

Notation 5.11. Concretely, $X \boxtimes Y$ is the set of equivalence classes of pairs $(x, \upsilon) \in X \times Y^{\mathbb{N}}$, modulo the equation $(x[\rho], \upsilon) = (x, \upsilon \circ \rho)$, for any $\rho \colon \mathbb{N} \to \mathbb{N}$. We denote such an equivalence class by $x(\!|\upsilon|\!)$, and extend the notation to $\sigma(\!|\upsilon|\!)$, for any assignment $\sigma \colon \mathbb{N} \to X$, i.e., $\sigma(\!|\upsilon|\!)(n) = \sigma(n)(\!|\upsilon|\!)$ for all n.

Proposition 5.12 [LS14, Theorem 8.1]**.** *When Y is equipped with $\mathbb{N}$-module structure, $X \boxtimes Y$ admits a canonical $\mathbb{N}$-module structure, such that $x(\!|\upsilon|\!)[f] = x(\!|\upsilon[f]|\!)$. This makes the category $\mathbb{N}$-**Mod** of $\mathbb{N}$-modules into a skew monoidal category (with unit $\mathbb{N}$, and invertible right unitor). Furthermore, the forgetful functor is monoidal, and creates monoids in the sense that monoids are the same in $\mathbb{N}$-**Mod** and in **Set**.*

Proof. To apply [LS14, Theorem 8.1], we need to prove that tensoring on the right in **Set** preserves reflexive coequalisers, which holds by interchange of colimits since $X \otimes Y = X \times Y^{\mathbb{N}}$ is the $Y^{\mathbb{N}}$-fold coproduct of X with itself. □

In fact, this extends to $\mathbb{N}$-$\mathbf{Mod}_{\mathbb{N}}$:

Proposition 5.13. *Given pointed $\mathbb{N}$-modules (X, r, v) and (Y, s, w), the $\mathbb{N}$-module $X \boxtimes Y$ is canonically pointed by the map*

$$\mathbb{N} \xrightarrow{\rho_{\mathbb{N}}} \mathbb{N} \otimes \mathbb{N} \xrightarrow{v \otimes w} X \otimes Y \xrightarrow{\kappa_{X,Y}} X \boxtimes Y.$$

Proof. This result was proved and formalised in a general skew monoidal setting in [BHL20], see [Laf22b, IModules.PtIModule_tensor]. □

Now that we have defined the tensor product of pointed $\mathbb{N}$-modules, we may introduce structural strengths.

Definition 5.14 [BHL20, Definition 2.11]**.** A **structural strength** on an endofunctor $\Sigma \colon \mathbf{Set} \to \mathbf{Set}$ is a natural transformation $st_{X,Y} \colon \Sigma(X) \otimes Y \to \Sigma(X \otimes Y)$, where X is any set and Y is a pointed $\mathbb{N}$-module, making the following diagrams commute,

$$\begin{array}{ccc} & \Sigma(A) & \\ {}^{\rho_{\Sigma(A)}}\swarrow & & \searrow^{\Sigma(\rho_A)} \\ \Sigma(A) \otimes \mathbb{N} & \xrightarrow[st_{A,\mathbb{N}}]{} & \Sigma(A \otimes \mathbb{N}) \end{array}$$

$$\begin{array}{ccccc} (\Sigma(A) \otimes X) \otimes Y & \xrightarrow{st_{A,X} \otimes Y} & \Sigma(A \otimes X) \otimes Y & \xrightarrow{st_{A \otimes X, Y}} & \Sigma((A \otimes X) \otimes Y) \\ {\scriptstyle \alpha'_{\Sigma(A),X,Y}} \downarrow & & & & \downarrow {\scriptstyle \Sigma(\alpha'_{A,X,Y})} \\ \Sigma(A) \otimes (X \boxtimes Y) & & \xrightarrow[st_{A, X \boxtimes Y}]{} & & \Sigma(A \otimes (X \boxtimes Y)) \end{array}$$

where $\alpha'_{A,X,Y}$ is $(A \otimes \kappa_{X,Y}) \circ \alpha_{A,X,Y}$, for any A.

Remark 5.15. In examples, the first axiom will entail that the "identity" assignment should cross operations unchanged. In terms of De Bruijn monads, the "identity" assignment is merely the variables map v, so, e.g., in the setting of Example 2.5, the axiom boils down to the fact that lifting v yields v again: $\Uparrow v = v$. The second axiom will entail the substitution lemma $x[\sigma][\phi] = x[\sigma[\phi]]$, where we recall that by definition $\sigma[\phi](n) = \sigma(n)[\phi]$. E.g., if crossing a given unary operation o maps assignments σ to σ', the axiom says that

$(\sigma[\phi])' = \sigma'[\phi']$. This is just what is needed for a proof by induction to go through, as in

$$\begin{array}{rcll} o(x)[\sigma][\phi] & = & o(x[\sigma'][\phi']) & \text{by the assumed binding condition} \\ & = & o(x[\sigma'[\phi']]) & \text{by induction hypothesis} \\ & = & o(x[(\sigma[\phi])']) & \text{by the second axiom} \\ & = & o(x)[\sigma[\phi]] & \text{by the binding condition again.} \end{array}$$

The technique extends to operations with more complex arities.

5.2. **De Bruijn algebras as Σ-monoids.** Let us now interpret binding signatures as structurally strong endofunctors, and show that the corresponding category of models coincides with De Bruijn algebras.

We can readily equip the endofunctor Σ_S associated to any binding signature S (Definition 2.17) with a structural strength $\mathbf{dbs}_S$, which we call the **De Bruijn** strength prescribed by S on Σ_S.

Remark 5.16. Let us recall that by definition, Σ_S ignores the binding information in S: we have

$$\Sigma_S(X) = \sum_{o \in O} X^{p_o},$$

where $ar(o) = (n_1^o, \ldots, n_{p_o}^o)$ for all $o \in O$.

In order to define $\mathbf{dbs}_S$, we start by adapting the definition of assignment lifting (Definition 2.6) to pointed $\mathbb{N}$-modules.

Definition 5.17. Let (Y, r, v) be a pointed $\mathbb{N}$-module. For any assignment $\sigma \colon \mathbb{N} \to Y$, we define the assignment $\Uparrow\sigma \colon \mathbb{N} \to Y$ by

$$\begin{array}{rcl} (\Uparrow\sigma)(0) & = & v(0) \\ (\Uparrow\sigma)(n+1) & = & \sigma(n)[\uparrow], \end{array}$$

where $\uparrow \colon \mathbb{N} \to \mathbb{N}$ denotes the successor map. Let $\Uparrow^0 A = A$, and $\Uparrow^{n+1} A = \Uparrow(\Uparrow^n A)$.

Using this, let us now define the De Bruijn strength of the identity functor. We will then iterate the process to show that each iterated lifting also equips the identity functor with a structural strength. Finally, we will use this as a basis for equipping the endofunctor Σ_S associated with any binding signature S, with a structural strength.

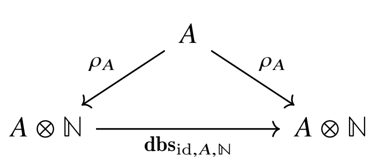
Definition 5.18. The **first De Bruijn strength** of the identity functor is the map

$$\begin{array}{rccl} \mathbf{dbs}_{\mathrm{id},X,Y} \colon & X \otimes Y & \to & X \otimes Y \\ & (x, \sigma) & \mapsto & (x, \Uparrow\sigma), \end{array}$$

defined for all $X \in \mathbf{Set}$ and $Y \in \mathbb{N}\text{-}\mathbf{Mod}_{\mathbb{N}}$.

Proposition 5.19. *The first De Bruijn strength is a structural strength on the identity functor.*

Proof. We first check commutation with the right unitor, in this case



This triangle commutes because any $a \in A$ is mapped by ρ_A to (a, id), and then by $\mathbf{dbs}_{\mathrm{id},A,\mathbb{N}}$ to $(a, \Uparrow \mathrm{id})$. But $\Uparrow \mathrm{id} = \mathrm{id}$, hence the result.

For commutation with the associator,

$$\begin{array}{ccccc}
(A \otimes X) \otimes Y & \xrightarrow{\mathbf{dbs}_{\mathrm{id},A,X\otimes Y}} & (A \otimes X) \otimes Y & \xrightarrow{\mathbf{dbs}_{\mathrm{id},A\otimes X,Y}} & (A \otimes X) \otimes Y \\
{\scriptstyle \alpha_{A,X,Y}} \downarrow & & & & \downarrow {\scriptstyle \alpha_{A,X,Y}} \\
A \otimes (X \otimes Y) & & & & A \otimes (X \otimes Y) \\
{\scriptstyle A\otimes\kappa_{X,Y}} \downarrow & & & & \downarrow {\scriptstyle A\otimes\kappa_{X,Y}} \\
A \otimes (X \boxtimes Y) & & \xrightarrow[\mathbf{dbs}_{\mathrm{id},A,X\boxtimes Y}]{} & & A \otimes (X \boxtimes Y)
\end{array}$$

we observe that any triple $(a, \sigma, \upsilon) \in (A \otimes X) \otimes Y$ is mapped by the bottom left composite to

$$(a, \Uparrow(\sigma[\![\upsilon]\!])),$$

where we define $\sigma[\![\upsilon]\!](n) := \sigma(n)[\![\upsilon]\!]$. Furthermore, (a, σ, υ) is mapped by the top right composite to

$$(a, (\Uparrow\sigma)[\![\Uparrow\upsilon]\!]),$$

so we are left with the task of proving

$$\Uparrow(\sigma[\![\upsilon]\!]) = (\Uparrow\sigma)[\![\Uparrow\upsilon]\!].$$

Let $u \colon \mathbb{N} \to X$ and $v : \mathbb{N} \to Y$ denote the points of X and Y. We proceed by case analysis:

- At 0, we have:

$$\begin{array}{rcl}
\Uparrow(\sigma[\![\upsilon]\!])(0) & = & v(0) \\
& = & (\Uparrow\upsilon)(0) \\
& = & u(0)[\![\Uparrow\upsilon]\!] \\
& = & (\Uparrow\sigma)(0)[\![\Uparrow\upsilon]\!] \\
& = & (\Uparrow\sigma)[\![\Uparrow\upsilon]\!](0).
\end{array}$$

- At any $n + 1$, we have

$$\begin{array}{rcl}
\Uparrow(\sigma[\![\upsilon]\!])(n+1) & = & \sigma[\![\upsilon]\!](n)[\uparrow] \\
& = & \sigma(n)[\![\upsilon]\!][\uparrow] \\
& = & \sigma(n)[\![\upsilon[\uparrow]]\!],
\end{array}$$

  where by definition $\upsilon[\uparrow](n) = \upsilon(n)[\uparrow]$.

$$\begin{array}{rcl}
(\Uparrow\sigma)[\![\Uparrow\upsilon]\!](n+1) & = & (\Uparrow\sigma)(n+1)[\![\Uparrow\upsilon]\!] \\
& = & \sigma(n)[\uparrow][\![\Uparrow\upsilon]\!] \\
& = & \sigma(n)[\![(\Uparrow\upsilon)\circ\uparrow]\!].
\end{array}$$

  But we have $(\Uparrow\upsilon)\circ\uparrow = \upsilon[\uparrow]$ since, for all $p \in \mathbb{N}$, we have

$$((\Uparrow\upsilon)\circ\uparrow)(p) = (\Uparrow\upsilon)(p+1) = \upsilon(p)[\uparrow] = \upsilon[\uparrow](p).$$ □

Furthermore, we have:

