Names: I denote, therefore I am

Murdoch J. Gabbay,   www.gabbay.org.uk

Wednesday 11 November 2009

Thanks to Phil Trinder

Names

This talk will be mostly questions.

Interesting questions.

Answers? In time, perhaps.

Names

By name I mean a term in a language whose function is to denote.

Here are some example names:

- Universal variable x, as in $\forall x.(x = x)$.
- Pointers l and l', as in $!l = !l'$.
- Existential variable $?y$, as in $\forall x.\exists y.(x = y) \longrightarrow x = ?y$.
- Independence-friendly logic y/x, as in $\forall x.\exists y/x.(x = y)$.
- Variable symbol x, as in '$\lambda x.t$'.
- Meta-variable t, as in '$\lambda x.t$'.
- Natural language: "the man who walks in the park", "the man, whom I saw", "the king of France".
- Choice $\epsilon x.(x = 2)$.
- … and so on.

Names' behaviour

Names have nontrivial operational and logical behaviour.

For example, variable symbols and meta-variables are bound up with each other in specifications of logics and programming languages.

β-equivalence and $\forall$-introduction (variable symbols and meta-variables):

$$(\lambda x.r)t = r[x := t] \qquad \frac{\Gamma \vdash \phi \quad (x \notin \mathsf{fv}(\Gamma))}{\Gamma \vdash \forall x.\phi}$$

Existential variables have operational significance in proof-search:

$$\Xi,\ \exists x.\phi \quad \Longrightarrow \quad \Xi,\ \exists x.\phi,\ \phi[x := ?x]$$

What does this mean?

Questions

What are we doing when we “specify the λ-calculus” or “axiomatise Higher-Order Logic”?

What is a meta-variable? . . . an incomplete proof? . . . what are anaphora? . . . what is $\epsilon x.(x = 2)$?

What does it really mean for a variable to ‘depend on another variable’, or ‘not to depend on another variable’?

Do not confuse familiarity with understanding.

These are interesting questions.

A distinction

What does it mean to denote?

When we write 'x', we may think of this as denoting an arbitrary element of some domain. We may introduce a valuation ς and give semantics $[\![\text{-}]\!]_\varsigma$ to terms using it. $[\![x]\!]_\varsigma = \varsigma(x)$.

Fine distinction: This tells us what x denotes, but it does not tell us what x is, or what 'to denote' is.

In order to say 'A denotes B', we should ask what A, B, and 'to denote' are.

What is our theory of denotation, and if we do not identify the denotation of a name with the name itself, then what are these 'name' objects that are doing the denoting?

Practical importance of names

This question can become practially important in a number of ways.

Suppose we are manipulating the syntax of a logic or programming language. Then we need to manipulate the variable symbols in that syntax. These variable symbols are not innocuous; they can be α-renamed. Famously, it is difficult to manage this inside traditional programming languages. This is because they were designed without facilities for a datatype of variable symbols.

Suppose we are interested in meta-programming. Then again, we need to manipulate program syntax, but this syntax can also be executed. So we are manipulating ‘open code fragments’.

(There’s more. . .)

Names

Suppose we are constructing a theorem-prover. Then we may need to be concerned with how to axiomatise systems such as first-order logic and higher-order logic. The standard way to do this is using simple types. $\forall$ is a constant of higher type $(\iota \rightarrow o) \rightarrow o$. A problem with this is that higher types are large and complex. Higher-order unification, for example, is undecidable.

Some names may simply have no obvious denotation: 'The king of France'.

Some names may have an explicitly intensional content. Reasoning on pointers.

And so on.

I denote, therefore I am not

The usual slogan is: “I denote, therefore I am not.”

If a name denotes something, then we can throw away the name and just keep the denotation. This is reasonable, but insufficient for many purposes.

What is a name? What is it to denote?

I denote, therefore I am

Partial answer:

Nominal sets provide a semantics for names and for objects (sets, functions) containing names.

This model was originally designed for variable symbols; the names were atomic, but could be α-abstracted.

I propose to enrich this model with extra structure.

E.g. substitution action to model variables ('to denote' = 'to be substituted for').

E.g. dependency is modelled by generalising the notion of name, so that names can contain other names.

...and so on.

Applications

Foundations (new foundational logics).

Theorem-provers (implementations of foundational logics / semantics of proof-search).

Meta-programming (programs that build programs).

Correctness proofs (operational techniques).

Formal logic (new logics, new semantics for existing logics).

Linguistics (anaphora, definite description).

Efficient implementation (avoid skolemisation, so avoid beta-redexes and higher-order unification).