

Pragmatic reasoning about languages with binding?

Randy Pollack

Version of December 21, 2004

Needed: A challenge problem set

Some Naturally Occurring Problems

Freshness of Globally Bound Variables

Simultaneous Substitutions, Environments, Infinite Structures

Approaches to Formalizing Binding

First Order Representations

Logics with “freshness”

Higher Order Abstract Syntax (HOAS)

Weak HOAS

Needed: A challenge problem set

Many technical approaches proposed to reason formally about languages with binding

- ▶ Higher Order Abstract Syntax (HOAS)
 - ▶ LF (Harper, Honsell, Plotkin)
 - ▶ Hybrid (Ambler, Crole, Momigliano)
 - ▶ Twelf (Pfenning, Shürmann)
 - ▶ Miller/McDowell/Tiu
- ▶ Weak HOAS
 - ▶ Context Calculus (Honsell, Miculan)
 - ▶ Desperoux/Felty/Hirschowitz
- ▶ “Freshness” approaches
 - ▶ FM, nominal (Pitts, Gabbay, Urban, Cheney)
 - ▶ Schoepp/Stark
- ▶ Concrete approaches
 - ▶ de Bruijn, Stoughton, McKinna/Pollack, Gordon/Melham, . . .

Is it feasible to reason formally about languages with binding?

Can any of these approaches express all the definitions and arguments we want to use?

- ▶ Concrete approaches surely can, but at expense of very detailed reasoning.
- ▶ Can more convenient approaches do the job?

How can we know if an approach is adequate before investing significant effort in using it?

Needed: A challenge problem set for reasoning about binding

- ▶ In order to test technical approaches to reasoning about binding, we suggest developing a challenge problem set.
- ▶ When we come across a problem that doesn't work well in existing techniques, we add it to the problem set.
- ▶ The challenge problem set should be unified and simple enough that it takes at most a few days to develop a solution using a good approach.

Do you have problems to suggest, that give difficulty in one of the established techniques?

Some Naturally Occurring Problems

Simply typed lambda terms

- ▶ Let A, B, \dots be *simple types* (implicational propositions).
- ▶ x, y, z, \dots are variables.
- ▶ Define terms (a, b, c, \dots):

$$a ::= x \mid \lambda x. b \mid (ab).$$

- ▶ *Valid contexts* (Γ, Δ) are lists of uniquely labelled assumptions:
 - ▶ $x_1:A_1, \dots, x_n:A_n$ where the x_i are pairwise disjoint.

Are the following judgements equivalent?

- ▶ \vdash has many derivations per derivable judgement.

$$\text{ass} \quad \Gamma \vdash x : A \qquad \Gamma \text{ valid, } x:A \in \Gamma$$

$$\text{elim} \quad \frac{\Gamma \vdash c : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash ca : B}$$

$$\text{intro} \quad \frac{\Gamma, x:A \vdash [x/y]b : B}{\Gamma \vdash \lambda y.b : A \rightarrow B} \quad x \notin b$$

- ▶ \Vdash has at most one derivation per derivable judgement.

$$\text{ass} \quad \Gamma \Vdash x : A \qquad \Gamma \text{ valid, } x:A \in \Gamma$$

$$\text{elim} \quad \frac{\Gamma \Vdash c : A \rightarrow B \quad \Gamma \Vdash a : A}{\Gamma \Vdash ca : B}$$

$$\text{intro} \quad \frac{\forall x. x \notin \Gamma \implies \Gamma, x:A \Vdash [x/y]b : B}{\Gamma \Vdash \lambda y.b : A \rightarrow B}$$

Proof?

Lemma $\Gamma \Vdash a : A \implies \Gamma \vdash a : A$.

- ▶ Proof direct by induction on the derivation of $\Gamma \Vdash a : A$.

Lemma $\Gamma \vdash a : A \implies \Gamma \Vdash a : A$.

