

Variable binding and substitution for (nameless) dummies

Ambroise LAFONT

Joint work with André & Tom Hirschowitz, Marco Maggesi

October 2021

A (new) theory of syntax for De Bruijn

Main definitions

- Operation of **binding arity** $\vec{k} \in \mathbb{N}^*$, in a **De Bruijn monad**.
- **Model** of a *binding signature*.

Our main characterisations

- Parallel substitution in the term model,
via recursive equations.
- Term model,
via *Initial Algebra Semantics*.

Theories of syntax

	No quotient	Mere set	Substitution
Nominal sets [Gabbay-Pitts '99]	×	✓	×
Substitution monoids [Fiore-Plotkin-Turi '99]	✓	×	✓
De Bruijn monads (our work)	✓	✓	✓

Nominal sets [Gabbay-Pitts '99]

- Named variables.
- ✗ Involves quotient (α -equivalence).
- ✗ No (built-in) substitution (only injective renamings)

Substitution monoids [Fiore-Plotkin-Turi '99]

- *Well-scoped syntax* = indexed by the number of free variables

Λ_n = terms with at most n free vars.

\Rightarrow Not a mere set of terms.

Our work: De Bruijn monads

- **Mere sets:** Simple enough to be formalised in HOL light
- Yet, essentially equivalent to the standard substitution monoids [Fiore-Plotkin-Turi '99]

Plan

- ① Binding arities
- ② Specification of substitution
- ③ Initial Algebra Semantics
- ④ Equations
- ⑤ Formalisation

What is a bound variable?

Scope of the question

Any syntax

- De Bruijn encoded;
- Specified by a **binding signature**.

Binding signatures

$$\text{app} : (0, 0), \quad \text{abs} : (1)$$

Binding signature for λ -calculus

Binding arity

$$\text{op} : \overbrace{(k_1, \dots, k_n)}^{\text{Binding arity}} \quad \Leftrightarrow \quad \text{op}(t_1, \dots, t_n) \text{ binds } k_i \text{ variables in } t_i,$$

Term model for a binding signature S

$$\begin{array}{ll} T_S \ni t_i ::= n & (n \in \mathbb{N}) \\ \quad | \text{op}(t_1, \dots, t_n) & \text{for each op} : (k_1, \dots, k_n) \in S \end{array}$$

What does it mean for $\text{op}(t_1, \dots, t_n)$ to bind k_i variables in t_i ?

Binding condition for abs : (1)

$$\forall \sigma : \mathbb{N} \rightarrow \Lambda,$$

$$(\lambda.t)[\sigma] = \lambda.(t[\uparrow\sigma])$$

0 is bound by λ in t :

$$\sigma : \mathbb{N} \rightarrow \Lambda$$

$$\uparrow\sigma : \mathbb{N} \rightarrow \Lambda$$

$$n \mapsto \begin{cases} 0 & \text{if } n = 0 \\ \sigma(n-1)[k \mapsto k+1] & \text{otherwise} \end{cases}$$

Binding condition for $\text{op} : (k_1, \dots, k_n)$

$$\text{op}(t_1, \dots, t_n)[\sigma] = \text{op}(t_1[\uparrow^{k_1} \sigma], \dots, t_n[\uparrow^{k_n} \sigma])$$

$0, \dots, k_i - 1$ are bound in t_i :

- $\uparrow^{k_i} \sigma$ preserves them;
- $\uparrow^{k_i} \sigma(p + k_i) = \sigma(p)[q \mapsto q + k_i]$.

Plan

- ① Binding arities
- ② Specification of substitution
- ③ Initial Algebra Semantics
- ④ Equations
- ⑤ Formalisation

Substitution is tricky

[Huet '94]

“We now define [unary] substitution. We again give first the intuitive recursive definition [...].”

$$M[0 := N] = \text{subst_rec } N \ M \ 0$$

Recursive Definition $\text{subst_rec } N \ M \ k = \text{Match } M \text{ with}$

```

  Ref(i)  -> if k<i then Ref(i-1)
            if k=i then lift k N
            if k>i then Ref(i)
| Abs(P)  -> Abs(subst_rec N P (k+1))
| App(P,Q) -> App(subst_rec N P k, subst_rec N Q k).
```

Can you trust your substitution?

Yes, by uniqueness.