Proposition 5.20. *Structurally strong endofunctors compose, in the sense that if F and G are structurally strong endofunctors, then so is $G \circ F$, with structural strength given by the composite*

$$G(F(X)) \otimes Y \to G(F(X) \otimes Y) \to G(F(X \otimes Y)). \tag{5.2}$$

Proof. For the first axiom, we have

$$\begin{array}{ccccc} & & G(F(A)) & & \\ & \swarrow^{\rho_{G(F(A))}} & \downarrow{\scriptstyle G(\rho_{F(A)})} & \searrow^{G(F(\rho_A))} & \\ G(F(A))\otimes\mathbb{N} & \xrightarrow[st^G_{F(A),\mathbb{N}}]{} & G(F(A)\otimes\mathbb{N}) & \xrightarrow[G(st^F_{A,\mathbb{N}})]{} & G(F(A\otimes\mathbb{N})). \end{array}$$

The second axiom holds by chasing the following diagram.

$$\begin{array}{ccccc} G(F(A))\otimes X\otimes Y & \xrightarrow{st^G_{F(A),X}\otimes Y} & G(F(A)\otimes X)\otimes Y & \xrightarrow{G(st^F_{A,X})\otimes Y} & G(F(A\otimes X))\otimes Y \\ \Big\downarrow{\scriptstyle \alpha} & & {\scriptstyle st^G_{F(A)\otimes X,Y}}\downarrow & & \downarrow{\scriptstyle st^G_{F(A\otimes X),Y}} \\ & & G(F(A)\otimes X\otimes Y) & \xrightarrow[G(st^F_{A,X\otimes Y})]{} & G(F(A\otimes X)\otimes Y) \\ & & {\scriptstyle G(\alpha)}\downarrow & & \downarrow{\scriptstyle G(st^F_{A\otimes X,Y})} \\ G(F(A))\otimes(X\otimes Y) & & G(F(A)\otimes(X\otimes Y)) & & G(F(A\otimes X\otimes Y)) \\ & & & & \downarrow{\scriptstyle G(F(\alpha))} \\ \Big\downarrow{\scriptstyle G(F(A))\otimes\kappa_{X,Y}} & & {\scriptstyle G(F(A)\otimes\kappa_{X,Y})}\Big\downarrow & & G(F(A\otimes(X\otimes Y))) \\ & & & & \downarrow{\scriptstyle G(F(A\otimes\kappa_{X,Y}))} \\ G(F(A))\otimes(X\boxtimes Y) & \xrightarrow[st^G_{F(A),X\boxtimes Y}]{} & G(F(A)\otimes(X\boxtimes Y)) & \xrightarrow[G(st^F_{A,X\boxtimes Y})]{} & G(F(A\otimes(X\boxtimes Y))) \end{array}$$

□

Combining the last two results, any $\mathrm{id}^n = \mathrm{id}$ is structurally strong, with the following strength, obtained by inductively unfolding (5.2):

Definition 5.21. Let the n**th De Bruijn strength** of the identity functor, $\mathbf{dbs}^n$, be defined by $\mathbf{dbs}^n_{\mathrm{id},X,Y}(x,\sigma) = (x, \Uparrow^n \sigma)$.

In summary:

Proposition 5.22. *Each* $\mathbf{dbs}^n$ *is a structural strength on the identity functor.*

Let us now extend this to general binding arities:

Proposition 5.23. *Given structurally strong endofunctors* (F, st^F) *and* (G, st^G)*, the pointwise product* $F \times G$ *admits the structural strength defined at any* $X \in \mathbf{C}$ *and* $Y \in \mathbb{N}\text{-}\mathbf{Mod}_{\mathbb{N}}$ *by the composite*

$$(F(X)\times G(X))\otimes Y \xrightarrow{\langle \pi_1\otimes Y, \pi_2\otimes Y\rangle} (F(X)\otimes Y)\times(G(X)\otimes Y) \xrightarrow{st^F_{X,Y}\times st^G_{X,Y}} F(X\otimes Y)\times G(X\otimes Y). \quad (5.3)$$

Proof. The first axiom holds by chasing the following diagram.

$$\begin{array}{ccccc} & & F(X)\times G(X) & & \\ & \swarrow^{\rho_{F(X)\times G(X)}} & \downarrow{\scriptstyle \rho_{F(X)}\times\rho_{G(X)}} & \searrow^{F(\rho_X)\times G(\rho_X)} & \\ (F(X)\times G(X))\otimes\mathbb{N} & \xrightarrow[\langle\pi_1\otimes\mathbb{N},\pi_2\otimes\mathbb{N}\rangle]{} & (F(X)\otimes\mathbb{N})\times(G(X)\otimes\mathbb{N}) & \xrightarrow[st^F_{X,\mathbb{N}}\times st^G_{X,\mathbb{N}}]{} & F(X\otimes\mathbb{N})\times G(X\otimes\mathbb{N}) \end{array}$$

For the second axiom, we need to prove that the following diagram commutes.

$$
\begin{array}{ccc}
(F(X)\times G(X))\otimes Y\otimes Z & \xrightarrow{\langle \pi_1\otimes Y,\pi_2\otimes Y\rangle\otimes Z} & ((F(X)\otimes Y)\times(G(X)\otimes Y))\otimes Z \\
\alpha\downarrow & & \downarrow (st^F_{X,Y}\times st^G_{X,Y})\otimes Z \\
(F(X)\times G(X))\otimes(Y\otimes Z) & & (F(X\otimes Y)\times G(X\otimes Y))\otimes Z\\
(F(X)\times G(X))\otimes\kappa_{Y,Z}\downarrow & & \downarrow\langle\pi_1\otimes Z,\pi_2\otimes Z\rangle\\
 & & (F(X\otimes Y)\otimes Z)\times(G(X\otimes Y)\otimes Z)\\
 & & \downarrow st^F_{X\otimes Y,Z}\times st^G_{X\otimes Y,Z}\\
(F(X)\times G(X))\otimes(Y\boxtimes Z) & & F(X\otimes Y\otimes Z)\times G(X\otimes Y\otimes Z)\\
\langle\pi_1\otimes(Y\boxtimes Z),\pi_2\otimes(Y\boxtimes Z)\rangle\downarrow & & \downarrow F(\alpha_{X,Y,Z})\times G(\alpha_{X,Y,Z})\\
 & & F(X\otimes(Y\otimes Z))\times G(X\otimes(Y\otimes Z))\\
 & & \downarrow F(X\otimes\kappa_{Y,Z})\times G(X\otimes\kappa_{Y,Z})\\
(F(X)\otimes(Y\boxtimes Z))\times(G(X)\otimes(Y\boxtimes Z)) & \xrightarrow[st^F_{X,Y\boxtimes Z}\times st^G_{X,Y\boxtimes Z}]{} & F(X\otimes(Y\boxtimes Z))\times G(X\otimes(Y\boxtimes Z))
\end{array}
$$

For this, since the target is a product, we proceed componentwise, and by symmetry it suffices to check the first:

$$
\begin{array}{ccccc}
(F(X)\times G(X))\otimes Y\otimes Z & \xrightarrow{\pi_1\otimes Y\otimes Z} & F(X)\otimes Y\otimes Z & \xrightarrow{st^F_{X,Y}\otimes Z} & F(X\otimes Y)\otimes Z\\
 & & & & \downarrow st^F_{X\otimes Y,Z}\\
\alpha\downarrow & & \downarrow\alpha & & F(X\otimes Y\otimes Z)\\
 & & & & \downarrow F(\alpha_{X,Y,Z})\\
(F(X)\times G(X))\otimes(Y\otimes Z) & \xrightarrow{\pi_1\otimes(Y\otimes Z)} & F(X)\otimes(Y\otimes Z) & & F(X\otimes(Y\otimes Z))\\
(F(X)\times G(X))\otimes\kappa_{Y,Z}\downarrow & & \downarrow F(X)\otimes\kappa_{Y,Z} & & \downarrow F(X\otimes\kappa_{Y,Z})\\
(F(X)\times G(X))\otimes(Y\boxtimes Z) & \xrightarrow[\pi_1\otimes(Y\boxtimes Z)]{} & F(X)\otimes(Y\boxtimes Z) & \xrightarrow[st^F_{X,Y\boxtimes Z}]{} & F(X\otimes(Y\boxtimes Z)).
\end{array}
$$

□

Corollary 5.24. *For any binding arity* $a = (n_1, \ldots, n_p)$*, the family*

$$
\begin{array}{rcl}
\mathbf{dbs}_{a,X,Y}\colon X^p\otimes Y & \to & (X\otimes Y)^p\\
((x_1,\ldots,x_p),\sigma) & \mapsto & ((x_1,\Uparrow^{n_1}\sigma),\ldots,(x_p,\Uparrow^{n_p}\sigma)),
\end{array}
$$

for all sets X *and pointed* $\mathbb{N}$*-modules* Y*, defines a structural strength on the endofunctor* $X\mapsto X^p$*, which we call the* ***De Bruijn strength*** $\mathbf{dbs}_a$ *of* a*.*

Proof. By inductively unfolding (5.3) using Proposition 5.22. □

As promised, let us now express the binding condition in terms of strengths:

Proposition 5.25. *For any binding arity* $a = (n_1, \ldots, n_p)$ *and De Bruijn monad* (M, s, v)*, a map* $o\colon M^p \to M$ *satisfies the* a*-binding condition w.r.t.* (s, v) *iff the following pentagon commutes.*

$$
\begin{array}{ccccc}
X^p\otimes X & \xrightarrow{\mathbf{dbs}_{a,X,X}} & (X\otimes X)^p & \xrightarrow{s^p} & X^p\\
o\otimes X\downarrow & & & & \downarrow o\\
X\otimes X & & \xrightarrow[s]{} & & X
\end{array}
\tag{5.4}
$$

Proof. The bottom left composite maps any tuple $((x_1, \ldots, x_p), \sigma)$ to $o(x_1, \ldots, x_p)[\sigma]$, while the top right one maps it first to

$$((x_1, \Uparrow^{n_1} \sigma), \ldots, (x_p, \Uparrow^{n_p} \sigma)),$$

then to

$$(x_1[\Uparrow^{n_1} \sigma], \ldots, x_p[\Uparrow^{n_p} \sigma]),$$

and finally to

$$o(x_1[\Uparrow^{n_1} \sigma], \ldots, x_p[\Uparrow^{n_p} \sigma]),$$

as desired. □

At last, let us now define the De Bruijn strength of the endofunctor Σ_S induced by an arbitrary binding signature S. For this, just as we have shown that structurally strong endofunctors are closed under products (Proposition 5.23), we start by showing that they are closed under coproducts.

Proposition 5.26. *Given structurally strong endofunctors $(F_i, st^i)_{i \in I}$, the pointwise coproduct $\sum_i F_i$ admits the structural strength defined at any $X \in \mathbf{C}$ and $Y \in I\text{-}\mathbf{Mod}_I$ by the composite*

$$\Big(\sum_i F_i(X)\Big) \otimes Y \cong \sum_i (F_i(X) \otimes Y) \xrightarrow{\sum_i st^i_{X,Y}} \sum_i F_i(X \otimes Y).$$

Proof. The first axiom holds by chasing the following diagram.

$$\begin{array}{ccccc}
 & & \sum_i F_i(X) & & \\
 & \swarrow^{\rho_{\sum_i F_i(X)}} & \downarrow \sum_i \rho_{F_i(X)} & \searrow^{\sum_i F_i(\rho_X)} & \\
(\sum_i F_i(X)) \otimes \mathbb{N} & \xrightarrow[\cong]{} & \sum_i (F_i(X) \otimes \mathbb{N}) & \xrightarrow[\sum_i st^i_{X,\mathbb{N}}]{} & \sum_i F_i(X \otimes \mathbb{N})
\end{array}$$

The second axiom holds by chasing the diagram in Figure 1. □

Let us finally put things together:

Corollary 5.27. *For any binding signature $S = (O, ar)$, the endofunctor Σ_S induced by S admits as structural strength the composite*

$$(\sum_o X^{|ar(o)|}) \otimes Y \cong \sum_o (X^{|ar(o)|} \otimes Y) \xrightarrow{\sum_o \mathbf{dbs}_{ar(o),X,Y}} \sum_o (X \otimes Y)^{|ar(o)|},$$

or more concretely

$$\begin{array}{rcl}
\Sigma_S(X) \otimes Y & \to & \Sigma_S(X \otimes Y) \\
((o, (x_1, \ldots, x_{p_o})), \sigma) & \mapsto & (o, ((x_1, \Uparrow^{n_1} \sigma), \ldots, (x_{p_o}, \Uparrow^{n_{p_o}} \sigma))),
\end{array}$$

for all sets X and pointed $\mathbb{N}$-modules Y, where $ar(o) = (n_1, \ldots, n_{p_o})$.

