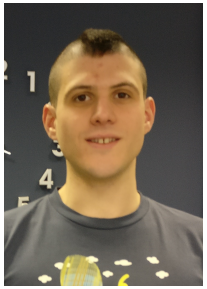
Terms with Bindings as an Abstract Data Type

Jasmin Blanchette, Lorenzo Gheri, Andrei Popescu, Dmitriy Traytel



Vrije Universiteit Amsterdam Middlesex University London ETH Zürich

Terms of the λ-calculus

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

The set Tr of λ-terms, ranged over by t, s etc., defined by grammar:

$$t \ ::= \ \mathsf{Vr}\ x \ \mid \ \mathsf{Ap}\ t_1\ t_2 \ \mid \ \mathsf{Lm}\ x\ t$$

(will often omit the injection Vr of variables into terms)
... with the addition that terms are equated (identified) modulo α-equivalence (a.k.a. naming equivalence)

E.g., Lm x x is considered <u>equal</u> 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
  $\_[\_/\_] : \mathsf{Tr} \to \mathsf{Tr} \to \mathsf{Var} \to \mathsf{Tr}$
  e.g., $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [\mathsf{Ap}\ x\ x\ /\ y] = \mathsf{Lm}\ x'\ (\mathsf{Ap}\ x'\ (\mathsf{Ap}\ x\ x))$
- swapping $\_[\_ \wedge \_] : \mathsf{Tr} \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Tr}$
  e.g., $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [x \wedge y] = \mathsf{Lm}\ y\ (\mathsf{Ap}\ y\ x)$
- (finite) permutation
  $\mathsf{Perm} = \{\sigma : \mathsf{Vr} \to \mathsf{Vr} \mid \{x \mid \sigma\ x \neq x\}\ \text{finite}\ \}$
  $\_[\_] : \mathsf{Tr} \to \mathsf{Perm} \to \mathsf{Tr}$
  e.g., $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [x \mapsto y, y \mapsto z] = \mathsf{Lm}\ y\ (\mathsf{Ap}\ y\ z)$
- freshness $\_\mathsf{fresh}\_ : \mathsf{Var} \to \mathsf{Tr} \to \mathsf{Bool}$
  e.g., $x\ \mathsf{fresh}\ \mathsf{Lm}\ x\ x$

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. $\{x \in \mathsf{Var} \mid \neg\, x \text{ fresh } t\}$ 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.

Parenthesis: ADT vs representation issues

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 α-equivalence)

$$t \ ::= \ \mathsf{Vr}\ x \ \mid \ \mathsf{Ap}\ t_1\ t_2 \ \mid \ \mathsf{Lm}\ x\ t$$

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

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

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 α-equivalence—which is usually very complex

2. Work with a de a Bruijn encoding

$$t \ ::= \ n \ | \ \text{Ap}\ t_1\ t_2 \ | \ \text{DBLm}\ t$$

λ-abstraction takes no variable input, bound variables replaced by numbers which indicate which λ binds them.

Advantage: again, a free datatype

Major disadvantages:

- Dangling references DBLm 3—n umber 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 (Norrish/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 $\rightarrow$ Termoid)

Then Lm x t is defined as LLm ($\lambda y.\ t[y/x]$). 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 substituion not well behaved)
- The recursor inherited from de Bruijn or functional (weak HOADS) encodings must replace the standard λ-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) $z \neq x \;\Rightarrow\; z \text{ fresh } \mathsf{Vr}\ x$
(Fr2) $z \text{ fresh } s \;\Rightarrow\; z \text{ fresh } t \;\Rightarrow\; z \text{ fresh } \mathsf{Ap}\ s\ t$
(Fr3) $z = x \;\vee\; z \text{ fresh } t \;\Rightarrow\; z \text{ fresh } \mathsf{Lm}\ x\ t$

Swapping versus constructors
(SwVr) $(\mathsf{Vr}\ x)\,[z_1 \wedge z_2] = \mathsf{Vr}\ (x[z_1 \wedge z_2])$
(SwAp) $(\mathsf{Ap}\ s\ t)\,[z_1 \wedge z_2] = \mathsf{Ap}\ (s\,[z_1 \wedge z_2])\ (t\,[z_1 \wedge z_2])$
(SwLm) $(\mathsf{Lm}\ x\ t)\,[z_1 \wedge z_2] = \mathsf{Lm}\ (x\,[z_1 \wedge z_2])\ (t\,[z_1 \wedge z_2])$

Algebraic properties of swapping
(SwId) $t\,[z \wedge z] = t$
(SwInv) $t\,[z \wedge y]\,[x \wedge y] = t$
(SwCmp) $t\,[x \wedge y]\,[z_1 \wedge z_2] = (t\,[z_1 \wedge z_2])\,[(x\,[z_1 \wedge z_2]) \wedge (y\,[z_1 \wedge z_2])]$

Preliminaries: basic properties of terms II

Permutation versus constructors

(PmVr) $(\mathsf{Vr}\ x)\,[\sigma] = \mathsf{Vr}\ (\sigma\ x)$

(PmAp) $(\mathsf{Ap}\ s\ t)\,[\sigma] = \mathsf{Ap}\ (s\,[\sigma])\ (t\,[\sigma])$

(PmLm) $(\mathsf{Lm}\ x\ t)\,[\sigma] = \mathsf{Lm}\ (\sigma\ x)\ (t\,[\sigma])$

Algebraic properties of permutation

(PmId) $t\,[\mathrm{id}] = t$

(PmComp) $t\,[\sigma]\,[\tau] = t\,[\tau \circ \sigma]$

Preliminaries: basic properties of terms III

Substitution versus constructors

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

(Sb2) $(\mathsf{Ap}\ t_1\ t_2)\,[s/z] = \mathsf{Ap}\,(t_1\,[s/z])\,(t_2\,[s/z])$

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

Abstraction rules

(CongSw) $z \notin \{z_1, z_2\} \Rightarrow z \text{ fresh } t_1, t_2 \Rightarrow t_1[z \wedge x_1] = t_2[z \wedge x_1] \Rightarrow$
$\mathsf{Lm}\ x_1\ t_1 = \mathsf{Lm}\ x_2\ t_2$

(CongSb) $z \notin \{z_1, z_2\} \Rightarrow z \text{ fresh } t_1, t_2 \Rightarrow t_1[z/x_1] = t_2[z/x_1] \Rightarrow$
$\mathsf{Lm}\ x_1\ t_1 = \mathsf{Lm}\ x_2\ t_2$

Preliminaries: basic properties of terms IV

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

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

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

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

Freshness condition for binders (barebone version)
(FCB) $\exists x.\ x$ fresh Lm $x\ t$

Recursion principles: barebone versions

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

Main results:

1) Pitts	2) Norrish	3) Our results
PmVr PmAp PmLm	SwVr SwAp SwLm	SwVr SwAp SwLm
PmId PmComp	SwId SwComp SwInv	SwCong
FrPm FCB	FrVr FrAp FrLm	FrVr FrAp FrLm
~~FinSupp~~		

Variations:

4) Pitts-Norrish hybrid	5) Our results
SwVr SwAp SwLm	SwVr SwAp SwLm
SwId SwComp SwInv	SwRen
FrSw FCB	FrVr FrAp FrLm

Expressiveness:
$1 = 4 < 2 < 5 < 3$