Two-and-a-halfth-order lambda-calculus: a calculus of the informal meta-level.

Murdoch J. Gabbay, Heriot-Watt University, Scotland

DSG group meeting
Thursday 7 February 2008

Joint work with Dominic Mulligan

The λ-calculus

A simple and efficient syntax for talking about functions. Functions are useful: they turn up in higher-order logic, higher-order unification and rewriting, programming languages, theorem-provers, and lots more.

The λ-calculus is the de facto standard syntax for functions.

The informal meta-level

The informal meta-level (by this I intend ‘the discourse of a typical theory paper’) is full of capture-avoidance conditions and capturing substitution:

- λ-calculus: $(\lambda x.r)[y \mapsto t] = \lambda x.(r[y \mapsto t])$ x fresh for t
- π-calculus: $\nu x.(P \mid Q) = P \mid \nu x.Q$ x fresh for P
- First-order logic: $\forall x.(\phi \Rightarrow \psi) = \phi \Rightarrow \forall x.\psi$ x fresh for ϕ

The informal meta-level

Capture-avoidance conditions are on the right-hand side; they relate object-variables x, y to meta-variables r, t, P, Q, ϕ, ψ.

Capturing substitution ('instantiation') is when meta-variables are turned into terms, as in:

> "Set r to x and t to x in $(\lambda x.r)[y{\mapsto}t]$; obtain $(\lambda x.x)[y{\mapsto}x]$."

Conventional wisdom has it that these are just operations on syntax.

The informal meta-level, formalised

We argue that what happens at the informal meta-level reflects mathematical entities which — like functions — may be studied using a λ-calculus.

That means a λ-calculus with λ-abstraction over object-level variables (as usual) and meta-level variables (as unusual) and freshness conditions.

We also need α-equivalence.

This all turns out to be very interesting indeed.

Syntax of two-and-a-halfth-order λ-calculus

Fix sets $a, b, c, \ldots$ and $X, Y, Z, \ldots$ of level 1 and level 2 variables.

A permutation π is a finitely supported bijection of leve l 1 variables. 'Finitely supported' means $\pi(a) = a$ for all but finitely many level 1 variables.

Define syntax by:

$$r, s, t, u, v \;::=\; a \mid \pi \cdot X \mid \lambda a.r \mid \lambda X.r \mid rr$$

The part to do with a, $\lambda a.r$, and rr is the 'usual' λ-calculus.

Level 2 interacting with level 1

"Set t to be x in $\lambda x.t$" is modelled by the reduction

$$(\lambda X.(\lambda a.X))a \to (\lambda a.X)[X := a] \equiv \lambda a.a.$$

Here $\equiv$ is syntactic identity up to level 2 α-equivalence and $[X := a]$ is a level 2 substitution.

$[X := a]$ does not avoid capture by λa, modelling the behaviour of instantiation.

Within a single level everything is as usual:

$$(\lambda b.(\lambda a.b))a \to (\lambda a.b)[b \mapsto a] \to \lambda a'.(b[b \mapsto a]) \to \lambda a'.a$$

$$(\lambda Y.(\lambda X.Y))X \to (\lambda X.Y)[Y := X] \equiv \lambda X'.(Y[Y := X]) \equiv \lambda X'.X.$$

Level 1 α-equivalence

Write $=_\alpha$ for α-equivalence.

We do not want $\lambda a.X =_\alpha \lambda b.X$.

If this were so, then also

$$\lambda X.\lambda a.X =_\alpha \lambda X.\lambda b.X \quad \text{and} \quad (\lambda X.\lambda a.X)a =_\alpha (\lambda X.\lambda b.X)a$$

and therefore (reducing a bit)

$$\lambda a.a =_\alpha \lambda b.a.$$

Level 1 α-equivalence

So we use freshness $a\#X$ ('a does not occur in whatever X is instantiated to) and permutations π, borrowed from nominal terms.

$$\text{If } b\#X \text{ then } \lambda a.X =_\alpha \lambda b.(b\ a)\cdot X.$$

Here $(b\ a)\cdot b = a$, $(b\ a)\cdot a = b$, and $(b\ a)\cdot c = c$. It is a swapping.

$\lambda X.\lambda a.X =_\alpha \lambda X.\lambda b.X$ is never true; we cannot control the input to X so we cannot guarantee $b\#X$.

However, instantiating X to a in $b\#X \vdash \lambda a.X$ — a term-in-context — note that $b\#a$ and

$$\lambda a.a =_\alpha \lambda b.(b\ a)\cdot a \equiv \lambda b.b.$$