*We call this the **De Bruijn strength** $\mathbf{dbs}_S$ of Σ_S.*

Proof. Recalling that, by Definition 2.17, we have $\Sigma_S(X) = \sum_{o \in O} X^{|ar(o)|}$, Σ_S is a coproduct of functors $X \mapsto X^{|ar(o)|}$ with structural strengths $\mathbf{dbs}_{ar(o)}$ by Corollary 5.24, hence admits the given structural strength by Proposition 5.26. □

In order to relate the initial-algebra semantics of §3 to the strength-based approach of [FPT99, Fio08], let us recall the definition of models, following the generalisation to the skew monoidal setting [BHL20].

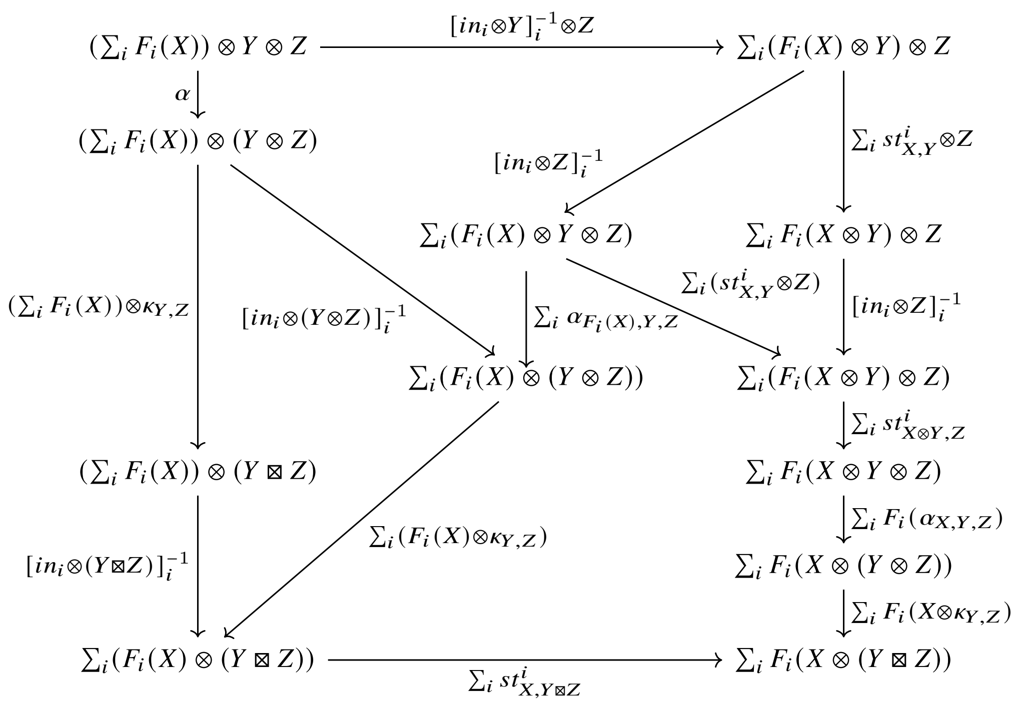


Figure 1: Diagram chasing for the proof of Proposition 5.26.

Definition 5.28. Given an endofunctor Σ with structural strength st, a **Σ-monoid** is an object X, equipped with monoid and Σ-algebra structures, say $s\colon X \otimes X \to X$, $v\colon \mathbb{N} \to X$, and $a\colon \Sigma(X) \to X$, making the following pentagon commute.

$$\begin{array}{ccccc} \Sigma(X) \otimes X & \xrightarrow{st_{X,X}} & \Sigma(X \otimes X) & \xrightarrow{\Sigma(s)} & \Sigma(X) \\ {\scriptstyle a \otimes X}\downarrow & & & & \downarrow{\scriptstyle a} \\ X \otimes X & & \xrightarrow[s]{\qquad\qquad} & & X \end{array} \tag{5.5}$$

A morphism of Σ-monoids is a map which is both a monoid and a Σ-algebra morphism.

Let Σ - **Mon** denote the category of Σ-monoids and morphisms between them.

We may at last relate the initial-algebra semantics of §3 with the strength-based approach:

Proposition 5.29. *For any binding signature $S = (O, ar)$ and De Bruijn monad (M, s, v) equipped with a map $o_M\colon M^p \to M$ for all $o \in O$ with $ar(o) = (n_1, \ldots, n_p)$, the following are equivalent:*

(i) *each map $o_M\colon M^p \to M$ satisfies the **a-binding condition** w.r.t. (s, v);*
(ii) *the corresponding map $\Sigma_S M \to M$ renders the pentagon* (5.5) *(with $\Sigma := \Sigma_S$ and $st :=$ **dbs**$_S$) commutative.*

Proof. By universal property of coproduct and distributivity, the pentagon (5.5) commutes iff each corresponding pentagon (5.4) does, which holds iff each o satisfies the $ar(o)$-binding condition w.r.t. (s, v) by Proposition 5.25. □

Corollary 5.30. *For any binding signature S, we have an isomorphism Σ_S -* **Mon** $\cong$ *S -* **DBAlg** *of categories over* **DBMnd**.

This readily entails the following (bundled) reformulation of Theorems 2.21 and 3.16.

Corollary 5.31. *Consider any binding signature* $S = (O, ar)$*, and let* DB_S *denote the initial* $(\mathbb{N} + \Sigma_S)$*-algebra, with structure maps* $v\colon \mathbb{N} \to \mathrm{DB}_S$ *and* $a\colon \Sigma_S(\mathrm{DB}_S) \to \mathrm{DB}_S$*. Then:*

- (i) *There exists a unique substitution map* $s\colon \mathrm{DB}_S \otimes \mathrm{DB}_S \to \mathrm{DB}_S$ *such that*
  - *the map* $\mathbb{N} \otimes \mathrm{DB}_S \xrightarrow{v \otimes \mathrm{DB}_S} \mathrm{DB}_S \otimes \mathrm{DB}_S \xrightarrow{s} \mathrm{DB}_S$ *coincides with the left unit of the skew monoidal structure* $(n, f) \mapsto f(n)$*, and*
  - *the pentagon* (5.5) *(with* $\Sigma := \Sigma_S$*) commutes.*
- (ii) *This substitution map turns* (DB_S, v, s, a) *into a* Σ_S*-monoid.*
- (iii) *This* Σ_S*-monoid is initial in* Σ_S - **Mon**.

Proof. Let **Mon**(**Set**) denote the category of monoids in **Set** for the skew monoidal structure. We have an equality **Mon**(**Set**) = **DBMnd** of categories, and the algebra structure $\Sigma_S(\mathrm{DB}_S) \to \mathrm{DB}_S$ is merely the cotupling of the maps o_{DB_S} of Theorem 2.21. This correspondence translates one statement into the other. □

Remark 5.32. This result hints at a potential push-button proof of Theorems 2.21 and 3.16 (and Corollary 5.31). Indeed, it is almost an instance of [BHL20, Theorem 2.15]: the latter is stated for general skew monoidal categories instead of merely **Set**, but does not directly apply in the present setting, because it assumes that the tensor product is finitary in the second argument. However, we expect the generalisation consisting in replacing this finitarity condition with α-accessibility to be straightforward.

6. Module-based interpretation of the binding conditions

In the previous section, we have shown that the construction of De Bruijn algebras generalises from binding signatures to structurally strong endofunctors, thus yielding a categorical status for binding signatures and a categorical interpretation of the binding conditions.

In this section, we give binding signatures an alternative categorical status, as **parametric modules** over De Bruijn monads, together with a corresponding categorical interpretation of the binding conditions. For this, we merely adapt to De Bruijn monads the treatment proposed for mere monads by Hirschowitz and Maggesi [HM07, HM10].

Compared to the original setting [HM07], a peculiarity is that **derivation** of modules, the module-theoretic incarnation of variable binding, preserves the underlying object. In other words, it only affects substitution.

In §6.1, we introduce modules over a De Bruijn monad. In §6.2, we define module derivation. We then introduce parametric De Bruijn modules in §6.3, and show how any binding signature S yields such a module M_S. Finally, in §6.4, we define the category M - **MAlg** of modular algebras of a parametric De Bruijn module M, and show that they provide an alternative categorical interpretation of the binding conditions by exhibiting an isomorphism S - **DBAlg** $\cong M_S$ - **MAlg** of categories over **DBMnd**.

6.1. Modules over De Bruijn monads and first-order signatures.

There is a general notion of module over a monoid in a monoidal (or skew monoidal) category; we just give the instance we are concerned with. Intuitively, if X is a De Bruijn monad, an X-module is a set that admits substitution of variables by elements of X:

Definition 6.1. For any De Bruijn monad (X, s, v), an X**-module** is a set A equipped with a **substitution** map, or **action**,

$$r \colon A \times X^{\mathbb{N}} \to A$$

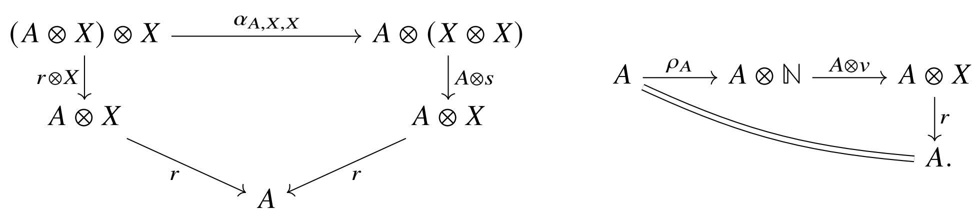
subject to the following condition, where we we use Notation 2.2:

for all $a \in A$ and $f, g \in X^{\mathbb{N}}$, we have $\quad a[f][g] = a[f[g]] \quad$ and $\quad a[v] = a.$

Remark 6.2. Please note that the first equation involves both substitution maps: $a[f]_A[g]_A = a[f[g]_X]_A$.

Remark 6.3. The definition is consistent with the definition of $\mathbb{N}$-modules (Definition 5.4), viewing $\mathbb{N}$ as a De Bruijn monad as in Example 2.4.

Remark 6.4. Equivalently, an X-module is an algebra for the monad $- \otimes X$, using the skew monoidal structure of Corollary 3.12. Indeed, the equations amount to commutation of the following diagrams,



which are exactly the equations for $(- \otimes X)$-algebras.

Let us now introduce a few basic constructions of modules:

Definition 6.5. Consider any De Bruijn monad X.

- The **tautological** X-module is X itself, with action $X \times X^{\mathbb{N}} \to X$ given by substitution.
- Given X-modules U and V, their **binary product** is $U \times V$, with action given by

$$\begin{aligned} U \times V \times X^{\mathbb{N}} &\to U \times V \\ (u, v, \sigma) &\mapsto (u[\sigma]_U, v[\sigma]_V). \end{aligned}$$

  This extends straightforwardly to small products.
- Given X-modules U and V, their **coproduct** is $U + V$, with action defined by case analysis:

$$\begin{array}{rcl} (U+V) \times X^{\mathbb{N}} & \to & U+V \\ (in_1(u), \sigma) & \mapsto & in_1(u[\sigma]_U) \\ (in_2(v), \sigma) & \mapsto & in_2(v[\sigma]_V). \end{array}$$

  This extends straightforwardly to small coproducts.

