Names and variables

Murdoch J. Gabbay, Heriot-Watt University, Scotland

City University, London
Tuesday 27 February 2007

Thanks to Artur Garcez

Names

Names are very interesting. They turn up in syntax, for example in logical predicates such as $\forall x.x = x$, and in programming languages in terms such as $\lambda x.xx$.

I began my research career by (re-)discovering Fraenkel-Mostowski set theory and observing that it provides a semantic model of α-equivalence.

Names

In Fraenkel-Mostowski I and Andrew Pitts were able to represent inductive datatypes of parse trees with binding, and extract some really nice reasoning principles on them.

See equivariance and the Gabbay-Pitts И (‘NEW’) quantifier.

For example $\text{И}a.\phi$ means ‘ϕ holds of some fresh a’. We don’t have to specify what a is fresh for, and that saves mathematicians from maintaining a context of ‘known names’ in the theory.

Equivariance says that $\phi(x)$ if and only if $\phi((a\ b)x)$ where $(a\ b)x$ is a and b swapped bijectively in x. This is useful for α-renaming bound names without losing inductive hypotheses on subtrees of parse trees.

Uses of Fraenkel-Mostowski techniques

It’s been very successful:

α-prolog, FreshML, unification (with these names), rewriting (with these names), nominal logic, one-and-a-halfth-order logic, nominal algebra, capture-avoiding substitution as a nominal algebra, the NEW calculus of contexts, hierarchical nominal rewriting, FreshLIB, Scrap Your Nameplate, a nominal package in Isabelle/HOL, . . .

(not all of this list was done by me!)

. . . and almost certainly more I don’t know about.

Uses of Fraenkel-Mostowski techniques

But almost all of the research above has been devoted just to studying how to manipulate names in abstract syntax.

The underlying denotational model is always parse trees!

But there are many other uses of names.

Names in denotation

Names as variable symbols. IMO a variable symbol is a name with a substitution action.

Names as pointers; a pointer is like a variable symbol, but it's also a datum in the language.

Names in linguistics. In the sentence 'A dog walks. A man sees it.' the sentence-fragment 'A dog' can be understood as an existentially quantified variable, but one which can be accessed as a datum and linked to by later sentences.

Names in denotation

Names as existential variables. In the intro-rule for the existential quantifier

$$\frac{\Gamma \vdash P[x \mapsto t]}{\Gamma \vdash \exists x.P}$$

the t comes ‘out of nowhere’; the rules requires we guess it. This is obviously a computational nightmare. In fact, the t is an existential variable.

Names as wires. A name can be used as a wire to express linkage between different parts of a program. For example ports, IP addresses, exceptions, or ... just wires.

Denotational models

I’m now working to develop a variety of denotational structures in which names are first-class entities.

So a name is not just something in the syntax — interesting as the behaviour of these names is, as outlined at the beginning of this talk.

I propose: a name is an actual mathematical entity. Different names, have different mathematical properties.

Why?

Why?

Why!

Obviously, because it’s fun.

But also, if we get good denotational models of names then we maybe we can use them to build new derivation systems and programming languages. By carefully tuning the properties of this denotational model, we may be able to derive weaker systems than are currently available.

Weaker system = better computational properties.

Why!

Currently there are two options: either a name has no substition action, or we use full-fledged variables with λ-abstraction. That brings in function spaces, which are large and computationally intractable.

So for a start, I would like to build classes of denotations of variables with weak substitution actions; capturing some of the power of function application, but not too much.

This has already been done using syntactic techniques guided by algorithmics of calculations such as unification on existing syntax; higher-order patterns are an example. I will be guided by mathematical denotation and (probably) develop new syntaxes.

Why!

Also, current functional techniques only permit capture-avoiding substitution.

But in practice other kinds of substitution are useful. For example when we write

‘let ϕ be $x = x$ in $\forall x.\phi$’

we expect to get

‘the answer is $\forall x.x = x$’.

Here x in $x = x$ has been captured by $\forall x$. This is relevant e.g. in representing incomplete proofs.

Why!

I am very interested in existential variables and am trying to build a logic with two classes of variables; one for the universally quantified variables, one for the existentially quantified variables.

Machine learning?

Higher-order generalisation is the problem: given t and u find a most specific v such that there exist terms t' and u' such that $vt' = t$ and $vu' = u$. In that sense, v generalises t and u.

The problem is that functions are a little bit too powerful for this notion to be as useful as it could be.

It would be nice if we could have some notion of substitution on names — but without the full computational power of the λ-calculus.

Conclusions

I am developing a toolbox of techniques for manipulating names and variable symbols and fine-tuning their properties in new and very interesting ways.

The basic tool is an idea: treat names as denotational entities. Then you can impose just the right set of axioms to get the behaviour you want.

The innovation is to apply the ‘nominal’ treatment of names — originally developed to specify and manipulate names in parse trees representing abstract syntax — to structures that are not parse trees; e.g. functions, sets, and so on.