Variable binding and substitution for (nameless) dummies

Ambroise LAFONT
Joint work with André & Tom Hirschowitz, Marco Maggesi

November 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

$$\underbrace{\text{app} : (0,0), \qquad \text{abs} : (1)}_{\textbf{Binding signature} \text{ for } \lambda\text{-calculus}}$$

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

Term model for a binding signature S

$$
\begin{aligned}
T_S \ni t_i ::= {} & n && (n \in \mathbb{N}) \\
\mid {} & \mathrm{op}(t_1, \ldots, t_n) && \text{for each } \mathrm{op} : (k_1, \ldots, k_n) \in S
\end{aligned}
$$

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

Substitution crossing a binder

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

$$(\lambda.t)[\sigma] = \mathbf{?}$$

Variables after λ

- 0 is bound:

$$(\lambda.0)[\sigma] = \lambda.0$$

- $n+1$ refers to the free variable n:

$$(\lambda.1)[\sigma] = \lambda.(\sigma(0)[k \mapsto k+1])$$

Binding condition for abs : (1)

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

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

$$\begin{aligned} \Uparrow\sigma : \mathbb{N} &\to \Lambda \\ 0 &\mapsto 0 \\ n+1 &\mapsto \sigma(n)[k \mapsto k+1] \end{aligned}$$

Binding condition for op : $(k_1, \ldots, k_n)$

$$\mathsf{op}(t_1, \ldots, t_n)[\sigma] = \mathsf{op}(t_1[\Uparrow^{k_1} \sigma], \ldots, t_n[\Uparrow^{k_n} \sigma])$$

$0, \ldots, k_i - 1$ 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 [...]."

$$M[0 := N] = \texttt{subst\_rec N M 0}$$

```
Recursive Definition subst_rec N M k = Match M 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

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

- left unitality

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

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

$$\mathsf{op}(\ldots, t_i, \ldots)[\sigma] = \mathsf{op}(\ldots, t_i[\Uparrow^{k_i} \sigma], \ldots).$$

Plan

Initial Algebra Semantics

A general methodology for specification

X is **specified** by a **signature** S $\Leftrightarrow$
- X has a S-**model** structure.
- This S-model is **initial**.

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.

$$\exists!\ f : \mathbb{N} \to Y$$
$$\begin{cases} f(0) & = f_0 \\ f(n+1) & = E(f(n)) \end{cases} \qquad Y + 1 \xrightarrow{[E, f_0]} Y.$$

Models of a binding signature S

$$
\begin{aligned}
\text{Models} \quad &= \textbf{De Bruijn monads} \text{ with } \textit{compatible } S\textit{-operations}. \\
\text{Term model } T_S \quad &= \text{initial model.}
\end{aligned}
$$

De Bruijn monads: synthetic definitions

- DB monad = monad relative[1] to the functor $1 \to \mathrm{Set}$ picking $\mathbb{N}$.
- DB monad = monoid for a skew monoidal[1] structure on sets.

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

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 Set:

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

Monads from DB monads

$$\text{DB monad } X \quad \mapsto \quad \text{monad } \overline{X} : \mathsf{Set} \to \mathsf{Set}$$

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

$$x \in \overline{X}(\{0, \ldots, n-1\}) \subset X \quad \Leftrightarrow \quad x \textbf{ has support } n$$

$$\Leftrightarrow \quad \text{if } \sigma \text{ fixes the first } n \text{ variables, then } x[\sigma] = x.$$

Equivalence with *well-behaved* monads

$$\text{Monads on Set} \underset{\overline{X} \leftarrow\!\shortmid X}{\overset{T \mapsto T(\mathbb{N})}{\rightleftarrows}} \text{DB monads}$$

restricts to an equivalence

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

Well-behaved monads: an equivalent definition

Proposition [Trnková '69]

Any endofunctor F on Set preserves non empty binary intersections.

Corollary

Let T be a finitary monad on 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 Set

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

Corollary

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

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

The previous equivalence

$$\textit{Well-behaved}\text{ monads} \quad \simeq \quad \text{DB monads with } \textit{finite support}$$

lifts to

$$\underbrace{\textit{Well-behaved } S\text{-monoids}}_{\substack{\text{Well-behaved monads } T \text{ with} \\ \text{compatible } S\text{-operations}}} \quad \simeq \quad S\text{-}\textbf{models}\text{ with finite support}$$

Models of a binding signature S

Definition of S-models

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

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

$$\mathsf{op} : X^n \to X$$

satisfying the $\vec{k}$-binding condition:

$$\mathsf{op}(t_1, \ldots, t_n)[\sigma] = \mathsf{op}(t_1[\Uparrow^{k_1} \sigma], \ldots, t_n[\Uparrow^{k_n} \sigma]).$$

Example of λ-calculus

Binding signature of λ-calculus

$$\mathsf{app} : (0,0) \qquad \mathsf{abs} : (1)$$

Models of λ-calculus

$$(X, -[-], \mathsf{var}) \qquad \mathsf{app} : X \times X \to X \qquad \mathsf{abs} : X \to X$$

satisfying the binding conditions, i.e.,

$$\mathsf{app}(t,u)[\sigma] = \mathsf{app}(t[\sigma], u[\sigma]) \qquad \mathsf{abs}(t)[\sigma] = \mathsf{abs}(t[\Uparrow \sigma]).$$

Initial Algebra Semantics for a binding signature S

Reminder: specification of substitution

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

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

Moreover,

- $(T_S, var, -[-])$ is a DB monad

  (i.e., right unitality and associativity hold).
- The induced S-model is **initial**.

Plan

Models of λ-calculus modulo β/η-equation

Λ-models X s.t.

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

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

Structure of an equation

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

lhs, rhs:

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

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	app : $(0,0)$ abs : (1)
pairs (lhs, rhs) of operations defined on any S-model	$(\mathsf{lhs}_\eta, \mathsf{rhs}_\eta)$ $(\mathsf{lhs}_\beta, \mathsf{rhs}_\beta)$

Models: S-models X s.t. $\mathsf{lhs}(x) = \mathsf{rhs}(x)$ for all (lhs, rhs) in S_{eq}.
Initial model: T_S quotiented by $\mathsf{lhs}(x) \sim \mathsf{rhs}(x)$ + 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) Signatures with equations (with axiomatised quotients).

De Bruijn syntax in Coq

Binding signatures

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

Syntax = one single (parameterised) inductive type

```
Inductive T (S : signature) : Type :=
  Var : ℕ → T S
| Op  : ∀ (o : O 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.
- Extends to
  - simply-typed syntax;
  - signatures with equations (e.g., λ-calculus modulo β and η).