6.2. **Derivation of substitution for modules.** In this subsection, we explain module derivation. This operation does not change the carrier of the module, hence it acts on the substitution map only. In fact, it acts via the second argument of substitution, namely the assignment, as in §3 and §5.

Definition 6.6. Given a De Bruijn monad X, the **derivative** $A^{(1)}$ of an X-module A has the same carrier as A, with action given by

$$\begin{array}{rcl} A \times X^{\mathbb{N}} & \to & A \\ (a, \sigma) & \mapsto & a[\Uparrow\sigma], \end{array}$$

where $\Uparrow\sigma$ is as in Definition 2.6:

$$\begin{array}{rcl}(\Uparrow\sigma)(0) & = & v(0) \\ (\Uparrow\sigma)(n+1) & = & \sigma(n)[\uparrow],\end{array}$$

Of course we may iterate this operation:

Definition 6.7. Let $A^{(0)} = A$, and $A^{(n+1)} = (A^{(n)})^{(1)}$.

6.3. **Binding signatures as parametric modules.** In order to interpret binding signatures, we now introduce a parametric version of modules. For this, we construct a category **DBMod** whose objects are pairs of a De Bruijn monad and a module over it, and then define parametric modules as sections of the forgetful functor **DBMod** → **DBMnd**.

Definition 6.8. Let **DBMod** denote the category with

- as objects all pairs $(X, (U, a))$ of a De Bruijn monad X and an X-module (U, a), with $a : U \times X^{\mathbb{N}} \to U$, and
- as morphisms $(X, (U, a)) \to (Y, (V, b))$ all pairs (f, g), where $f : X \to Y$ is a De Bruijn monad morphism, and $g : U \to V$ is a map making the following diagram commute,

$$\begin{array}{ccc} U \times X^{\mathbb{N}} & \xrightarrow{g\times f^{\mathbb{N}}} & V \times Y^{\mathbb{N}} \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ U & \xrightarrow[g]{} & V \end{array}$$

  or equivalently, $g(u[\sigma]_U) = g(u)[f \circ \sigma]_V$, for all $u \in U$ and $\sigma : \mathbb{N} \to X$.

The **forgetful functor** $\mathcal{U}$: **DBMod** → **DBMnd** maps any $(X, (U, a))$ to X, and any (f, g) to f.

We now introduce parametric modules:

Definition 6.9. A **parametric De Bruijn module** is a section of the forgetful functor $\mathcal{U}$: **DBMod** → **DBMnd**, i.e., a functor M : **DBMnd** → **DBMod** such that $\mathcal{U} \circ M = \mathrm{id}_{\mathbf{DBMnd}}$.

Binding signatures naturally induce parametric De Bruijn modules:

Definition 6.10.

- The **tautological** parametric De Bruijn module, denoted by θ, maps any De Bruijn monad X to itself, with action $X \times X^{\mathbb{N}} \to X$ given by substitution.
- The **derivative** $U^{(1)}$ of a parametric De Bruijn module U is defined to map any De Bruijn monad X to $U(X)^{(1)}$, and any morphism $f : X \to Y$ to $(f, U(f))$. This works because the following square commutes.

$$\begin{array}{ccc} X^{\mathbb{N}} & \xrightarrow{f^{\mathbb{N}}} & Y^{\mathbb{N}} \\ {\scriptstyle \Uparrow}\downarrow & & \downarrow{\scriptstyle \Uparrow} \\ X^{\mathbb{N}} & \xrightarrow[f^{\mathbb{N}}]{} & Y^{\mathbb{N}} \end{array}$$

  Indeed, letting v_X and v_Y denote the respective variables maps of X and Y, we show by case analysis on $n \in \mathbb{N}$ that for all $\sigma : \mathbb{N} \to X$, we have $f^{\mathbb{N}}(\Uparrow\sigma)(n) = \Uparrow(f^{\mathbb{N}}(\sigma))(n)$:

– at 0, we have

$$\begin{aligned}
f^{\mathbb{N}}(\Uparrow\sigma)(0) &= f(\Uparrow\sigma(0)) \\
&= f(v_X(0)) \\
&= v_Y(0) \\
&= \Uparrow(f^{\mathbb{N}}(\sigma))(0),
\end{aligned}$$

– and at any $n+1$, we have

$$\begin{aligned}
f^{\mathbb{N}}(\Uparrow\sigma)(n+1) &= f(\Uparrow\sigma(n+1)) \\
&= f(\sigma(n)[\uparrow_X]) \\
&= f(\sigma(n))[f \circ \uparrow_X] \\
&= f(\sigma(n))[\uparrow_Y] \\
&= \Uparrow(f^{\mathbb{N}}(\sigma))(n+1).
\end{aligned}$$

- The **nth derivative** $U^{(n)}$ of a parametric De Bruijn module U is defined by induction: $U^{(0)} = U$ and $U^{(n+1)} = (U^{(n)})^{(1)}$.
- Given parametric De Bruijn modules U and V, their **binary product** maps any X to the X-module product $U(X) \times V(X)$. This extends straightforwardly to small products.
- The parametric De Bruijn module M_a induced by a binding arity $a = (n_1, \ldots, n_p)$ is the product $\prod_{i \in p} \theta^{(n_i)}$ of derivatives of the tautological parametric De Bruijn module.
- Given parametric De Bruijn modules U and V, their **coproduct** maps any X to the X-module coproduct $U(X) + V(X)$. This extends straightforwardly to small coproducts.
- The parametric De Bruijn module M_S induced by a binding signature $S = (O, ar)$ is the coproduct $\sum_{o \in O} M_{ar(o)}$ of the parametric De Bruijn modules induced by the arities of all operations.

6.4. **Interpreting the binding conditions.** In the previous subsection, we have interpreted binding signatures as parametric modules, but we have not yet defined the models of a parametric module. Let us do this now, and prove that, for any binding signature S, the category of De Bruijn S-algebras is isomorphic to the category of models of the induced parametric De Bruijn module M_S.

Definition 6.11.

- Given a parametric De Bruijn module U, a **U-algebra** is a De Bruijn monad X, equipped with an X-module morphism $\alpha \colon U(X) \to X$.
- For any U, given U-algebras (X, α) and (Y, β), a **U-algebra morphism** is a De Bruijn monad morphism $f \colon X \to Y$ making the following diagram commute,

$$\begin{array}{ccc}
U(X) & \xrightarrow{U(f)} & U(Y) \\
\alpha\big\downarrow & & \big\downarrow\beta \\
X & \xrightarrow[f]{} & Y
\end{array}$$

  or equivalently $f(\alpha(u)) = \beta(U(f)(u))$, for all $u \in U(X)$.
- For any U, U-algebras and morphisms between them form a category, which we denote by U-**MAlg**.
- The **forgetful functor** $\mathcal{U}^M \colon U\text{-}\mathbf{MAlg} \to \mathbf{DBMnd}$ maps any (X, α) to X.

As announced, let us prove

Proposition 6.12. *For any binding signature S, the categories S - **DBAlg** and M_S - **MAlg** are isomorphic over **DBMnd**.*

Proof. The key point is that for any binding arity $a = (n_1, \ldots, n_p)$, a map $o\colon X^p \to X$ is an operation of binding arity a iff it is an X-module morphism $\prod_{i=1}^{p} X^{(n_i)} \to X$. Indeed, the latter condition unfolds to the fact that, for any assignment $\sigma\colon \mathbb{N} \to X$ and tuple $(e_1, \ldots, e_p) \in X^p$, we have

$$o(e_1, \ldots, e_p)[\sigma] = o(e_1[\Uparrow^{n_1}\sigma], \ldots, e_p[\Uparrow^{n_p}\sigma]),$$

which is exactly the a-binding condition (2.1). □

We readily obtain the following (bundled) reformulation of Theorems 2.21 and 3.16.

Corollary 6.13. *Consider any binding signature $S = (O, ar)$, and let DB_S denote the initial $(\mathbb{N} + \Sigma_S)$-algebra, with structure maps $v\colon \mathbb{N} \to \mathrm{DB}_S$ and $a\colon \Sigma_S(\mathrm{DB}_S) \to \mathrm{DB}_S$. Then:*

(i) *There exists a unique substitution map $s\colon \mathrm{DB}_S \otimes \mathrm{DB}_S \to \mathrm{DB}_S$ such that*
  - *the map $\mathbb{N} \otimes \mathrm{DB}_S \xrightarrow{v \otimes \mathrm{DB}_S} \mathrm{DB}_S \otimes \mathrm{DB}_S \xrightarrow{s} \mathrm{DB}_S$ coincides with the left unit of the skew monoidal structure $(n, f) \mapsto f(n)$, and*
  - *a is an X-module morphism.*

(ii) *This substitution map turns (DB_S, v, s, a) into an M_S-algebra.*

(iii) *This M_S-algebra is initial in M_S - **MAlg**.*

7. Simply-typed extension

In this section, we extend the framework of §2–3, which is untyped, to the simply-typed case. The development essentially follows the same pattern, replacing sets with families.

We fix in the whole section a set $\mathbb{T}$ of **types**, and call **$\mathbb{T}$-sets** the objects of $\mathbf{Set}^{\mathbb{T}}$. A morphism $X \to Y$ is a family $(X(\tau) \to Y(\tau))_{\tau \in \mathbb{T}}$ of maps.

7.1. De Bruijn $\mathbb{T}$-monads.

In this subsection, we define the typed analogue of De Bruijn monads.

The role of $\mathbb{N}$ will be played in the typed context by the following $\mathbb{T}$-set.

Definition 7.1. Let $\mathbf{N} \in \mathbf{Set}^{\mathbb{T}}$ be defined by $\mathbf{N}(\tau) = \mathbb{N}$.

Definition 7.2. Given a $\mathbb{T}$-set X, an **X-assignment** is a morphism of indexed sets $\mathbf{N} \to X$. We sometimes merely use "assignment" when X is clear from context.

Notation 7.3.

- We observe that $\mathbb{T}$-sets form a cartesian closed category, where the exponential object X^Y is given by $(X^Y)(\tau) = X(\tau)^{Y(\tau)}$.
- We distinguish it from the hom-set by writing the latter $[Y, X]$.
- For any set A and $\mathbb{T}$-set X, let $A \cdot X = \sum_{a \in A} X$ denote the A-fold coproduct of X.

The analogue of the tensor product $X \otimes Y = X \times Y^{\mathbb{N}}$ will be played by $[\mathbf{N}, Y] \cdot X$, i.e., the iterated self-coproduct of X, with one copy per Y-assignment (see Definition 7.14 below).

Example 7.4. Consider arbitrary $\mathbb{T}$-sets X, Y, and Z.

- The $\mathbb{T}$-set $[X, Y] \cdot Z$ is such that for all types τ, we have

$$([X, Y] \cdot Z)(\tau) = [X, Y] \cdot Z(\tau) = [X, Y] \times Z(\tau).$$

- The $\mathbb{T}$-set $Y^X \times Z$ is such that for all types τ, we have

$$(Y^X \times Z)(\tau) = Y(\tau)^{X(\tau)} \times Z(\tau).$$

We will use the former for generalising substitution to the typed case.

Notation 7.5. For coherence with the untyped case, we tend to write an element of $([\mathbf{N}, Y] \cdot X)(\tau)$ as (x, f), with $x \in X(\tau)$ and $f \colon \mathbf{N} \to Y$.

Furthermore, Notation 2.2 straightforwardly adapts to the typed case as follows.

Notation 7.6. Consider any map $s \colon [\mathbf{N}, Y] \cdot X \to Z$.

