Is there a quick way to get names and binding in some nominal sense up and running in Isabelle/HOL polymorphically on the type-class ‘type’, without having to rewrite the whole thing from scratch — yes or no?

Murdoch J. Gabbay

JAIST, Ishikawa, 4/2005

Introduction

Many thanks to Rene ‘pre*cise*ly’ ‘mountain on a mountain’ ‘gadget king’ Vestergaard, and Randy Pollack, for arranging this party.

Thanks also to Randy Pollack for putting his slides online in advance. We should all do that.

Answer:

Theorem 1. Yes.

Proof. We polymorphically exhibit a type $[\alpha]\beta$ with constructor

$$[-]- \ : \ \alpha \to (\beta \to [\alpha]\beta)$$

(like pairs) and destructor

$$\circ \ : \ ([\alpha]\beta) \to \alpha \to \beta$$

(like functions).

— we give a function *in* Higher-Order Logic

$$[-]- \ : \ \alpha \to (\beta \to (\alpha \to \beta))$$

such that $([x]y)x = y$ is provable and $[-]-$ is a binary term-former.

Historical overview

Obviously I can't do it like Randy can but let me do my best with what experience I have:

We (Gabbay and Pitts 1999-2001) introduced a semantic model of abstraction $[\mathbb{A}]X$ based on Fraenkel-Mostowski sets.

This was exciting because it was *polymorphic* — abstraction was a functor — and $[\mathbb{A}]X$ is always the same size as X.

Of course there was more, e.g. $И$ quantifier, $[\mathbb{A}]X$ preserved LUBs of countably ascending chains. We could talk about inductively-defined datatypes of syntax up to binding.

But we can do that anyway: what was new was that it was polymorphic *and* not too big.

Historical overview

Dream: use $[\mathbb{A}]X$ to do binding once and for all.

Note that $[\mathbb{A}]X$ could be applied to function-spaces, i.e. all of computer science as I understood it at the time, so 'once and for all' really does mean that.

Did you know that $[\mathbb{A}](Y^X)$ is isomorphic to $([\mathbb{A}]Y)^{[\mathbb{A}]X}$? Andrew refused to believe me when I first noticed it. *That's* how good the theory was.

Something went wrong

Nominal Sets (Frankel-Mostowski sets combined with types) is inconsistent with the axiom of choice.

$\epsilon x : \mathbb{A}.\top$ chooses *some* element of $\mathbb{A}$. It is not possible to cross FM with Isabelle/HOL. Not without reengineering lots of code.

I'd already done that kind of thing once turning Isabelle/ZF into Isabelle/ZFA. I wasn't going to do it again. Not all alone, unpaid, unloved, unsupported, etc.

So:

Christian Urban . . .

. . . said ‘OK, forget polymorphism’.

Instead of ‘Fresh Isabelle’, we build Fresh *in* Isabelle.

Weak in theory, in practice viable.

‘Weak’ because you pay a ‘tax’ proving for your every function that it has finite support, i.e. that it does not contain too much of ϵ. ‘Viable’ because the tax is a flat percentage of your effort (if you’re lucky) and not prohibitive; especially I think on an inductive set so you have a nice proof-principle.

Jamie Gabbay. . .

. . . has been thinking.

Can we have our cake and eat it.

Polymorphism and all the other nice properties of $\mathbb{A}$ and $[\mathbb{A}]X$, *and* choice ϵ, *and* build it all in Isabelle/HOL (any one of these three requirements is already non-trivial).

Let's do philosophy

Frankel-Mostowksi/Nominal/Fresh techniques give you:

- Finite support ($\mathcal{M}$).
- Variables inhabit a distinct type/category of elements ($\mathbb{A}$).

These clearly come from the underlying semantic model.

Classical mathematics give you:

- Choice ϵ.
- Function-spaces Y^X.

The two are incompatible. What should we throw out? *Well, what do we want?* To reason on syntax-with-binding. The semantic model is a means to that end. So get rid of it.

Throw out the semantic model.

Kill FM. What? You can’t do that Jamie!

No, wait. I can explain.

Table of methods

- Atoms do inhabit a separate type. — Fresh method
- 'There is always a fresh atom' is explicit.
- Abstraction is a binary term-former.
- No ϵ in typed theory.

- Atoms do not inhabit a separate type. — Functional method
- 'There is always a fresh atom' is not explicit.
- Abstraction (λ) is not a binary term-former.
- ϵ is in typed theory.

- Atoms do not inhabit a separate type. — I propose: a-method
- 'There is always a fresh atom' is explicit.
- Abstraction is a binary term-former. $\leftarrow$ Nominal!
- ϵ is in typed theory.

Apparent contradictions:

- Atoms do not inhabit a separate type.
- 'There is always a fresh atom' is explicit.
- Abstraction is a binary term-former. $\leftarrow$ Nominal!
- ϵ is in typed theory.

If abstraction is a binary term-former and atoms do not inhabit a separate type, what is 'abstract 2 in in 17'?

If 'there is always a fresh atom' is explicit, then 'is an atom' must be explicit. But then we can use collection to separate out the atoms, contradicting 'atoms do not inhabit a separate type'.

We can also choose 'THE atom of such-and-such a type', which is what led to the problems with FM and HOL.