- ▶ Attempt proof by induction on the derivation of $\Gamma \vdash a : A$.
 - ▶ Consider the case of rule *intro*.
 - ▶ Any derivation of \vdash will use a particular variable, say x_0 .
 - ▶ The IH for this case is

$$\Gamma, x_0:A_0 \Vdash [x_0/y]b : B \quad (x_0 \notin b) \tag{1}$$

but to use the *intro* rule for \Vdash we need the premise

$$\forall x. x \notin \Gamma \implies \Gamma, x:A_0 \Vdash [x/y]b : B$$

- ▶ We cannot reason from a particular variable to all variables!

Proof continued: $\Gamma \vdash a : A \implies \Gamma \Vdash a : A$

This proof by Michael Norrish,
inspired by *nominal techniques* of Gabbay/Pitts/Urban.

- ▶ Let $(xy) \cdot b$ mean *permute all occurrences of x and y in term b* .
- ▶ As a lemma, show

$$\Gamma \Vdash a : A \implies \forall x y. (xy) \cdot \Gamma \Vdash (xy) \cdot a : A. \quad (2)$$

- ▶ Now, pick $x \notin \Gamma$. From (1) and (2) have

$$(xx_0) \cdot (\Gamma, x_0 : A_0) \Vdash (xx_0) \cdot ([x_0/y]b) : B$$

i.e. $\Gamma, x : A_0 \Vdash [x/y]b : B$ as required.

We have a nice proof; what's the problem?

- ▶ We have to prove this property for every language, and every judgement on that language.
 - ▶ For some judgements, it isn't as easy as this example.
- ▶ But the equivalence is a meta-fact; it doesn't really depend on the particular judgement.
- ▶ Thus, the meta-language should handle this equivalence once and for all.
- ▶ Really want to write rule *intro* with a freshness quantifier:

$$\frac{\text{fresh } x. \Gamma, x:A \Vdash [x/y]b : B}{\Gamma \Vdash \lambda y. b : A \rightarrow B}$$

For example, Gabbay/Pitts/Urban, or Miller/Tiu.

A PER Semantics for Categorical Typing

Simplified from Coquand/Pollack/Takeyama, TLCA'03

- ▶ Have pure λ -terms as above a, b, M, N, \dots
- ▶ Have a syntax of dependent types

$$A, B ::= EIM \mid \text{fun } A x.B \mid \star$$

- ▶ Objects in \star are “names” of types;
 - ▶ for $M : \star$, EIM is the type named by M .
- ▶ Simultaneously define:
 1. Type equality: a PER on syntactic types
 - ▶ Write $A = B$.
 - ▶ Write $A \in \mathbf{Type}$ for $A = A$.
 2. Interpretation of a type: for $A \in \mathbf{Type}$, a PER, \bar{A} , on objects
 - ▶ Write $M = N : \bar{A}$.
 - ▶ Write $M : \bar{A}$ for $M = M : \bar{A}$.

Informal definition of semantics

▶ Parameterised on base cases:

- ▶ a given PER \bar{x} , and
- ▶ a given family \mathcal{E} over \bar{x} .
- ▶ These must have a property *saturated*.

▶ Three cases in the definition:

1. $\star = \star$ where \bar{x} is the given PER.

2. $\frac{M = N : \bar{x}}{EI M = EI N}$ where $\overline{EI M}$ is $\mathcal{E}(M)$.

3. $\frac{A_1 = A_2 \quad M_1 = M_2 : \overline{A_1} \implies B_1[M_1] = B_2[M_2]}{\text{fun } A_1 \ x_1. B_1 = \text{fun } A_2 \ x_2. B_2}$

where $F_1 = F_2 : \overline{\text{fun } A \ x. B}$ iff

$M_1 = M_2 : \overline{A} \implies F_1 M_1 = F_2 M_2 : \overline{B[M_1]}$.

How to formalize this definition?

Problems:

- ▶ induction-recursion not available in current proof tools
- ▶ occurrence of $B[N_1]$ is not a substructure of $\text{fun } A \text{ x. } B$ in:

$$F_1 = F_2 : \overline{\text{fun } A \text{ x. } B} \text{ iff } N_1 = N_2 : \overline{A} \implies F_1 N_1 = F_2 N_2 : \overline{B[N_1]}.$$

Solution: Unwind induction-recursion into

- ▶ first, a recursive definition of \overline{A} ...
- ▶ ... parameterized by a simultaneous substitution to make the definition structural;
 - ▶ We could use well-founded recursion to define \overline{A} ,
 - ▶ but that is *much* harder to reason about in intensional type theory.
- ▶ second, an inductive definition of $A = B$.