- For all $\tau \in \mathbb{T}$, $x \in X(\tau)$, and $g \colon \mathbf{N} \to Y$, we write $x[g]_{s,\tau}$ for $s_\tau(x, g)$, or even $x[g]$ when s and τ are clear from context.
- Furthermore, s gives rise to the map

$$\begin{array}{rcl} [\mathbf{N}, Y] \cdot X^{\mathbf{N}} & \to & Z^{\mathbf{N}} \\ \tau \mapsto (g, f : \mathbb{N} \to X(\tau)) & \mapsto & n \mapsto f(n)[g]_{s,\tau}. \end{array}$$

  We use notation similar to Notation 2.2 for this map, i.e., $f[g]_{s,\tau}(n) := f(n)[g]_{s,\tau}$, or $f[g](n) = f(n)[g]$ when s and τ are clear from context.
- We use the same notation for the map

$$\begin{array}{rcl} [\mathbf{N}, Y] \times [\mathbf{N}, X] & \to & [\mathbf{N}, Z] \\ (g, f) & \mapsto & \tau, n \mapsto f(n)[g]_{s,\tau}. \end{array}$$

The definition of De Bruijn monads generalises almost *mutatis mutandis*:

Definition 7.7. A **De Bruijn $\mathbb{T}$-monad** is a $\mathbb{T}$-set X, equipped with

- a **substitution** morphism $s \colon [\mathbf{N}, X] \cdot X \to X$, which takes an element $x \in X$ and an assignment $f \colon \mathbf{N} \to X$, and returns an element $x[f]$, and
- a **variables** morphism $v \colon \mathbf{N} \to X$,

such that for all $x \in X$, and $f, g \colon \mathbf{N} \to X$, we have

$$x[f][g] = x[f[g]] \qquad v(n)[f] = f(n) \qquad x[v] = x.$$

Example 7.8. The $\mathbb{T}$-set $\mathbf{N}$ itself is clearly a De Bruijn $\mathbb{T}$-monad, with variables given by the identity and substitution $[\mathbf{N}, \mathbf{N}] \cdot \mathbf{N} \to \mathbf{N}$ given by evaluation. It is in fact initial in $\mathbf{DBMnd}(\mathbb{T})$.

Example 7.9. The set Λ_{ST} of simply-typed λ-terms with free variables of type τ in $\mathbb{N} \times \{\tau\}$, considered equivalent modulo α-renaming, forms a De Bruijn monad. Variables $\mathbf{N} \to \Lambda_{ST}$ are given by mapping, at any τ, any $n \in \mathbb{N}$ to the variable (n, τ). Substitution $[\mathbf{N}, \Lambda_{ST}] \cdot \Lambda_{ST} \to \Lambda_{ST}$ is standard, capture-avoiding substitution. One main purpose of this section is to characterise Λ_{ST} by a universal property, and reconstruct it categorically.

Remark 7.10. Untyped languages with multiple syntactic categories form De Bruijn monads. Indeed, it suffices to take $\mathbb{T}$ to be the set of syntactic categories, and, for each $c \in \mathbb{T}$, let $X(c)$ be the set of terms of syntactic category c. This should be taken with a grain of salt, though, as this assumes that each syntactic category has its kind of variables. E.g., let us consider a λ-calculus in which we wish to distinguish values from terms. Syntax then goes as follows:

$$
\begin{aligned}
e &::= v \mid e\ e && \text{(terms)} \\
v &::= x \mid \lambda x.e && \text{(values).}
\end{aligned}
$$

Attempting to organise this as a (simply-typed) De Bruijn monad X, we take $\mathbb{T} = 2 = \{\mathbf{t}, \mathbf{v}\}$, and let $X(\mathbf{t})$ be the set of terms, while $X(\mathbf{v})$ is the set of values. However, X fails to be a De Bruijn monad because there are no term variables. One way of understanding this is that values form an untyped De Bruijn monad, and terms form a module over it [HHL20, HHL22]. We present a simply-typed version of this approach in detail below in §7.5. Another way out consists in adding term variables α to the syntax, which thus becomes:

$$
\begin{aligned}
e &::= \alpha \mid v \mid e\ e && \text{(terms)} \\
v &::= x \mid \lambda x.e && \text{(values).}
\end{aligned}
$$

7.2. Morphisms of De Bruijn $\mathbb{T}$-monads.

Definition 7.11. A morphism $(X, s, v) \to (Y, t, w)$ between De Bruijn $\mathbb{T}$-monads is a morphism $f : X \to Y$ of $\mathbb{T}$-sets commuting with substitution and variables, in the sense that for all $\tau \in \mathbb{T}$, $x \in X(\tau)$, and $g : \mathbf{N} \to X$ we have $f_\tau(x[g]) = f_\tau(x)[f \circ g]$ and $f \circ v = w$.

Remark 7.12. More explicitly, the first axiom says: $f_\tau(s_\tau(x, g)) = t_\tau(f_\tau(x), f \circ g)$.

Proposition 7.13. *De Bruijn $\mathbb{T}$-monads and morphisms between them form a category* $\mathbf{DBMnd}(\mathbb{T})$.

7.3. De Bruijn $\mathbb{T}$-monads as relative monads.

The presentation based on relative monads extends to the typed setting, by replacing the functor $\mathbb{N} : 1 \to \mathbf{Set}$ with $\mathbf{N} : 1 \to \mathbf{Set}^{\mathbb{T}}$, so that $\mathrm{Lan}_{\mathbf{N}}(X)(Y) \cong [\mathbf{N}, Y] \cdot X$. Thus, $\mathbf{DBMnd}(\mathbb{T})$ is equivalently the category of monads relative to the functor $1 \to \mathbf{Set}^{\mathbb{T}}$ picking $\mathbf{N}$. For the record, let us explicitly introduce the corresponding tensor product.

Definition 7.14. For any $\mathbb{T}$-sets X and Y, let $X \otimes Y = [\mathbf{N}, Y] \cdot X$.

Notation 7.15. For coherence with the untyped case, we tend to write an element of $(X \otimes Y)(\tau)$ as (x, f), with $x \in X(\tau)$ and $f : \mathbf{N} \to Y$.

7.4. Initial-algebra semantics.

We now adapt the initial-algebra semantics of §3 to the typed case.

7.4.1. *Assignment lifting.* Let us start by generalising lifting to the typed case. This relies on a typed form of lifting, which acts on all variables of a given type, leaving all other variables untouched.

Definition 7.16. Let (X, s, v) denote any De Bruijn $\mathbb{T}$-monad. We first define a typed analogue $\uparrow^\tau$ of the $\uparrow$ of Definition 2.6, as below left, and then the **lifting** of any assignment $\sigma \colon \mathbf{N} \to X$ as below right.

$$\begin{array}{rcll} (\uparrow^\tau)_\tau(n) & = & v_\tau(n+1) & \\ (\uparrow^\tau)_{\tau'}(n) & = & v_{\tau'}(n) & (\text{if } \tau \neq \tau') \end{array} \qquad \begin{array}{rcll} (\Uparrow^\tau \sigma)_\tau(0) & = & v_\tau(0) & \\ (\Uparrow^\tau \sigma)_\tau(n+1) & = & \sigma_\tau(n)[\uparrow^\tau] & \\ (\Uparrow^\tau \sigma)_{\tau'}(n) & = & \sigma_{\tau'}(n)[\uparrow^\tau] & (\text{if } \tau \neq \tau'). \end{array}$$

Finally, for any sequence $\gamma = (\tau_1, \ldots, \tau_n)$ of types, we define $\Uparrow^\gamma \sigma$ inductively, by $\Uparrow^\varepsilon \sigma = \sigma$ and $\Uparrow^{\gamma,\tau} \sigma = \Uparrow^\tau (\Uparrow^\gamma \sigma)$, where ε denotes the empty sequence.

7.4.2. *Binding arities and binding conditions.* We may now generalise binding arities and the binding conditions to the typed setting.

Definition 7.17 [FH10]**.**

- A **first-order arity** is a pair $((\tau_1, \ldots, \tau_p), \tau) \in \mathbb{T}^* \times \mathbb{T}$ of a list of types and a type.
- A **binding arity** is a tuple $a = (((\gamma_1, \tau_1), \ldots, (\gamma_p, \tau_p)), \tau)$, where each $\gamma_i \in \mathbb{T}^*$ is a list of types, and each τ_i, as well as τ, are types. In other words, $a \in (\mathbb{T}^* \times \mathbb{T})^* \times \mathbb{T}$.
- The **first-order arity $|a|$ associated** to a is $((\tau_1, \ldots, \tau_p), \tau) \in \mathbb{T}^* \times \mathbb{T}$.

Remark 7.18. An arity $(((\gamma_1, \tau_1), \ldots, (\gamma_p, \tau_p)), \tau)$ may be understood as follows:

- τ is the return type;
- $(\tau_1, \ldots, \tau_p)$ are the argument types;
- each list $\gamma_i = (\tau^i_1, \ldots, \tau^i_{q_i})$ specifies that the ith argument should be considered as binding q_i variables, of respective types $\tau^i_1,\ldots,\tau^i_{q_i}$.

Notation 7.19. We write any arity $(((\gamma_1, \tau_1), \ldots, (\gamma_p, \tau_p)), \tau)$ as an inference rule

$$\frac{\gamma_1 \vdash \tau_1 \qquad \ldots \qquad \gamma_p \vdash \tau_p}{\vdash \tau}\ .$$

Example 7.20. The binding signature for simply-typed λ-calculus has two operations $\mathrm{lam}_{\tau,\tau'}$ and $\mathrm{app}_{\tau,\tau'}$ for each pair (τ, τ') of types, of respective arities

$$\frac{\tau \vdash \tau'}{\vdash \tau \to \tau'} \qquad \text{and} \qquad \frac{\vdash \tau \to \tau' \qquad \vdash \tau}{\vdash \tau'}\ .$$

This allows us to generalise the binding conditions, as follows.

Definition 7.21. Let $a = (((\gamma_1, \tau_1), \ldots, (\gamma_p, \tau_p)), \tau)$ be any binding arity, and M be any $\mathbb{T}$-set equipped with morphisms $s \colon [\mathbf{N}, M] \cdot M \to M$ and $v \colon \mathbf{N} \to M$. An **operation of binding arity** a is a map $o \colon M(\tau_1) \times \ldots \times M(\tau_p) \to M(\tau)$ satisfying the following **a-binding condition** w.r.t. (s, v):

$$\begin{array}{l} \forall \sigma \colon \mathbf{N} \to M, x_1, \ldots, x_p \in M(\tau_1) \times \ldots \times M(\tau_p), \\ o(x_1, \ldots, x_p)[\sigma] = o(x_1[\Uparrow^{\gamma_1} \sigma], \ldots, x_p[\Uparrow^{\gamma_p} \sigma]). \end{array} \tag{7.1}$$

7.4.3. *Binding signatures and algebras.* Finally, we generalise signatures and their models to the typed setting, and state a typed initiality theorem.

Definition 7.22. A **first-order typed signature** consists of a set O of **operations**, equipped with an **arity** map $ar\colon O \to \mathbb{T}^* \times \mathbb{T}$.

Definition 7.23. Consider a first-order signature $S := (O, ar)$.

- An **S-algebra** is a set X, together with, for each operation $o \in O$ with arity $((\tau_1, \ldots, \tau_p), \tau)$, a map $o_X(\tau_1) \times \ldots \times X(\tau_p) \to X(\tau)$.
- A morphism $X \to Y$ of S-algebras is a map between underlying sets commuting with operations, in the sense that for each $o \in O$, letting $((\tau_1, \ldots, \tau_p), \tau) := ar(o)$, we have for all $x_1, \ldots, x_p \in X(\tau_1) \times \ldots \times X(\tau_p)$, $f_\tau(o_X(x_1, \ldots, x_p)) = o_Y(f_{\tau_1}(x_1), \ldots, f_{\tau_p}(x_p))$.