The basic technical kernel of my suggestion, in the context of Isabelle/HOL, is:

- Introduce a predicate $atom$ of type $(\alpha :: \texttt{term}) \rightarrow Prop$.
- Introduce axioms of the form $atom(t) == \texttt{false}$ for all t which are *not* actually variable symbols.

E.g. $atom(0) == \texttt{false}$ and $atom(succ(x)) == \texttt{false}$. I recently did a first-order logic with $atom$; the conditions become a derivation rule

$$\Gamma, \mathbf{at}\, t \vdash \Delta$$

if t is not a variable symbol. I called it a-logic: has cut-elimination and a sound and complete semantics.

Proof of Theorem 1. Using $atom$, see overleaf.

ϵ is there but its type is $((\alpha : \texttt{term}) \rightarrow o) \rightarrow \alpha$ stops us choosing arbitrary atoms.

a-logic

```
theory alogic = HOL + Datatype_Universe:

subsection {* Primitive logic *}
global
typedecl
 ('a,'b) abstrtype
arities
 abstrtype :: (type,type) type
consts
 atom     :: "'a ::logic => prop"
 abstr    :: "['a :: type, 'b ::type] => ('a,'b)abstrtype"
 subst    :: "[('a,'b)abstrtype, 'a] => 'b"
 freshfor :: "['a :: type, 'b :: type] => bool"
```

α-logic: polymorphic explicit substitution

```
subsubsection {* Axioms and basic definitions *}
defs
 freshfor_def:
  "(freshfor x y) == ALL z. (subst (abstr x y) z) = y"
axioms
 fresh:
  "!!y. (!!x.(freshfor x y) ==> atom x == true ==> P)
                                                 ==> P"
 sub_a:
  "(atom x == true) ==> ((subst (abstr x x) z) = z)"
 sub_app:
  "subst (abstr x (f y)) z =
           ((subst (abstr x f) z) (subst (abstr x y) z))"
 abstr_alpha:
  "freshfor x' y ==>
             abstr x y = abstr x' (subst (abstr x y) x')"
 abstr_surj:
  "(!!x y. atom x == true ==> z = abstr x y ==> P z)
                                               ==> P z"
```

α-logic: a type of λ-calculus

```
typedecl lambda
consts Lam :: "(lambda,lambda)abstrtype => lambda"
       App :: "lambda => lambda => lambda"
axioms
 lambda_induct:
  "(!!x y. P y ==> P (Lam (abstr x y))) ==>
   (!!x y. P x ==> P y ==> P (App x y)) ==>
   (!!x. atom x == true ==> P x)
      ==> P z"
```

This could either be an inductive datatype (given axioms such as `Lam x = App y z ==> false`), or a model ...

Reductions

...depending on whether you use `reduce` or `=`.

```
consts reduce ::
 "lambda => lambda => bool"
axioms reduce_beta:
 "atom x == true ==>
  reduce (App (Lam (abstr x y)) z) (subst (abstr x y) z)"
reduce_cong_App:
 "reduce y y' ==> reduce (App x y) (App x y')"
reduce_cong_Lam:
 "atom x == true ==> reduce y y' ==>
             reduce (Lam (abstr x y)) (Lam (abstr x y'))"
```

Comments

I have not used the inductive datatypes package; we could, but we'd have to convince it that `abstr` has nice properties. No time.

We cannot write `EX x. atom a`, only `(!!x. atom x ==> Q) ==> Q`. Not too bad. Consequence of the type of `atom`.

But: anybody can load this theory and get proving.

Maybe even ϵ could be taken out of Isabelle/HOL where it is not really needed, but now incrementally and transparently.

. . . and back to nominal techniques

You get from FM to α-logic by associating to every type a set of atoms which injects into it and interprets meta-level variable symbols (so for example there are axioms for substitution).

(In my paper I give an elementary model based on valuations.)

Further comments

This seems promising, but I'm not guaranteeing results! I've only just thought of this so I have not had time to do the research.

α-logic is a bit like a 'Nominal Sets with substitutions everywhere', if you want to think of it that way (and you don't have to if you don't want to). The slogan is 'variables and substitutions are fundamental'.

It's fully incremental, in the sense that you can have it as an Isabelle/HOL theory, *and* presumably reuse nominal code because of commonalities of form (e.g. `abstr`). Indeed, it's just as if you get substitution for free and then just carry on as usual.

It has taxes but so do they all. Some taxes could be handled 'once and for all' later, if deemed worthwhile.

Further comments

By the way, a-logic is more generally applicable than I have shown here. For example, do you want context substitution?

Introduce $< :: (\alpha :: type) \rightarrow \alpha \rightarrow prop$ with axioms such as

$$atom(x) \wedge atom(y) \wedge atom(z) \wedge x<y \wedge y<z \Rightarrow x<z$$

and axioms for explicit substitution such as

$$y<y' \Rightarrow x[y \mapsto z][y' \mapsto z'] = x[y' \mapsto z'][y \mapsto z[y' \mapsto z']]$$

for possibly-capturing substitution provided y' is 'stronger' than y.

Thus we can axiomatise any flavour of substitution we prefer, e.g. context substitution.

All polymorphic in Isabelle/HOL.