Example: λ -calculus

$\exists! - [-] : \Lambda \times \Lambda^{\mathbb{N}}$ satisfying

- left unitality

$$n[\sigma] = \sigma(n),$$

- binding conditions for app / abs

$$(t \ u)[\sigma] = t[\sigma] \ u[\sigma] \qquad (\lambda.t)[\sigma] = \lambda.(t[\uparrow \sigma]).$$

Specification of substitution

General case, for any binding signature S

$\exists! - [-] : T_S \times (T_S)^{\mathbb{N}} \rightarrow T_S$ satisfying

- left unitality

$$n[\sigma] = \sigma(n),$$

- the binding condition for every $\text{op} : (k_1, \dots, k_n)$ in S

$$\text{op}(\dots, t_i, \dots)[\sigma] = \text{op}(\dots, t_i[\uparrow^{k_i} \sigma], \dots).$$

Plan

- ① Binding arities
- ② Specification of substitution
- ③ Initial Algebra Semantics
- ④ Equations
- ⑤ Formalisation

Initial Algebra Semantics

A general methodology for specification

X is **specified** by a **signature** $S \Leftrightarrow \begin{cases} \bullet X \text{ has a } S\text{-model structure.} \\ \bullet \text{ This } S\text{-model is } \mathbf{initial}. \end{cases}$

Example: \mathbb{N}

- S = endofunctor on Set:

$$S(Y) = Y + 1$$

- S -model = S -algebra.
- Initial S -algebra:

$$\mathbb{N} + 1 \xrightarrow{[succ, 0]} \mathbb{N}.$$

Initiality as a characterisation

- Initial object: unique (up to unique iso).
- Initiality \simeq recursion principle.

$$\begin{array}{l} \exists! f : \mathbb{N} \rightarrow Y \\ \left\{ \begin{array}{l} f(0) = f_0 \\ f(n+1) = E(f(n)) \end{array} \right. \end{array} \quad Y + 1 \xrightarrow{[E, f_0]} Y.$$

Models of a binding signature S

Models = **De Bruijn monads** with *compatible S -operations*.

Term model T_S = initial model.

De Bruijn monads: synthetic definitions

- DB monad = monad relative¹ to the functor $1 \rightarrow \mathbf{Set}$ picking \mathbb{N} .
- DB monad = monoid for the (skew) monoidal¹ product on sets:

$$X \otimes Y = X \times Y^{\mathbb{N}}.$$

¹[Altenkirch-Chapman-Uustalu '15] introduces relative monads and relates them to skew monoids.

De Bruijn monads: analytic definition

Components of a DB monad $(X, -[-], var)$

Set	X
Substitution map	$-[-] : X \times X^{\mathbb{N}} \rightarrow X$
Variables map	$var : \mathbb{N} \rightarrow X$

Equations satisfied by a DB monad

Left unitality	$var(n)[\sigma] = \sigma(n)$
Right unitality	$t[n \mapsto var(n)] = t$
Associativity	$t[\sigma][\delta] = t[n \mapsto \sigma(n)[\delta]]$

Examples

- λ -calculus (De Bruijn encoding).
- T_S for any binding signature S .
- Restriction of a monad T on \mathbf{Set} :

Set	$T(\mathbb{N})$
Substitution	$\text{bind} : T(\mathbb{N}) \times T(\mathbb{N})^{\mathbb{N}} \rightarrow T(\mathbb{N})$
Variables	$\text{ret} : \mathbb{N} \rightarrow T(\mathbb{N})$

Monads from DB monads

DB monad $X \mapsto$ monad $\overline{X} : \text{Set} \rightarrow \text{Set}$

$$\overline{X}(\mathbb{N}) = X$$

$$\begin{aligned}
 x \in \overline{X}(n) \subset X &\Leftrightarrow x \text{ **has support** } n \\
 &\Leftrightarrow \text{if } \sigma \text{ fixes the first } n \text{ variables,} \\
 &\quad \text{then } x[\sigma] = x.
 \end{aligned}$$

Equivalence with *well-behaved* monads

$$\text{Monads on Set} \begin{array}{c} \xrightarrow{T \mapsto T(\mathbb{N})} \\ \xleftarrow{\overline{X} \mapsto X} \end{array} \text{DB monads}$$

restricts to an equivalence

$$\underbrace{\text{Well-behaved monads}}_{\substack{\text{Finitary monads } T \text{ on Set} \\ \text{preserving binary intersections}}} \simeq \underbrace{\text{DB monads with finite support}}_{\substack{\text{DB monads } X \text{ s.t.} \\ \text{every } x \in X \text{ has a support } n.}}$$

Well-behaved monads: an equivalent definition

Proposition [Trnková '69]

Any endofunctor F on \mathbf{Set} preserves non empty binary intersections.

Corollary

Let T be a finitary monad on \mathbf{Set} .

Then, T is well-behaved iff it preserves empty binary intersections.

Well-behaved monads: a sufficient condition

Proposition [Adámek-Milius-Bowler-Levy '12]

A monad on \mathbf{Set}

- either preserves the initial object,
- either preserves empty binary intersections.