We denote by S-**alg** the category of S-algebras and morphisms between them.

Definition 7.24.

- A **$\mathbb{T}$-binding signature** consists of a set O of **operations**, equipped with an arity map $O \to (\mathbb{T}^* \times \mathbb{T})^* \times \mathbb{T}$.
- The first-order signature $|S|$ associated with a binding signature $S := (O, ar)$ is $|S| := (O, |ar|)$, where $|ar|\colon O \to \mathbb{T}^* \times \mathbb{T}$ maps any $o \in O$ to $|ar(o)|$.

Let us now present the notion of De Bruijn S-algebra:

Definition 7.25. Consider any $\mathbb{T}$-binding signature $S := (O, ar)$.

- A **De Bruijn S-algebra** consists of a De Bruijn $\mathbb{T}$-monad (X, s, v), together with, for all $o \in O$, an operation of binding arity $ar(o)$.
- A morphism of De Bruijn S-algebras is a map $f\colon X \to Y$ between underlying sets, which is a morphism both of De Bruijn monads and of $|S|$-algebras.

We denote by S-**DBAlg** the category of De Bruijn S-algebras and morphisms between them.

In order to extend the initiality theorem to the typed case, we need to define the endofunctor induced by a $\mathbb{T}$-binding signature S, which only depends on $|S|$, as in the untyped case.

Definition 7.26 (Induced endofunctor)**.**

- For any $\tau \in \mathbb{T}$, let $\mathbf{y}_\tau$ denote the $\mathbb{T}$-set defined by

$$\begin{array}{rcl} \mathbf{y}_\tau(\tau) & = & 1 \\ \mathbf{y}_\tau(\tau') & = & \emptyset \quad (\text{if } \tau' \neq \tau). \end{array}$$

- The endofunctor Σ_a induced by any arity

$$a = \frac{\tau^1_1, \ldots, \tau^1_{q_1} \vdash \tau_1 \qquad \ldots \qquad \tau^p_1, \ldots, \tau^p_{q_p} \vdash \tau_p}{\vdash \tau}$$

  is defined by $\Sigma_a(X) = (X(\tau_1) \times \ldots \times X(\tau_p)) \cdot \mathbf{y}_\tau$. Thus, a Σ_a-algebra is a $\mathbb{T}$-set X equipped with a morphism $(X(\tau_1) \times \ldots \times X(\tau_p)) \cdot \mathbf{y}_\tau \to X$, or equivalently a map $X(\tau_1) \times \ldots \times X(\tau_p) \to X(\tau)$.
- The endofunctor Σ_S induced by any $\mathbb{T}$-binding signature $S = (O, ar)$ is defined by

$$\Sigma_S(X) = \sum_{o \in O} \Sigma_{ar(o)}(X).$$

We have the following typed extension of the initiality theorem.

Theorem 7.27. *For any* $\mathbb{T}$*-binding signature* S*, let* DB_S *denote the initial* $(\mathbf{N}+\Sigma_S)$*-algebra, with structure morphisms* $v\colon \mathbf{N} \to \mathrm{DB}_S$ *and* $a\colon \Sigma_S(\mathrm{DB}_S) \to \mathrm{DB}_S$*, inducing maps*

$$o_{\mathrm{DB}_S}\colon \mathrm{DB}_S(\tau_1) \times \ldots \times \mathrm{DB}_S(\tau_p) \to \mathrm{DB}_S(\tau)$$

for all $o \in O$ *with* $ar(o) = (((\gamma_1, \tau_1), \ldots, (\gamma_p, \tau_p)), \tau)$*. Then:*

(i) *There exists a unique morphism* $s\colon [\mathbf{N}, \mathrm{DB}_S] \cdot \mathrm{DB}_S \to \mathrm{DB}_S$ *such that*
- *for all* $\tau \in \mathbb{T}$*,* $n \in \mathbb{N}$*, and* $f\colon \mathbf{N} \to \mathrm{DB}_S$*,* $s_\tau(v_\tau(n), f) = f_\tau(n)$*, and*
- *for all* $o \in O$*, the map* o_{DB_S} *satisfies the* $ar(o)$*-binding condition w.r.t.* (s, v)*.*

(ii) *This morphism* s *turns* (DB_S, v, s, a) *into a De Bruijn* S*-algebra.*

(iii) *This De Bruijn* S*-algebra is initial in* S*-***DBAlg**.

7.5. **Application: values in simply-typed λ-calculus.** We saw in Example 7.20 that the De Bruijn monad of simply-typed λ-calculus terms admits a simple signature. But we also mentioned in Remark 7.10 that (untyped) values may be organised as a monad, with terms forming a module over it. In this subsection, as announced, we present a signature for a simply-typed version of this.

We want elements of our De Bruijn monad at any type to be **values** of that type. (Indeed, values are closed under value substitution.)

However, in order to define a signature for this De Bruijn monad, we cannot use application. Indeed, application returns terms which are not values.

In order to solve this problem, we need to introduce the following auxiliary construction. The idea is to pack up all occurrences of application between layers of value operations (abstraction and variable), into a single operation.

We do this by introducing **application binary trees**, which are proof derivations generated by the following rules,

$$\frac{}{\sigma \vdash_{BT} \sigma} \qquad\qquad \frac{\Gamma \vdash_{BT} \sigma \to \tau \qquad \Delta \vdash_{BT} \sigma}{\Gamma, \Delta \vdash_{BT} \tau}$$

where Γ, Δ denotes concatenation of lists of simple types. Thus a proof of $\Gamma \vdash_{BT} \tau$ is essentially a simply-typed term involving only application, with one, linearly used free variable for each type in Γ, in the same order. Linearity is here used to keep track of all leaves in the typing context, which we will now use to define the desired binding signature.

Definition 7.28. Let BT^{Γ}_{σ} denote the set of such application binary trees with conclusion $\Gamma \vdash_{BT} \sigma$.

We then take as binding signature S_{λ_v} for simply-typed values the one with one operation $L_{\pi,\sigma}$ of arity

$$\frac{\sigma \vdash \tau_1 \qquad \ldots \qquad \sigma \vdash \tau_n}{\vdash \sigma \to \tau}$$

for each simple type σ and application binary tree $\pi \in BT^{\tau_1,\ldots,\tau_n}_{\tau}$.

Example 7.29. For a simple example, if π is merely the axiom $\tau \mapsto \tau$, then $L_{\pi,\sigma}$ has the arity $\frac{\sigma \vdash \tau}{\vdash \sigma \to \tau}$ of λ-abstraction. In this case, $L_{\pi,\sigma}$ is thought of as forming $\lambda x : \sigma . v^{\tau}$ from any value v of type τ with an additional variable of type σ.

Example 7.30. For a less trivial, yet basic example, if π is

$$\frac{\tau_1 \to \tau_2 \vdash_{BT} \tau_1 \to \tau_2 \qquad \tau_1 \vdash_{BT} \tau_1}{\tau_1 \to \tau_2, \tau_1 \vdash_{BT} \tau_2},$$

then $L_{\pi,\sigma}$ has arity

$$\frac{\sigma \vdash \tau_1 \to \tau_2 \qquad \sigma \vdash \tau_1}{\vdash \sigma \to \tau_2}.$$

This operation is thought of as forming $\lambda x : \sigma.(f^{\tau_1 \to \tau_2}\ a^{\tau_1})$ from values f and a.

By Theorem 7.27, the initial De Bruijn S_{λ_v}-algebra has as carrier the initial algebra for the induced endofunctor, which is by construction the subset of values.

8. Equations

In this section, we introduce a notion of equational theory for specifying (typed) De Bruijn monads, following ideas from [FH09].

Definition 8.1. A **De Bruijn equational theory** consists of

- two binding signatures S and T, and
- two functors $L, R \colon S\text{-}\mathbf{DBAlg} \to T\text{-}\mathbf{DBAlg}$ over $\mathbf{DBMnd}(\mathbb{T})$, i.e., making the following diagram commute serially, where U^S and U^T denote the forgetful functors.

$$\begin{array}{ccc} S\text{-}\mathbf{DBAlg} & \overset{L}{\underset{R}{\rightrightarrows}} & T\text{-}\mathbf{DBAlg} \\ & {}_{U^S}\searrow \quad \swarrow_{U^T} & \\ & \mathbf{DBMnd}(\mathbb{T}) & \end{array}$$

Example 8.2. Recalling the binding signature S_Λ for λ-calculus from Example 2.15, let us define a De Bruijn equational theory for β-equivalence. We take $T_\beta = (1, 0)$, and for any De Bruijn S_Λ-algebra X,

- $L(X)$ has as structure map

$$\begin{array}{rcl} X^2 & \to & X \\ (e_1, e_2) & \mapsto & \mathrm{app}(\mathrm{lam}(e_1), e_2) \end{array}$$

  while
- $R(X)$ has as structure map

$$\begin{array}{rcl} X^2 & \to & X \\ (e_1, e_2) & \mapsto & e_1[e_2 \cdot \mathrm{id}]. \end{array}$$

  (Here $e_2 \cdot \mathrm{id}$ denotes the assignment $0 \mapsto e_2$, $n+1 \mapsto v(n)$.)

Definition 8.3. Given an equational theory $E = (S, T, L, R)$, a De Bruijn E**-algebra** is a De Bruijn S-algebra X such that $L(X) = R(X)$.

Let $E\text{-}\mathbf{DBAlg}$ denote the category of E-algebras, with morphisms of De Bruijn S-algebras between them.

Remark 8.4. The category $E\text{-}\mathbf{DBAlg}$ is an equaliser of L and R in $\mathbf{CAT}$.

Let us now turn to characterising the initial De Bruijn E-algebra, for any De Bruijn equational theory E. For this, we introduce the following relation.

Definition 8.5. For any De Bruijn equational theory $E = (S, T, L, R)$, with $S = (O, ar)$ and $T = (O', ar')$, let DB_S denote the initial $(\mathbf{N} + \Sigma_S)$-algebra. We define $\sim_E$ to be the smallest equivalence relation on DB_S satisfying the following rules,

$$\frac{}{o'_{L(\mathrm{DB}_S)}(e_1, \ldots, e_p) \sim_E o'_{R(\mathrm{DB}_S)}(e_1, \ldots, e_p)} \qquad \frac{e_1 \sim_E e'_1 \quad \ldots \quad e_q \sim_E e'_q}{o_{\mathrm{DB}_S}(e_1, \ldots, e_q) \sim_E o_{\mathrm{DB}_S}(e'_1, \ldots, e'_q)}$$

for all $e, e_1, \ldots$ in DB_S, $o' \in O'$ with $|ar'(o')| = p$, and $o \in O$ with $|ar(o)| = q$.

Example 8.6. For the equational theory of Example 8.2, the first rule instantiates precisely to the β-rule, while the second enforces congruence.

Theorem 8.7. *For any equational theory $E = (S, T, L, R)$, E-* **DBAlg** *admits an initial object, whose carrier set is the quotient* $\mathrm{DB}_S/\sim_E$.

Proof. We formalised the proof in Coq [Laf22a], where

- existence is called `quotsyntax.ini_morE_model_mor`,
- uniqueness is called `quotsyntax.ini_morE_unique`.

The specific case of λ-calculus modulo $\beta\eta$-equation has also been mechanised in HOL [Mag22]. □

Example 8.8. The initial model for the equational theory of Example 8.2 is the quotient of λ-terms in De Bruijn representation by β-equivalence.

9. Mechanised proofs

Our theoretical framework is in particular meant to help specifying and reasoning mechanically about binding syntax using De Bruijn representation. To give practical examples, we describe in this section two implementations, in HOL Light and Coq, that cover several crucial parts of the theory of this paper, and, in particular Theorems 2.21 and 3.16. The full source code is available on github [Laf22a, Mag22].

