Terms with Bindings as an Abstract Data Type

Jasmin Blanchette, Lorenzo Gheri, Andrei Popescu, Dmitriy Traytel

Vrije Universiteit Amsterdam  Middlesex University London  ETH Zürich
Terms of the $\lambda$-calculus

Var infinite set of variables, ranged over by $x, y, z$ etc.

The set $\text{Tr}$ of $\lambda$-terms, ranged over by $t, s$ etc., defined by grammar:

$$t ::= Vr\; x \mid Ap\; t_1\; t_2 \mid Lm\; x\; t$$

(with often omit the injection $Vr$ of variables into terms)

... with the addition that terms are equated (identified) modulo $\alpha$-equivalence (a.k.a. naming equivalence)

E.g., $Lm\; x\; x$ is considered equal to $Lm\; y\; y$
Tr as an abstract data type (ADT), usefully

Tr endowed with an algebraic structure, given by operators such as:

- the constructors \( Vr, Ap, Lm \)
- (capture-avoiding) substitution
  \( \_ [\_ / \_] : Tr \to Tr \to \text{Var} \to Tr \)
e.g., \( (Lm x (Ap x y)) [Ap x x / y] = Lm x' (Ap x' (Ap x x)) \)
- swapping \( \_ [\_ \land \_] : Tr \to \text{Var} \to \text{Var} \to Tr \)
e.g., \( (Lm x (Ap x y)) [x \land y] = Lm y (Ap y x) \)
- (finite) permutation
  \( \text{Perm} = \{ \sigma : Vr \to Vr \mid \{ x \mid \sigma x \neq x \} \text{ finite} \} \)
  \( \_ [\_] : Tr \to \text{Perm} \to Tr \)
e.g., \( (Lm x (Ap x y)) [x \mapsto y, y \mapsto z] = Lm y (Ap y z) \)
- freshness \( \_ \text{fresh} \_ : \text{Var} \to Tr \to \text{Bool} \)
e.g., \( x \text{ fresh Lm x x} \)
Tr as an abstract data type (ADT), usefully

Properties of the term algebra

- Various basic properties of the operators, e.g.,
  1. $x$ fresh $t$ implies $x$ fresh $s[t/x]$
  2. $\{x \in \text{Var} \mid \neg x$ fresh $t\}$ is finite

- Reasoning principle – induction
  ... can prove facts of interest with reasonably small effort

- Definition principle – recursion
  ... can define functions of interest with reasonably small effort

A subset of the above will characterize the Tr algebra uniquely up to isomorphism.
The particular representation does not matter: quotient, de Bruijn, weak/strong HOAS, locally named/nameless – it’s the same Platonic concept!

The correct measuring for an approach to syntax with bindings should not be representation, but the end product:

- How expressive/useful are the (inductive) reasoning and (recursive) definition principles?
- How expressive and modular is the construction of binding structures?
Focus: recursion principles

We want such principles to be:

▶ Expressive: cover functions of interest, cover complex binding structures

▶ Easy to use: not require complex verifications in order to go through
1. Work with the free datatype of raw terms (no $\alpha$-equivalence)

$$t ::= Vr \times | Ap \ t_1 \ t_2 | Lm \times t$$

Advantage: Can immediately define in proof assistants as standard datatypes:

$$\text{datatype Tr = Vr Var | Ap Tr Tr | Lm Var Tr}$$

This gives the standard free recursor.

Major disadvantages:

- Substitution is no well-behaved
- Most of the times we would need to prove that the function is invariant under $\alpha$-equivalence—which is usually very complex
2. Work with a de a Bruijn encoding

\[ t ::= n \mid \text{Ap} \ t_1 \ t_2 \mid \text{DBLm} \ t \]

\( \lambda \)-abstraction takes no variable input, bound variables replaced by numbers which indicate which \( \lambda \) binds them.

Advantage: again, a free datatype

Major disadvantages:

- Dangling references DBLm \( 3 \)—number \( 3 \) refers to non-existing DBLm in the term
- Recursor talks about a fixed variable to be bound (via DBLm)
- In the end still must define a proper Lm, or keep encoding everything painfully using DBLm

But see some intelligent workarounds: Saving de Bruijn (Norris/Vestergaard 2007), Locally nameless (Charguéraud 2012), Functor categories (Fiore et al. 1999)
3. Regard abstraction as taking a function as input

Despeyroux et. al 1995 (weak HOAS), Gordon/Melham 1996
Regard terms as a subset of the datatype:

datatype Termoid = Vr Var | Ap Termoid Termoid | LLm (Var → Termoid)

Then Lm x t is defined as LLm (λy. t[y/x]). Proper subset: E.g.,
LLm(λx. if x = y then ... else ...) not a correct term.
Advantage: again, free-datatype recursor
Disadvantages:

▶ Use LLm applied to restricted function space instead of Lm
▶ Cannot easily define useful functions
Some not very useful recursion principles

Summary of the disadvantages:

- The recursor inherited from waw-term encodings suffers from lack of abstraction (notably substitution not well behaved)
- The recursor inherited from de Bruijn or functional (weak HOADS) encodings must replace the standard $\lambda$-abstraction with a different primitive
Some more useful recursion principles