Corollary

Let T be a finitary monad on \mathbf{Set} s.t. $T(\emptyset) \neq \emptyset$.
Then, T is well-behaved.

A link with substitution monoids [Fiore-Plotkin-Turi '99]

The previous equivalence

Well-behaved monads \simeq DB monads with *finite support*

lifts to

Well-behaved S -monoids \simeq **S -models** with finite support

Well-behaved monads T with
compatible S -operations

Models of a binding signature S

Definition of S -models

DB monad + an **operation of binding arity** $\vec{k} \in \mathbb{N}^*$
for each $\text{op} : \vec{k} \in S$.

Operation of binding arity $\vec{k} \in \mathbb{N}^n$ in a DB monad $(X, -[-], \text{var})$

$$\text{op} : X^n \rightarrow X$$

satisfying the \vec{k} -binding condition:

$$\text{op}(t_1, \dots, t_n)[\sigma] = \text{op}(t_1[\uparrow^{k_1} \sigma], \dots, t_n[\uparrow^{k_n} \sigma]).$$

Example of λ -calculus

Binding signature of λ -calculus

$$\text{app} : (0, 0) \quad \text{abs} : (1)$$

Models of λ -calculus

$$(X, -[-], \text{var}) \quad \text{app} : X \times X \rightarrow X \quad \text{abs} : X \rightarrow X$$

satisfying the binding conditions, i.e.,

$$\text{app}(t, u)[\sigma] = \text{app}(t[\sigma], u[\sigma]) \quad \text{abs}(t)[\sigma] = \text{abs}(t[\uparrow\uparrow \sigma]).$$

Initial Algebra Semantics for a binding signature S

Reminder: specification of substitution

$\exists ! - [-] : T_S \times T_S^{\mathbb{N}} \rightarrow T_S$ compatible with

- variables (*left unitality*);
- every $\text{op} : (k_1, \dots, k_n)$ in S (*binding conditions*).

Moreover,

- $(T_S, \text{var}, -[-])$ is a DB monad
(i.e., right unitality and associativity hold).
- The induced S -model is **initial**.

Plan

- ① Binding arities
- ② Specification of substitution
- ③ Initial Algebra Semantics
- ④ Equations
- ⑤ Formalisation

Models of λ -calculus modulo β/η -equation

Λ -models X s.t.

$$\eta : \forall t \in X, \quad \text{abs}(\text{app}(t[n \mapsto n+1], 0)) = t$$

$$\beta : \forall (t, u) \in X \times X, \quad \text{app}(\text{abs}(t), u) = t[0 \mapsto u, n+1 \mapsto n]$$

Structure of an equation

$$\forall \vec{t} \in X^n, \text{lhs}_X(\vec{t}) = \text{rhs}_X(\vec{t})$$

lhs, rhs:

- defined on any model X ;
- operations of some binding arity (a_1, \dots, a_n) ,

e.g., for β , lhs, rhs : $(1, 0)$.

Signature S_{eq} with equations

Components of S_{eq}	Example: $\Lambda_{\beta\eta}$
a binding signature S	$\text{app} : (0, 0) \quad \text{abs} : (1)$
pairs (lhs, rhs) of operations defined on any S -model	$(\text{lhs}_\eta, \text{rhs}_\eta)$ $(\text{lhs}_\beta, \text{rhs}_\beta)$

Models: S -models X s.t. $\text{lhs}(x) = \text{rhs}(x)$ for all (lhs, rhs) in S_{eq} .

Initial model: T_S quotiented by $\text{lhs}(x) \sim \text{rhs}(x) + \text{congruences}$.

Plan

- ① Binding arities
- ② Specification of substitution
- ③ Initial Algebra Semantics
- ④ Equations
- ⑤ Formalisation

Formalisation of DB monads

- In Coq (2400 LoC) and HOL Light (without dependent types).
- Untyped case.
- Specification of substitution.
- Initial Algebra Semantics.
 - Binding signatures.
 - (Coq) Equations (with axiomatised quotients).

De Bruijn syntax in Coq

Binding signatures

```
Record signature :=
{ 0 : Type;
  ar : 0 → list ℕ }.
```

Syntax = one single (parameterised) inductive type

```
Inductive T (S : signature) : Type :=
  Var : ℕ → T S
| Op : ∀ (o : 0 S), vec (T S) (ar o) → T S.
      (* ^^ this is T^|ar o| *)
```

De Bruijn syntax in HOL Light

- Particular cases: λ -calculus, modulo β/η .
- General case: raw syntax + wellformed predicates.

Conclusion

A simple theory of syntax

- For De Bruijn representation.
- Essentially equivalent to the substitution monoids of [FPT '99].
- Simple enough to be mechanised without dependent types.
- Simply-typed syntax.