9.1. **HOL Light proof.** Let us start by discussing the HOL Light formalisation. One key fact is that, despite being much weaker than ZFC set theory or the Calculus of Inductive Constructions, HOL is expressive enough to program the De Bruijn encoding and to reason about it. For instance, other representations, such as those based on monads over sets [HM10] (possibly via the nested datatype technique [BP99, HM12]) are not directly implementable in HOL.

The reader does not need to be familiar with higher-order logic (HOL) to follow the essential ideas of this section. HOL is based on simply-typed λ-calculus. Following a standard denotational semantics in Zermelo-Fraenkel set theory, types in HOL can be thought of as non-empty sets.

Our HOL Light implementation is divided into two main parts. The first part treats the specific case of λ-calculus and can be useful to illustrate the essential ideas of this paper in a simple—yet paradigmatic—setting.

Notation 9.1. In the following, we refer to the theorems and definitions in the HOL code by indicating their name in parentheses in `teletype` font, e.g., (`INITIAL_MORPHISM_UNIQUE`). HOL terms and formulas are enclosed in backquotes as in `` `2 + 2` ``, types are prefixed by a colon as in `` `:bool` ``.

The type `:dblambda` of λ-calculus is simply defined as the following inductive type

```
let dblambda_INDUCT, dblambda_RECURSION = define_type
  "dblambda = REF num | APP dblambda dblambda | ABS dblambda";;
```

The lifting function (`‘DERIV‘`) and substitution function `‘SUBST‘` are defined in the obvious way. The equations that characterise them are summarised in the following theorem (`SUBST_CLAUSES`).

```
|- (!f i. SUBST f (REF i) = f i) /\
   (!f x y. SUBST f (APP x y) = APP (SUBST f x) (SUBST f y)) /\
   (!f x. SUBST f (ABS x) = ABS (SUBST (DERIV f) x)) /\
   (!f. DERIV f 0 = REF 0) /\
   (!f i. DERIV f (SUC i) = SUBST (REF o SUC) (f i))
```

The names `‘SUC‘` and `‘o‘` respectively denote successor and function composition. The symbols `‘/\‘` and `‘!‘` are HOL notation for conjunction and universal quantification. We recognise:

- in the first line, the variables map,
- in the next two lines, the binding conditions for application and abstraction, and
- in the final two lines, the equations defining the lifting of an assignment.

We show that the functions `‘SUBST‘` and `‘DERIV‘` satisfying the above identities are unique (`SUBST_DERIV_UNIQUE`); thus, we have formalised the first point of Theorem 2.21 for the case of λ-calculus.

The second point of the Theorem translates in a law for associativity (`SUBST_SUBST`) and a second law for unitality (`SUBST_REF`) complementing the first equation of (`SUBST_CLAUSES`) above.

We also provide the classical definition of unary substitution (`SUBST1`, as found e.g., in [Hue94]) and then show how the latter is an instance of the former (`SUBST1_EQ_SUBST`). As shown by other authors [ACCL90, SST15], reasoning on parallel substitution can be significantly easier. Here for instance, we prove the associativity of unary substitution in a few lines (`SUBST1_SUBST1`) by reducing to parallel substitution. In contrast, proving the same result directly for unary substitution is less intuitive: the mere statement of the property to be proved by induction is tricky to devise.

Next, we introduce the category of De Bruijn monads (`MONAD`, `MONAD_MOR`) and their associated modules (`MODULE`, `MODULE_MOR`). Our Definition 2.3 presents De Bruijn monads as triples consisting of a set, an associative substitution operator and a two-sided unit. In HOL, this is implemented as a type `‘:A‘` together with a substitution operation `‘op:(num->A)->A->A‘` and a unit `‘e:num->A‘`.[2] However, the unit is uniquely determined by the substitution operator and it is denoted `‘UNIT op‘` in our implementation. Moreover, the type `‘:A‘` is automatically inferred. Thus, we simply identify a De Bruijn monad by its substitution operator, that is, we write `‘op IN MONAD‘` to indicate that `‘op‘` is (the substitution operator of) a monad.

```
|- !op. op IN MONAD <=>
        (!f g x:A. op g (op f x) = op (op g o f) x) /\
        (!f n. op f (UNIT op n) = f n) /\
        (!x. op (UNIT op) x = x)
```

[2] We warn the reader that in this code op is used for substitution operation and should not be confused with the operations of the syntax $o \in O$ of the previous sections.

The set of morphisms between two De Bruijn monads `op1` and `op2` is then defined as follows.

```
|- MONAD_MOR (op1,op2) =
   {h:A->B | op1 IN MONAD /\ op2 IN MONAD /\
             (!n. h (UNIT op1 n) = UNIT op2 n) /\
             (!f x. h (op1 f x) = op2 (h o f) (h x))}
```

Modules are implemented using a similar style. We implemented the constructions on modules needed for interpreting binding signatures: product (MPROD) and derivation (DMOP).

In this setup, we can state and prove Theorem 3.16 for the λ-calculus. The models of our syntax are De Bruijn monads endowed with functions `app` and `abs` that are module morphisms (DBLAMBDA_MODEL).

```
app IN MODULE_MOR op (MPROD op op, op)
lam IN MODULE_MOR op (DMOP op op, op)
```

Remark 9.2. Both product MPROD and derivation DMOP expect two arguments, for different reasons. Product expects the two modules it takes the product of. Derivation takes a monad and a module over it, and it derives the latter. The monad is needed because derivation relies on monadic substitution (in the definition of $\Uparrow$).

Model morphisms (DBLAMBDA_MODEL_MOR) are De Bruijn monad morphisms that commute with `app` and `abs`. We then obtain the universal property of λ-calculus by giving an initial model morphism (DBLAMBDAINIT, DBLAMBDAINIT_IN_DBLAMBDA_MODEL_MOR),

```
|- !op app lam.
      (op,app,lam) IN DBLAMBDA_MODEL
      ==> DBLAMBDAINIT (op,app,lam) IN
          DBLAMBDA_MODEL_MOR ((SUBST,UNCURRY APP,ABS),(op,app,lam))
```

and by proving its uniqueness (DBLAMBDAINIT_UNIQUE).

This part closes with the analogous theorem for the initial-algebra semantics of λ-calculus modulo $\beta\eta$-equivalence (EXP_MONAD_MOR_LC_EXPMAP, LC_EXPMAP_UNIQUE). The style is the one proposed in [HM10] which uses exponential monads, that is, a monads M endowed with a module isomorphism $\mathsf{abs} : M' \xrightarrow{\simeq} M$.

The second part of the HOL Light code implements Theorems 2.21 and 3.16 for arbitrary signatures.

The increased generality comes at a cost in this implementation: since HOL does not feature dependent types, it is impossible to implement the term algebra of a given binding signature as a mere type: one has to resort to a "well-formedness" predicate. From this perspective, it may be instructive to compare with our Coq implementation, which takes advantage of dependent types.

The above difficulty is solved in the standard way in HOL Light. First, we build a type `rterm` for **raw terms** over a "full" signature, i.e., one with countably many operations of each arity. We then introduce an inductive set (i.e., an inductive predicate) of well-formed terms (WELLFORMED_RULES) that selects the terms respecting a given signature. Besides this technical difficulty, the formal development follows the same pattern as for λ-calculus.

The substitution operator is specified by two equations (TMSUBST_CALUSES).

```
|- (!f i. TMSUBST f (TMREF i) = f i) /\
   (!f c args. TMSUBST f (FN c args) =
               FN c (MAP (\(k,x). k,TMSUBST (TMDERIV k f) x) args))
```

where 'TMREF' denotes variables and 'FN' denotes operations from the signature. The latter takes two arguments, the **name** of the construction 'c' (a natural number) and a list of pairs '(k,x)' where 'k' is a natural number denoting the number of bound variables and 'x' is a term.

We formulate the appropriate notion of category of models in this setting (`MODEL`, `MODEL_MOR`) of which the above data constitutes an object (`RTERM_IN_MODEL`). Then we prove the universal property by giving the initial morphism (`INITIAL_MORPHISM_IN_MODEL_MOR`) and show its uniqueness (`INITIAL_MORPHISM_UNIQUE`).

Finally, to tie the knot, we derive again the universal property for λ-calculus as an instance of this new, more general, framework (`DBLAMBDA_UNIVERSAL`).

9.2. **Coq proof.** We now discuss the Coq formalisation [Laf22a]. Our implementation addresses only the general case of arbitrary signatures since, as mentioned before, we do not gain anything by treating a particular case separately, thanks to dependent types. As also mentioned above, the formalisation covers signatures with equations, in the untyped case.

The reader who wants to skim through the main definitions and constructions of this implementation can look at the file `Summary.v`, which reviews the main constructions and results. The formalisation has an idiomatic style: the minute details of implementation pose no significant problem. Therefore, we just point the reader to the most relevant definitions.

We start with the file `syntaxdb.v`. The syntax is defined as an inductive type parameterised by a signature.

```
Record signature :=
  { O : Type;
    ar : O → list ℕ}.

Inductive Z (S : signature) : Type :=
  Var : ℕ → Z S
| Op : ∀ (o : O S), vec (Z S) (ar o) → Z S.
```

Here, **vec** A ℓ is defined as the inductive type of vectors of elements A, whose length is that of the list ℓ. Assuming a type X equipped with a substitution map $-[-]$ and variable embedding, the binding condition is defined as

```
Definition binding_condition (a : list ℕ) (op : vec X a → X) :=
    ∀ (f : ℕ → X)(v : vec X a),
      op v [ f ] = op (vec_map (fun n x ⇒ x [ f ^ ( n )]) v).
```

where

- $-[-]$ denotes substitution,
- f ^ (n) denotes $\Uparrow^n f$, and
- vec_map f maps a vector $v = (x_1, \ldots, x_n)$ of type **vec** A $(a_1, \ldots, a_n)$ to $(f\ a_1\ x_1, \ldots, f\ a_n\ x_n)$.

The definition of models is split into two parts: the data, and the properties.

```
Record model_data (S : signature) :=
  { carrier :> Type;
    variables : ℕ → carrier;
    ops : ∀ (o : O S), vec carrier (ar o) → carrier;
    substitution : (ℕ → carrier) → (carrier → carrier)
  }.
Record is_model {S : signature}(m : model_data S) := {
```

```
 substitution_ext : ∀ (f g : ℕ → m), (∀ n, f n = g n) → ∀ x, x [ f ] = x [ g ];
 variables_subst : ∀ x f, (variables m x) [ f ] = f x;
 ops_subst : ∀ (o : O S), binding_condition (variables m) (substitution (m := m)) (ops o);
 assoc : ∀ (f g : ℕ → m) (x : m), x [ g ] [ f ] = x [ (fun n ⇒ (g n) [ f ]) ] ;
 id_neutral : ∀ (x : m), x [ variables m ] = x
}.
Record model (S : signature) := {
 mod_carrier :> model_data S;
 mod_laws : is_model mod_carrier
}.
```

The symbol :> declares an implicit coercion. For example, given a term m of type **model_data**, we can then just write m when we actually mean carrier m. Coq implicitly inserts the field getter `carrier` whenever it is necessary, based on the typing constraints.

The first property substitution_ext looks superfluous: it intuitively follows from the fact that pointwise equal functions are equal. We however explicitly require this property because the latter general fact, called function extensionality, is not built-in in Coq.[3]

The remaining of the file `syntaxdb.v` consists of the definition of model morphisms and the proofs of Theorems 2.21 and 3.16. The rest of the formalisation focuses on initiality for signatures with equations (Theorem 8.7), in the untyped setting. Since quotients are not built-in in Coq, we axiomatise a quotient type $X//R$ in the file `Quot.v` for each equivalence relation R on a type X, that is, for each R of type **Eqv** X. The canonical projection is then denoted by $-/R\colon X \to X//R$.