Defining the interpretation of a type

Binary relation \bar{A} defined by recursion on structure of syntactic type A .

- ▶ Not yet a PER, but we later show that if $A = A$, then \bar{A} is a PER.
- ▶ In order to make this definition structural, parameterise it by a substitution.

$$\begin{aligned} \overline{\star, \sigma} &:= \star \\ \overline{EI M, \sigma} &:= \mathcal{E}(M\sigma) \\ \overline{\text{fun } A \text{ x.B}, \sigma} &:= F_1, F_2 \mapsto \\ & N_1 = N_2 : \overline{A\sigma} \implies F_1 N_1 = F_2 N_2 : \overline{B, (\sigma, x=N_1)} \\ \bar{A} &:= \overline{A, \sigma_{\text{id}}} \end{aligned}$$

Our formalization of binding must support simultaneous substitution.

The relation of being equal (correct) types

A relation, defined by induction.

$$\frac{}{\star = \star} \qquad \frac{M = N : \star}{EI M = EI N}$$

$$\frac{A_1 = A_2 \quad M_1 = M_2 : \overline{A_1} \implies B_1[M_1] = B_2[M_2]}{\text{fun } A_1 \ x_1. B_1 = \text{fun } A_2 \ x_2. B_2}$$

Lemma If $A = B$ then

- ▶ $\overline{A} = \overline{B}$ (extensionally)
- ▶ \overline{A} is a PER on objects
- ▶ $- = -$ is a PER on syntactic types

Infinite structures

- ▶ This example shows we need to formalise simultaneous n-ary substitution.
 - ▶ Can HOAS handle that?
- ▶ In fact, we use infinite *environments* of meta-type $id \rightarrow term$.
 - ▶ No finite support: can freshness logics handle that?
- ▶ Another example of infinite structures is Böhm trees (suggested by Barendregt).

Approaches to Formalizing Binding

First order representations

Clumsy, but can handle any informal structure and argument.

- ▶ Naive first order representation (Curry and Feys)
 - ▶ quotient by α -equivalence: feasible in an extensional logic?
 - ▶ substitution not structural
- ▶ Stoughton: named variables, simultaneous substitution.
 - ▶ simultaneous substitution is structural
- ▶ De Bruijn representation: nameless variables.
 - ▶ hard to work with, but there are developed libraries of lemmas
 - ▶ cannot represent languages that are not α -closed
- ▶ McKinna/Pollack: distinct classes of free and bound names.
 - ▶ long-winded, but concrete and structural
 - ▶ variations possible: e.g. locally bound variables are nameless

Unitary Substitution Not Structural

- ▶ $[-/-]b$ is defined by recursion on *length* of b ,
 - ▶ not on *structure* of b , since $[z/y]b$ is not a subterm of $\lambda y.b$.

$$[c/x]y \quad := \quad \text{if } x = y \text{ then } c \text{ else } y$$

$$[c/x](b_1 b_2) \quad := \quad ([c/x]b_1) ([c/x]b_2)$$

$$[c/x](\lambda y.b) \quad := \quad \lambda z.[c/x][z/y]b \quad \quad z \text{ sufficiently fresh}$$

Simultaneous Substitution is Structural

$$x\rho \quad := \quad \rho x$$

$$(a b)\rho \quad := \quad (a\rho) (b\rho)$$

$$(\lambda x.b)\rho \quad := \quad \lambda z.(b(\rho, x=z)) \quad \quad z \text{ sufficiently fresh}$$

- ▶ The choice of fresh z can be canonical;
 - ▶ then applying any substitution alpha-normalizes (Stoughton).

A *Locally Nameless* Representation of Objects

- ▶ Two classes of variables:
 - ▶ Parameters (“free” variables): an infinite type of names with decidable equality.
 - ▶ de Bruijn indexes (for bound variables) are natural numbers.

```

Inductive obj : Set :=
| vp : par -> obj          (* free variables *)
| vv : nat -> obj         (* locally bound *)
| vlam : obj -> obj
| vapp : obj -> obj -> obj.

