Two-and-a-halfth-order lambda-calculus

Murdoch J. Gabbay

WFLP, Siena, Italy
Friday 4 July 2008

Joint work with Dominic Mulligan

What are two-and-a-half levels?

Many of the basic systems of computer science, such as the lambda-calculus, first-order logic, or the pi-calculus, admit natural specifications involving

- object-level variables (‘level 1’),
- meta-level variables (‘level 2’), and
- freshness conditions.

For example:

Two levels

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

These informal statements mention two levels of variable; level 1 object-variables x, y and level 2 meta-variables r, t, P, Q, ϕ, ψ.

Capture-avoidance conditions are (freshness) constraints relating level 1 variables and the values that level 2 variables may assume.

Two levels

Meta-variables are naturally substituted with capturing substitution.

Consider the following quote:

> “Set r to xy in $\lambda y \lambda x.r$;
> obtain $\lambda y.\lambda x.xy$.”

Level 1 and level 2

This motivates a λ-calculus which is two copies of the λ-calculus glued together:

- At level 1, the 'object-level' calculus, has level 1 variables (atoms) $a, b, c, x, y, z, \ldots$.
- At level 2, the 'meta-level' calculus, has level 2 variables (unknowns) $X, Y, Z, R, T, \ldots$.
- Within each level (level 1, level 2), α- and β-conversion are as standard.
- Between levels, level 2 β-reduction does not avoid capture by level 1 λ-abstractions, modelling informal practice. For example . . .

Level 1 and level 2

"Set r to xy in $\lambda y.\lambda x.r$, obtain $\lambda y.\lambda x.xy$"

is modelled by:

$$(\lambda R.\lambda y.\lambda x.R)(xy) \to \lambda y.\lambda x.(xy)$$

Note that the β-reduction of R does not avoid capture.

This cannot be directly expressed in the 'ordinary' λ-calculus, where β-reduction always avoids capture:

$$(\lambda r.\lambda y.\lambda x.r)(xy) \to \lambda y'.\lambda x'.(xy)$$

The importance of having two levels

The λ-calculus, first- and higher-order logic, and the π-calculus have been well-studied.

The common language in which we study them — if one such language exists — has not been well-studied, or even agreed upon.

The importance of having a two level λ-calculus

There are several reasons to study a two-level λ-calculus:

- It models informal practice, formalises it, and makes it amenable to study.
- It does not require a logical framework (cf. HOAS; this gives you HOAS terms, but requires you to use a HOAS framework).
- The λ-calculus can be used as the basis of logics and theorem-provers.

  A two-level λ-calculus is a step towards building two level logics and theorem-provers which model informal practice in new ways.

  Speculative examples follow . . .

Examples

We indicate types with subscripts:

- $\forall P_o.(a_o \# P_o \Rightarrow P_o \Rightarrow \forall a_o.P_o)$

  Here o is a type of truth-values. $\forall$ is short for $\forall\lambda$ where $\forall$ is a constant symbol. $\#$ is short for $\#\lambda$ where $\#$ is a constant symbol intended to internalise the nominal freshness judgement. This models 'for all ϕ, if $a \notin \mathsf{fv}(\phi)$ then $\phi \Rightarrow \forall a.\phi$'.

- $\forall X_\alpha.(a_\beta \# X_\alpha \Rightarrow \lambda a_\beta.(X_\alpha a_\beta) = X_\alpha)$

  Here $=$ is a constant symbol, written infix. α and β are intended to be arbitrary types. This models η-equivalence (extensionality) at level 1.

Examples

- $\forall P_o.(\text{И} a_{\mathbb{A}}.\neg P_o) \Leftrightarrow \neg \text{И} a_{\mathbb{A}}.P_o$.

  Here И is short for $\text{И}\lambda$ where И is a constant symbol intended to internalise the Gabbay-Pitts ‘new’ quantifier [**?**]. $\neg$ and $\Leftrightarrow$ are constant symbols. $\mathbb{A}$ is a ‘type of atoms’ with no term-formers. This models the self-duality of И.

The axioms have mathematical force because they have been studied in previous work with level 2 variables but (since nominal terms have no λX) without a level 2 quantification explicitly represented in the syntax.

The importance of having two levels

Capture-avoiding substitution and all that surrounds in $(\lambda, \forall, \ldots)$ is well-studied.

Capturing substitution and what surrounds it, is not so well-studied. This is a source of difficult, interesting, and virgin mathematical problems.

This work is also part of a broader enquiry into names; it gives a functional semantics to nominal terms unknowns.

'Nominal terms' are a 'one-and-a-halfth' order system. Nominal terms have level 1 variables (atoms) and level 2 variables (unknowns). Nominal terms give level 2 variables no mathematical semantics. You can think of two-and-a-halfth order λ-calculus as 'functional semantics for nominal terms unknowns' — an operational one.

Technical details

That concludes the first half of my talk, designed to motivate and give background and informal intuitions.

In the second half I will sketch the system in more technical detail.

