

Logic of names and freshness

(Splitting the atom)

Murdoch J. Gabbay, Heriot-Watt University, Scotland

DSG, Heriot-Watt University, Scotland

27 October 2006

Thanks to Gudmund Grov

In this talk. . .

. . . let's take names in the λ -calculus seriously.

The λ -calculus

Syntax of terms is generated by:

$$e ::= x \mid ee \mid \lambda x.e.$$

Identified up to α -equivalence.

Meta-variables

Actually, the **practical** theory of the λ -calculus uses **meta-variables**.

A meta-variable is a variable (ranging over λ -terms) which is **instantiated** — substituted in a possibly capturing way. Usually considered a variable in the meta-language used to describe λ -terms, whence the ‘meta-’.

e in the last slide was a meta-variable. It can be instantiated: if we say

- ‘take e to be x in $\lambda x.e$ ’, then we get
- ‘ $\lambda x.x$ ’ and not ‘ $\lambda x'.x$ ’.

Internalise

Let's add that to our syntax:

$$e ::= x \mid ee \mid \lambda x.e \mid X.$$

Here X is a variable symbol like x .

However X is **syntax** enriching the language of terms to represent the ' e ' in ' $\lambda x.e$ '.

(Now e refers to an unknown term in the enriched syntax.)

Two kinds of substitution

Substitution $[X \mapsto x]$ is capturing:

$$(\lambda x.X)[X \mapsto x] = \lambda x.x.$$

Substitution $[x' \mapsto x]$ avoids capture:

$$(\lambda x.x')[x' \mapsto x] = \lambda x''.x.$$

α -equivalence

We lose α -equivalence.

$\lambda x.X$ is **not** α -equivalent to $\lambda y.X$.

If we identify $\lambda x.X$ and $\lambda y.X$ then we should identify their instances $\lambda x.x$ and $\lambda y.x$.

This would be bad.

α -equivalence

The natural notion of occurrence of one variable in another is identity.

- x does not occur in y because x is not identical to y .
- x does not occur in X because x is not identical to X .

The usual definition of α -equivalence implies that

$$\lambda x.X =_{\alpha} \lambda y.X \text{ if } x \text{ and } y \text{ do not occur in } e.$$

Well, x and y do **not** occur in X ; manifestly not, X is syntax.

So the first statement is appropriate, the second is not.

We lose locality of λ -abstracted variables

An interesting corollary of all this is:

We cannot consider x to be bound in $\lambda x.X$.

You can rename bound variables, and we just argued that we cannot rename x in $\lambda x.X$.

So x is **not** bound in $\lambda x.X$.

Internalise

Hang on. The **whole point** of the λ -calculus is to λ -abstract variables. So really we want this:

$$e ::= x \mid ee \mid \lambda x.e \mid X \mid \lambda X.e.$$

But now we're back where we're started, with variables x and X and meta-variables e .

I say that 2 equals ∞ . So let's write

$$e ::= a_i \mid ee \mid \lambda a_i.e.$$

Here a_i, b_j, c_k have levels i, j, k .

Two kinds of substitution

Suppose $i < j$. Substitution

$$(\lambda a_i . e)[b_j \mapsto f] = \lambda a_i . (e[b_j \mapsto f])$$

is capturing — b_j is like X and a_i is like x .

Substitution

$$(\lambda b_j . e)[a_i \mapsto f] = \text{?}.$$

We want ‘capture-avoiding’. But how do we avoid capture now that we have lost all notions of ‘ α -equivalence’ and ‘being bound in’? What is the appropriate generalisation of ‘ b_j not free in f ’?

Logic of freshness

The appropriate generalisation is **freshness** $a_i \# e$.

A **primitive freshness assertion** is $a_i \# b_j$ for $i < j$.

To a_i, b_j it looks like a meta-variable, which can become instantiated to something that might mention a_i .

$a_i \# b_j$ is a logical assertion 'this won't happen'.

Logic of freshness

$$P, Q, R ::= a_i \# b_j \mid \perp \mid P \Rightarrow P \quad (i < j)$$

Δ, Γ finite sets of predicates

$$\frac{}{\perp, \Delta \vdash \Gamma} (\perp \mathbf{L})$$

$$\frac{P, \Delta \vdash \Gamma, Q}{\Delta \vdash P \Rightarrow \Gamma, Q} (\Rightarrow \mathbf{R}) \quad \frac{\Delta \vdash \Gamma, P \quad Q, \Delta \vdash \Gamma, R}{P \Rightarrow Q, \Delta \vdash \Gamma, R} (\Rightarrow \mathbf{L})$$

This is just classical propositional logic with atoms $a_i \# b_j$ for $i < j$.
Write \top for $\perp \Rightarrow \perp$, $\neg P$ for $P \Rightarrow \perp$, and $P \wedge Q$ for $\neg(P \Rightarrow \neg Q)$.

Logic of freshness

Really we are interested in $a \# e$. Translate

$$[b_j \# a_i] = \top \quad (i \leq j) \qquad [a_i \# a_i] = \perp$$

$$[a_i \# b_j] = (a_i \# b_j) \qquad (i < j)$$

$$[a_i \# \lambda a_i. e'] = \top \qquad [b_j \# \lambda a_i. e'] = [b_j \# a_i] \Rightarrow [b_j \# e']$$

$$[a_i \# e'[a_i \mapsto e]] = [a_i \# e]$$

$$[b_j \# e'[a_i \mapsto e]] = [b_j \# e] \wedge ([b_j \# a_i] \Rightarrow [b_j \# e'])$$

$$[a_i \# e'e] = [a_i \# e'] \wedge [a_i \# e]$$

Logic of freshness

Identify x with a_1 , y with b_1 , X with a_2 , and Y with b_2 .