```

- ▶ Syntactically correct terms are those with no free indexes: an inductively defined property.

Syntactically correct locally nameless terms

An inductively defined predicate.

```

Inductive vclosed : obj -> Prop :=
| vc_vp : forall (p:par), vclosed (vp p)
| vc_vapp : forall (u v:obj)
            (pr1: vclosed u) (pr2: vclosed v),
            (*****)
            vclosed (vapp u v)
| vc_vlam : forall (u:obj) (p:par)
            (pr1: vclosed (vsub (vp p) u)),
            (*****)
            vclosed (vlam u).
  
```

Instead of inducting over term structure, we usually induct over a derivation that the term is `vclosed`.

Why This Representation?

- ▶ Datatype of terms, equal up to α -equivalence.
- ▶ Substitution is simple and structural.
 - ▶ Since correct terms are de Bruijn closed, substitution does *not* require lifting of free indexes.
 - ▶ Since parameters are not bound, substitution can never capture parameters.
- ▶ Concrete treatment of free variables.
 - ▶ Supports explicit contexts, simultaneous substitution, environments
 - ▶ The “swapping” proof of my first example applies here.
- ▶ This seems the best concrete representation . . .
- ▶ . . . but there is no fresh quantifier, or uniform treatment of eigenvariables.

Locally Nameless++ (Andy Gordon)

- ▶ Built on top of locally nameless representation.
- ▶ A derived operation of named abstraction.
 - ▶ $\lambda x.t$ defined to be “de Bruijn abstraction of t after replacing every occurrence of x with index 0”.
- ▶ Then define an extensional subtype of proper terms (`vclosed`).
 - ▶ Only possible in extensional logic.
- ▶ Implementation: Gordon/Melham axioms for alpha-conversion.
 - ▶ Extensive examples by Michael Norrish.
- ▶ Also see recent note by Wilfried Buchholz on this approach.

In this approach, globally bound variables must still be reasoned about explicitly.

Logics with “freshness”

- ▶ Pitts/Gabbay FM based approaches.
 - ▶ incompatible with AC
 - ▶ working implementation? (Urban?)
- ▶ Fiore/Plotkin/Turi abstract syntax.
 - ▶ very abstract; not implemented
- ▶ New work by Ian Stark and Ulrich Shöpp (CSL '04).
 - ▶ Gabbay/Pitts idea for dependent types
 - ▶ not yet developed, but very interesting!
- ▶ $FO\lambda^\nabla$ of Miller/Tiu (TOCL, to appear).

These are the approaches that I believe are most promising.

Higher Order Abstract Syntax (HOAS)

- ▶ Binding has meta-type $exp \rightarrow exp$.
- ▶ Expressions are not inductively defined: positivity requirements.
- ▶ Can any HOAS approach handle simultaneous substitution?

Some examples:

- ▶ Pure logical framework (Edinburgh LF).
 - ▶ No induction/recursion over object structure.
- ▶ Logical frameworks with modalities, schema checking,
 - ▶ Twelf (Pfenning/Schürmann)
 - ▶ Some induction/recursion over object structure.
- ▶ Less pure LF, such as Isabelle.
 - ▶ No induction/recursion over object structure.
- ▶ *Hybrid* (Ambler/Crole/Momigliano)
- ▶ Stratified (several level) LF, e.g. Miller/McDowell, Felty.

Weak HOAS

- ▶ Datatype of expressions $e ::= var \mid e@e \mid lam(var \rightarrow e)$.
- ▶ This type is not faithful because of “exotic” terms:

$$lam(\lambda v. \text{if } v = v_0 \text{ then } 2 \text{ else } 3)$$

- ▶ Exotic terms must be removed by some logical means.

Some examples:

- ▶ Honsell/Miculan context calculus.
 - ▶ axiomatic
- ▶ Desperoux/Felty/Hirschowitz.
 - ▶ validity predicate

These approaches get very complicated in practice.

Also, complicated justification.