A De Bruijn equational theory (Definition 8.1) is defined as a record with four fields.

```
Record equational_theory :=
  { metavariables : signature ;
    main_signature : signature ;
    left_handside : half_equation main_signature metavariables ;
    right_handside : half_equation main_signature metavariables
  }.
```

A **half-equation** is a functor between two categories of models preserving the underlying De Bruijn monad. In other words, it provides any model of the first signature with an algebra structure for the second signature, and this assignment is compatible with model morphisms.

```
Record half_equation (S1 : signature)(S2 : signature) :=
  {
    lift_ops :> ∀ (m : model S1), ∀ (o : O S2), vec m (ar o) → m;
    lift_ops_subst :
      ∀ (m : model S1) (o : O S2),
        binding_condition (variables m) (substitution (m := m))
                              (@lift_ops m o) ;
    lift_ops_natural : ∀ (m1 m2 : model S1) (f : model_mor m1 m2)
                              (o : O S2)(v : vec m1 (ar o)),
        lift_ops (vec_map (fun _ ⇒ f) v) = f (lift_ops v)
  }.
```

[3]Alternatively, we could have explicitly assumed function extensionality as an (unrestricted) axiom, as we do anyway when axiomatising quotient types.

A model of an equational theory is a model of the main signature equalising both half-equations, in the sense that they yield equal algebra structures.

```
Record model_equational (E : equational_theory) :=
  { main_model :> model (main_signature E) ;
    model_eq : ∀ o (v : vec main_model (ar o)),
        left_handside E main_model o v = right_handside E main_model o v
  }.
```

Following Definition 8.5, the initial algebra of an equational theory is obtained by quotienting the initial algebra of the main signature by the smallest congruent equivalence relation relating the images by the algebra structures induced by the two half-equations.

```
Inductive rel_Z (E : equational_theory) : Z (main_signature E) → Z (main_signature E)
→ Prop :=
| eqE : ∀ o v, rel_Z (left_handside E (ZModel _) o v) (right_handside E (ZModel _) o v)
| reflE : ∀ z, rel_Z z z
| symE : ∀ a b, rel_Z b a → rel_Z a b
| transE : ∀ a b c, rel_Z a b → rel_Z b c → rel_Z a c
| congrE : ∀ (o : O (main_signature E)) (v v' : vec _ (ar o)),
    rel_vec (@rel_Z E) v v' → rel_Z (Op o v) (Op o v').

Definition ZEr (E : equational_theory) : Eqv (Z (main_signature E)) :=
  Build_Eqv (@rel_Z E) (@reflE E) (@symE E)(@transE E) .


Definition ZE (E : equational_theory) := Z (main_signature E) // (ZEr E).
```

The congruence case congrE involves the pointwise relation **rel_vec** R induced on vectors by a relation R.

The rest of the file consists in showing that this definition indeed induces an initial model of the given equational theory. We also provide the instantiation on the equational signature `LCβη_sig` of λ-calculus modulo $\beta\eta$-equivalence.

10. Conclusion

We have proposed a simple, set-based theory of syntax with variable binding, which associates a notion of model (or algebra) to each binding signature, and constructs a term model following De Bruijn representation. The notion of model features a substitution operation. We have experienced the simplicity of this theory by implementing it in both Coq and HOL Light.

We have furthermore equipped the construction with an initial-algebra semantics, organising the models of any binding signature into a category, and proving that the term model is initial therein.

We have then studied this initial-algebra semantics in a bit more depth, in two directions.

- We have first established a formal link with the presheaf-based approach [FPT99], proving that well-behaved models (in a suitable sense on each side of the correspondence) agree up to an equivalence of categories.
- We have then recast the whole initial-algebra semantics into two established, abstract frameworks for syntax with variable binding, one based on strengths [FPT99, Fio08], the other on modules [HM07, HM10].

Finally, we have shown that our theory extends easily to a simply-typed setting, and smoothly incorporates equations and transitions.

Funding acknowledgement. This work was supported in part by a European Research Council (ERC) Consolidator Grant for the project "TypeFoundry", funded under the European Union's Horizon 2020 Framework Programme (grant agreement no. 101002277).

References

[AAC⁺18] Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. A type and scope safe universe of syntaxes with binding: their semantics and proofs. *Proceedings of the ACM on Programming Languages*, 2(ICFP):90:1–90:30, 2018. doi:10.1145/3236785.

[ABF⁺05] Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The PoplMark challenge. In *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2005. doi:10.1007/11541868_4.

[ACCL90] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. In Frances E. Allen, editor, *Proc. 17th International Symposium on Principles of Programming Languages*, pages 31–46. ACM, 1990. URL: http://dl.acm.org/citation.cfm?id=96709, doi:10.1145/96709.96712.

[ACU15] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/LMCS-11(1:3)2015.

[AGT10] J. Adámek, H. P. Gumm, and V. Trnková. Presentation of set functors: A coalgebraic perspective. *Journal of Logic and Computation*, 20(5):991–1015, 2010. doi:10.1093/logcom/exn090.

[AM21] Nathanael Arkor and Dylan McDermott. Abstract clones for abstract syntax. In Naoki Kobayashi, editor, *Proc. 6th International Conference on Formal Structures for Computation and Deduction*, volume 195 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.30.

[AMBL12] Jirí Adámek, Stefan Milius, Nathan J. Bowler, and Paul Blain Levy. Coproducts of monads on Set. In *Proc. 27th Symposium on Logic in Computer Science*, pages 45–54. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.16.

[AR94] J. Adámek and J. Rosicky. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994. doi:10.1017/CBO9780511600579.

[BHL20] Peio Borthelle, Tom Hirschowitz, and Ambroise Lafont. A cellular Howe theorem. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *Proc. 35th ACM/IEEE Symposium on Logic in Computer Science*. ACM, 2020. doi:10.1145/3373718.3394738.

[BP99] Richard S. Bird and Ross Paterson. De Bruijn notation as a nested datatype. *Journal of Functional Programming*, 9(1):77–91, 1999. doi:10.1017/S0956796899003366.

[BU07] Stefan Berghofer and Christian Urban. A head-to-head comparison of De Bruijn indices and names. *Electronic Notes in Theoretical Computer Science*, 174(5):53–67, 2007. doi:10.1016/j.entcs.2007.01.018.

[DB72] N. G. De Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[FH09] Marcelo Fiore and Chung-Kil Hur. On the construction of free algebras for equational systems. *Theoretical Computer Science*, 410:1704–1729, 2009. doi:10.1016/j.tcs.2008.12.052.

[FH10] M. P. Fiore and C.-K. Hur. Second-order equational logic. In *Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL 2010)*, 2010.

[Fio08] Marcelo P. Fiore. Second-order and dependently-sorted abstract syntax. In *Proc. 23rd Symposium on Logic in Computer Science*, pages 57–68. IEEE, 2008. doi:10.1109/LICS.2008.38.

[FPT99] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proc. 14th Symposium on Logic in Computer Science*. IEEE, 1999. doi:10.1109/LICS.1999.782615.

[FS22] Marcelo Fiore and Dmitrij Szamozvancev. Formal metatheory of second-order abstract syntax. *Proceedings of the ACM on Programming Languages*, 6(POPL), 2022. doi:10.1145/3498715.

[GP99] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *Proc. 14th Symposium on Logic in Computer Science*. IEEE, 1999.

[GP20] Lorenzo Gheri and Andrei Popescu. A formalized general theory of syntax with bindings: Extended version. *Journal of Automated Reasoning*, 64(4):641–675, 2020. doi:10.1007/s10817-019-09522-2.

[GT74] Joseph A. Goguen and James W. Thatcher. Initial algebra semantics. In *15th Annual Symposium on Switching and Automata Theory (SWAT)*, pages 63–77. IEEE, 1974.

[HHL20] André Hirschowitz, Tom Hirschowitz, and Ambroise Lafont. Modules over monads and operational semantics. In Zena M. Ariola, editor, *Proc. 5th International Conference on Formal Structures for Computation and Deduction*, volume 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.12.

[HHL22] André Hirschowitz, Tom Hirschowitz, and Ambroise Lafont. Modules over monads and operational semantics (expanded version). *Logical Methods in Computer Science*, 18, 2022. doi:10.46298/lmcs-18(3:3)2022.

[HM07] André Hirschowitz and Marco Maggesi. Modules over monads and linearity. In *Proc. 14th International Workshop on Logic, Language, Information and Computation*, volume 4576 of *Lecture Notes in Computer Science*, pages 218–237. Springer, 2007. doi:10.1007/978-3-540-73445-1_16.

[HM10] André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Information and Computation*, 208(5):545–564, 2010. doi:10.1016/j.ic.2009.07.003.

[HM12] André Hirschowitz and Marco Maggesi. Nested abstract syntax in Coq. *Journal of Automated Reasoning*, 49(3):409–426, 2012. doi:10.1007/s10817-010-9207-9.

[Hue94] Gérard Huet. Residual theory in λ-calculus: a formal development. *Journal of Functional Programming*, 4(3):371–394, 1994. doi:10.1017/S0956796800001106.

[Laf22a] Ambroise Lafont. amblafont/binding-debruijn: Initial algebra semantics for De Bruijn monads in Coq, August 2022. doi:10.5281/zenodo.7022679.

[Laf22b] Ambroise Lafont. Initial Sigma-monoids for skew monoidal categories in UniMath, June 2022. doi:10.5281/zenodo.6622835.

[LS14] Stephen Lack and Ross Street. On monads and warpings. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, LV(4):244–266, 2014.

[Mag22] Marco Maggesi. Initial algebra semantics for De Bruijn monads in HOL Light, September 2022. doi:10.5281/zenodo.7053686.

[Pit23] Andrew M. Pitts. Locally nameless sets. *Proceedings of the ACM on Programming Languages*, 7(POPL), 2023. doi:10.1145/3571210.

[Plo90] Gordon Plotkin. An illative theory of relations. In R. Cooper et al., editors, *Situation Theory and its Applications*, number 22 in CSLI Lecture Notes, page 133–146. Stanford University, 1990.

[Pow07] John Power. Abstract syntax: Substitution and binders: Invited address. *Electronic Notes in Theoretical Computer Science*, 173:3–16, 04 2007. doi:10.1016/j.entcs.2007.02.024.

[Shu21] Michael Shulman. You could have invented De Bruijn indices. Blog post at *The n-Category Café*, https://golem.ph.utexas.edu/category/2021/08/you_could_have_invented_de_bru.html, 2021.

[SSK19] Kathrin Stark, Steven Schäfer, and Jonas Kaiser. Autosubst 2: reasoning with multi-sorted De bruijn terms and vector substitutions. In Assia Mahboubi and Magnus O. Myreen, editors, *Proc. 8th International Conference on Certified Programs and Proofs*, pages 166–180. ACM, 2019. doi:10.1145/3293880.3294101.

[SST15] Steven Schäfer, Gert Smolka, and Tobias Tebbi. Completeness and decidability of De Bruijn substitution algebra in Coq. In Xavier Leroy and Alwen Tiu, editors, *Proc. 4th International Conference on Certified Programs and Proofs*, pages 67–73. ACM, 2015. URL: http://dl.acm.org/citation.cfm?id=2676724, doi:10.1145/2676724.2693163.

[STS15] Steven Schäfer, Tobias Tebbi, and Gert Smolka. Autosubst: Reasoning with De Bruijn terms and parallel substitutions. In Christian Urban and Xingyuan Zhang, editors, *Proc. 6th International Conference on Interactive Theorem Proving*, volume 9236 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2015. doi:10.1007/978-3-319-22102-1_24.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus – A Theory of Mobile Processes*. Cambridge University Press, 2001.

[Szl12] Kornel Szlachányi. Skew-monoidal categories and bialgebroids. *Advances in Mathematics*, 231:1694–1730, 2012. doi:10.1016/j.aim.2012.06.027.