$\vdash x \# \lambda x. x$ means $\vdash x \# x \Rightarrow x \# x$. ✓

$\vdash x \# X$. ✗

$x \# X \vdash x \# X$. ✓

$\vdash x \# X[x \mapsto X]$. ✗

$x \# Y \vdash x \# X[x \mapsto Y]$. ✓

$x \# X \vdash x \# X[x \mapsto Y]$. ✗

Logic of freshness

How about this: $c_1 \# a_2 \Rightarrow c_1 \# b_3 \vdash a_2 \# b_3 \Rightarrow c_1 \# b_3$?

Here b_3 is stronger than a_2 is stronger than c_1 .

Axiom of freshness

$(c_1 \# a_2 \Rightarrow c_1 \# b_3) \Rightarrow (a_2 \# b_3 \Rightarrow c_1 \# b_3)$ can be expressed as a derivation rule by:

$$\frac{a_i \# b_j, c_k \# a_i, \Delta \vdash \Gamma, c_k \# b_j}{a_i \# b_j, \Delta \vdash \Gamma, c_k \# b_j} \quad (k \leq i < j)$$

This is not an admissible rule in classical propositional logic.

If we intuitively feel this should be **added** to our logic, this suggests the existence of some non-trivial **model**.

This model **must** mention names in the denotation. Names cannot be eliminated by a valuation, as is traditionally the case.

What is this model? I do not know, yet.

Binding

Read $x\#e$ as x is fresh for e .

Recall that ‘abstracted’ \neq ‘bound’. For example $x\#\lambda x.X$ but x is not bound in $\lambda x.X$.

Sometimes we **really do** want to abstract. Then we use \forall .

\forall is simple enough: $\forall x.\lambda x.e$ **really is** equal to $\forall y.\lambda y.e$. So x (and y) under a \forall does **not** occur in e (unless actually mentioned in e)!

Binding

- $\lambda x.e$ abstracts x in e but does not bind in the expression as a whole.
- $\forall x.e$ does not abstract in e but does bind in the expression as a whole.

In the λ -calculus, because there are no levels, binding can be identified with abstraction. $\lambda x.e$ is $\forall x.\lambda x.e$.

Binding and abstraction

Easy! $a\#\forall b.e$ is $a\#e$.

For example:

$$a_1\#\forall b_2.b_2 \equiv a_1\#b_2. \quad \times$$

$$a_1\#c_3 \vdash a_1\#\forall b_2.c_3 \equiv a_1\#c_3. \quad \checkmark$$

Do you disagree?

Do we want $a\#\forall b.e$ to be $b\#a \Rightarrow a\#e$? I don't think that makes any difference.

Do we want $a\#\forall b.e$ to be $a\#b \Rightarrow a\#e$, so $a\#\forall b.b$?

Substitution (without \mathcal{N})

Remember the question on slide 11?

$$\text{(suba)} \quad a_i[a_i \mapsto e] = e$$

$$\text{(sub\#)} \quad a_i \# e' \vdash e'[a_i \mapsto e] = e'$$

$$\text{(subaa)} \quad e'[a_i \mapsto a_i] = e'$$

$$\text{(subf)} \quad f(e'_1, \dots, e'_n)[a_i \mapsto e] = f(e'_1[a_i \mapsto e], \dots, e'_n[a_i \mapsto e])$$

$$\text{(subabs>)} \quad ([c_k]e')[a_i \mapsto e] = [c_k](e'[a_i \mapsto e]) \quad (i > k)$$

$$\text{(subabs}\leq) \quad b_j \# e \vdash ([b_j]e')[a_i \mapsto e] = [b_j](e'[a_i \mapsto e]) \quad (i \leq j)$$

f is any function-symbol in $\{\lambda, \text{app}, \text{sub}\}$. We'd like a model of this, please.

Why care?

Modularity can be difficult in the λ -calculus (exceptions; strict parameter-passing).

The λ -calculus cannot see the names of its own variables (x in e in $(\lambda x.e)e'$ identified with e' ; $x \neq y$ means the **value** of x is not equal to the **value** of y ; no intensional equality).

The λ -calculus is not good at expressing mobility (no way to express $C[t] \mid D[u] \rightarrow C[u] \mid D[t]$; no abstraction over contexts $\lambda x.\lambda y$).

I believe that by carefully developing the mathematics of names, we can split λ up (as seen here) **and then put it together again more flexibly!**

Conclusions

You can extend ‘flat’ variables with a hierarchy of variables. In this way meta-level reasoning is internalised.

New theorems are required to make this work. In other words, this process has mathematical content.

‘Binders’ split up into: abstraction, freshness $\#$, a binder λ , and substitution $[x \mapsto e]$.

Interesting denotational models seem to exist, which I do not yet fully understand. Many design choices.

Interesting programming paradigms; we have a kind of ‘functional theory of graphs’.

Potential for many papers!

α -equivalence

α -equivalence is the least congruence containing:

$$y \notin \text{fv}(e) \quad \Rightarrow \quad \lambda y.(y \ x)e = \lambda x.e.$$

Here

- $(y \ x)$ is a **swapping** which exchanges y and x in e (even under λ -binder), and
- $\text{fv}(e)$ is the free variables of e defined in the usual way.