Variable binding and substitution for (nameless) dummies

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

FoSSaCS 2022

A simple theory for De Bruijn encoded syntax with substitution

	No quotient (α-equivalence)	No indexing	Built-in substitution
Substitution monoids [Fiore-Plotkin-Turi '99]	✓	× (indexed by fv)	✓
Nominal sets [Gabbay-Pitts '99]	×	✓	×
De Bruijn monads (our work)	✓	✓	✓

Plan

1. Bound variables and binding conditions
2. Initial Algebra Semantics
3. Equivalence with substitution monoids [Fiore-Plotkin-Turi '99]

Goal of this section

Answering the question

What is a bound variable?

... in a (De Bruijn) syntax specified by a **binding signature**.

Binding signatures [Plotkin '90]

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

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

Syntax for a binding signature S

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

Remarks

- No quotient by α-equivalence.
- Binding is not explicit:

$$\mathsf{op} : (k_1, \ldots, k_n) \quad \Leftrightarrow \quad \mathsf{op}(t_1, \ldots, t_n) \text{ binds } k_i \text{ variables in } t_i,$$

but the syntax does not depend on the k_i

Where is binding involved?

Variables in $\lambda.t$

De Bruijn encoding

- Crossing $\lambda \Rightarrow$ free variable shifting

$$\underbrace{n \in \mathbb{N}_o}_{\text{outside}} \quad \leftrightarrow \quad \underbrace{n+1 \in \mathbb{N}_i}_{\text{inside}}$$

- $0 \in \mathbb{N}_i$ = bound variable

How does it impact substitution of free variables?

Substitution crossing a binder

$\forall \sigma_o : \mathbb{N}_o \to \Lambda_o,$

$$(\lambda.t)[\sigma_o] = \lambda.(t[\sigma_i])$$

$$\begin{aligned} \sigma_i : \mathbb{N}_i &\to \Lambda_i \\ 0 &\mapsto 0 \\ \underbrace{n+1}_{\mathbb{N}_i} &\mapsto \underbrace{n}_{\mathbb{N}_o} \mapsto \underbrace{\sigma(n)}_{\Lambda_o} \mapsto \underbrace{\sigma(n)[k \mapsto k+1]}_{\Lambda_i} \end{aligned}$$

Notation

$$\sigma_i := \Uparrow \sigma_o$$

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

Initial Algebra Semantics

A general methodology for specification

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

$\Longrightarrow$ Needs an adequate notion of S-model.

Features of initiality

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

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

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

[1] Altenkirch-Chapman-Uustalu ['15] introduce 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 of DB monads

- λ-calculus (De Bruijn encoding).
- T_S for any binding signature S.
- Any monad on sets induces a DB monad (Cf next section)

Initial Algebra Semantics for a binding signature S

Models	$=$ De Bruijn monads with **compatible S-operations**.
Term model T_S	$=$ initial model.

DB monad $(X, -[-], var)$ with compatible S-operations

Definition

For each op : $(k_1, \ldots, k_n) \in S$, an operation

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

satisfying the binding condition:

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

Initial Algebra Semantics for a binding signature S

Models	$=$ De Bruijn monads with compatible S-operations.
Term model T_S	$=$ **initial model**.

Initiality of the term model T_S

Proof steps

1. Define
$$-[-] : T_S \times T_S^{\mathbb{N}} \to T_S$$
so that $(T_S, var, -[-])$ is a DB monad;
2. Show that T_S has compatible S-operations;
3. Show that the induced S-model is initial.

Next slides: describe the induced term model.

The term model for λ-calculus

Recursive definition of substitution

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

- Left unitality

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

- Binding conditions for app / abs

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

Proving that $(\Lambda, var, -[-])$ is a DB monad

Right unitality and associativity hold (by induction).

$\Rightarrow$ Induces a S_Λ-model.

The term model T_S for a binding signature S

Recursive definition of substitution

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

- Left unitality

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

- The binding condition for every op : $(k_1, \ldots k_n)$ in S

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

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

Right unitality and associativity hold (by induction).

$\Rightarrow$ Induces a S-model.

Plan

1. Bound variables and binding conditions
2. Initial Algebra Semantics
3. Equivalence with substitution monoids [Fiore-Plotkin-Turi '99]

Equivalence with monads

$$\underbrace{\textit{Well-behaved monads } T \text{ on sets}}_{\substack{T \text{ is finitary and} \\ \text{preserves binary intersections}}} \quad \simeq \quad \underbrace{\text{DB monads } X \text{ with } \textit{finite support}}_{\substack{\text{For every } x \in X, \exists n \text{ s.t.} \\ x \textbf{ has support } n.}}$$

Finite support

$$x \textbf{ has support } n \quad \Leftrightarrow \quad \text{“}fv(x) \subset \{0, \ldots, n-1\}\text{”}$$

More formally,

if $\sigma : \mathbb{N} \to X$ fixes the first n variables,

then σ fixes x, i.e, $x[\sigma] = x$

DB monads from monads

Restricting a monad T on sets:

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

$$\text{DB monad } X \xrightarrow{\text{standard categorical construction (Lan)}} \text{monad } \overline{X} : \mathsf{Set} \to \mathsf{Set}$$

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

$$\overline{X}(\{0, \dots, n-1\}) = \{x \in X \mid x \text{ has support } n\}$$

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

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

Conclusion

A simple theory of syntax

- For De Bruijn representation.
- Simple enough to be mechanised without dependent types.
- $\simeq$ Substitution monoids [FPT '99].
- Extensions:
  - Simply-typed syntax;
  - Signatures with equations (e.g., λ-calculus modulo β and η).