The Nominal Logic recursion principle

Michael Norrish’s improvement

Our own contribution: the world’s best recursor for bindings
Preliminaries: basic properties of terms

Freshness versus constructors

(Fr1) \( z \neq x \implies z \text{ fresh } Vr x \)

(Fr2) \( z \text{ fresh } s \implies z \text{ fresh } t \implies z \text{ fresh } Ap s t \)

(Fr3) \( z = x \lor z \text{ fresh } t \implies z \text{ fresh } Lm x t \)

Swapping versus constructors

(SwVr) \( (Vr x) [z_1 \land z_2] = Vr (x[z_1 \land z_2]) \)

(SwAp) \( (Ap s t) [z_1 \land z_2] = Ap (s [z_1 \land z_2]) (t [z_1 \land z_2]) \)

(SwLm) \( (Lm x t) [z_1 \land z_2] = Lm (x [z_1 \land z_2]) (t [z_1 \land z_2]) \)

Algebraic properties of swapping

(SwId) \( t [z \land z] = t \)

(SwInv) \( t [z \land y] [x \land y] = t \)

(SwCmp) \( t [x \land y] [z_1 \land z_2] = (t [z_1 \land z_2]) [(x [z_1 \land z_2]) \land (y [z_1 \land z_2])] \)
Permutation versus constructors

(PmVr) \((Vr \times) \[\sigma\] = Vr (\sigma \times)\)

(PmAp) \((Ap s t) \[\sigma\] = Ap (s [\sigma]) (t [\sigma])\)

(PmLm) \((Lm \times t) \[\sigma\] = Lm (\sigma \times) (t [\sigma])\)

Algebraic properties of permutation

(PmId) \(t [\text{id}] = t\)

(PmComp) \(t [\sigma] [\tau] = t [\tau \circ \sigma]\)
Substitution versus constructors

(Sb1) \((Vr \, x) \, [s/z] = \text{if } x = z\text{ then } s\text{ else } Vr \, x\)

(Sb2) \((Ap \, t_1 \, t_2) \, [s/z] = Ap \, (t_1 \, [s/z]) \, (t_2 \, [s/z])\)

(Sb3) \(x \neq z \Rightarrow x \text{ fresh } z \Rightarrow (Lm \times t) \, [s/z] = Lm \times (t \, [s/z])\)

Abstraction rules

(CongSw) \(z \not\in \{z_1, z_2\} \Rightarrow z \text{ fresh } t_1, t_2 \Rightarrow t_1[z\wedge x_1] = t_2[z\wedge x_1] \Rightarrow Lm \, x_1 \, t_1 = Lm \, x_2 \, t_2\)

(CongSb) \(z \not\in \{z_1, z_2\} \Rightarrow z \text{ fresh } t_1, t_2 \Rightarrow t_1[z/x_1] = t_2[z/x_1] \Rightarrow Lm \, x_1 \, t_1 = Lm \, x_2 \, t_2\)
Preliminaries: basic properties of terms IV

Finite support
(FinSupp) \( \exists X. \ X \text{ finite and } \forall x, y \notin X. \ t[x \mapsto y, y \mapsto x] = t \)

Definability of freshness from permutations
(FrPm) \( x \text{ fresh } t \iff \{ y \mid t[x \mapsto y, y \mapsto x] \neq t \} \text{ finite} \)

Definability of freshness from swapping
(FrSw) \( x \text{ fresh } t \iff \{ y \mid t[x \land y] \neq t \} \text{ finite} \)

Finiteness of freshness
(FrFin) \( \{ x \mid x \text{ fresh } t \} \text{ finite} \)

Freshness condition for binders (barebone version)
(FCB) \( \exists x. \ x \text{ fresh } Lm \times t \)
Recursion principles: barebone versions

Tr forms the initial model w.r.t. the following classes of models:

Main results:

<table>
<thead>
<tr>
<th>1) Pitts</th>
<th>2) Norrish</th>
<th>3) Our results</th>
</tr>
</thead>
<tbody>
<tr>
<td>PmVr PmAp PmLm</td>
<td>SwVr SwAp SwLm</td>
<td>SwVr SwAp SwLm</td>
</tr>
<tr>
<td>PmId PmComp</td>
<td>SwId SwComp SwInv</td>
<td>SwCong</td>
</tr>
<tr>
<td>FrPm FCB</td>
<td>FrVr FrAp FrLm</td>
<td>FrVr FrAp FrLm</td>
</tr>
<tr>
<td>FinSupp</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variations:

<table>
<thead>
<tr>
<th>4) Pitts-Norrish hybrid</th>
<th>5) Our results</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwVr SwAp SwLm</td>
<td>SwVr SwAp SwLm</td>
</tr>
<tr>
<td>SwId SwComp SwInv</td>
<td>SwRen</td>
</tr>
<tr>
<td>FrSw FCB</td>
<td>FrVr FrAp FrLm</td>
</tr>
</tbody>
</table>

Expressiveness:

\[ 1 = 4 < 2 < 5 < 3 \]