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 level 1 variables. 'Finitely supported' means $\pi(a) = a$ for all but finitely many level 1 variables.

Define syntax by:

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

This is two λ-calculi, level 1 at λa, level 2 at λX, glued together by being in the one syntax and joined at a shared application.

Level 2 interacting with level 1

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

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

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

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

Within a single level everything is as usual:

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

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

Free level 2 variables of

$$fv(a) = \{\} \qquad fv(\pi \cdot X) = \{X\}$$

$$fv(r'r) = fv(r') \cup fv(r)$$

$$fv(\lambda a.r) = fv(r) \qquad fv(\lambda X.r) = fv(r) \setminus \{X\}$$

We all know that we need this to express capture-avoidance conditions of level 2 substitution:

Level 2 substitution

$$a[X := t] \equiv a \qquad (\pi \cdot X)[X := t] \equiv \pi \cdot t$$

$$(\pi \cdot Y)[X := t] \equiv \pi \cdot Y \qquad (\lambda a.r)[X := t] \equiv \lambda a.(r[X := t])$$

$$(r'r)[X := t] \equiv (r'[X := t])(r[X := t])$$

$$(\lambda Y.r)[X := t] \equiv \lambda Y.(r[X := t]) \quad (Y \notin fv(t))$$

Capture-avoidance at level 1

It is not clear what the free level 1 variables of X in $\lambda a.X$ are. If we decide $fv(X) = \emptyset$ then we α-convert as follows

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

and we get wrong results because, for example

$$(\lambda X.\lambda a.X)a \to \lambda a.a \qquad (\lambda X.\lambda b.X)a \to \lambda b.a.$$

Thus, X represent an 'unknown element' in a capturing sense, and so has an unknown — an infinite — set of level 1 free variables (only finitely many of which will ever be taken up by a given level 2 β-reduct).

The notion of 'free level 1 variables' is inverted to the notion of 'level 1 freshness' $a\#r$:

Freshness

$$\frac{}{\Delta \vdash a\#b}\,(\mathbf{a\#b}) \qquad \frac{}{\Delta \vdash a\#\lambda a.r}\,(\mathbf{a\#\lambda a}) \qquad \frac{\Delta \vdash a\#r}{\Delta \vdash a\#\lambda b.r}\,(\mathbf{a\#\lambda b})$$

$$\frac{\pi^{\text{-}1}(a)\#X \in \Delta}{\Delta \vdash a\#\pi{\cdot}X}\,(\mathbf{a\#X}) \qquad \frac{\Delta \vdash a\#r' \quad \Delta \vdash a\#r}{\Delta \vdash a\#r'r}\,(\mathbf{a\#app})$$

$$\frac{\Delta, a\#X \vdash \pi(a)\#\pi{\cdot}r \quad (X \notin \Delta)}{\Delta \vdash \pi(a)\#\pi{\cdot}(\lambda X.r)}\,(\mathbf{a\#\lambda X})$$

An example freshness derivation, including level 2 abstraction

$$\cfrac{\cfrac{\overline{a\#X \vdash a\#X}\;(\mathbf{a\#X})}{a\#X \vdash a\#\lambda b.X}\;(\mathbf{a\#\lambda b})}{\vdash a\#\lambda X.\lambda b.X}\;(\mathbf{a\#\lambda X})$$

What's interesting here is that $a\#\lambda b.X$ is not derivable (unless we assume $a\#X$), but $a\#\lambda X.\lambda b.X$ is.

Permutation

$$\pi \cdot a \equiv \pi(a) \qquad \pi \cdot (\pi' \cdot X) \equiv (\pi \circ \pi') \cdot X$$

$$\pi \cdot (r'r) \equiv (\pi \cdot r')(\pi \cdot r) \qquad \pi \cdot (\lambda a.r) \equiv \lambda \pi(a).(\pi \cdot r)$$

$$\pi \cdot (\lambda X.r) \equiv (\lambda X.\pi \cdot r[X := \pi^{-1} \cdot X])$$

Then define α-equivalence as follows:

$$b\#r \Rightarrow \lambda a.r =_\alpha \lambda b.(b\ a) \cdot r.$$

We use use level 1 permutation rather than level 1 substitution because it interacts smoothly with level 1 and level 2 abstraction.

Congruence

$$\frac{\Delta \vdash r \rhd s}{\Delta \vdash \lambda a.r \rhd \lambda a.s}(\rhd\lambda\mathbf{a}) \qquad \frac{\Delta \vdash r \rhd s \quad \Delta \vdash t \rhd u}{\Delta \vdash rt \rhd su}(\rhd\mathbf{app})$$

$$\frac{\Delta \vdash r \rhd s \quad (X \notin \Delta)}{\Delta \vdash \lambda X.r \rhd \lambda X.s}(\rhd\lambda\mathbf{X}) \qquad \frac{\Delta \vdash r \rhd s \quad \Delta \vdash a\#s \quad \Delta \vdash b\#s}{\Delta \vdash r \rhd (a\ b) \cdot s}(\rhd\alpha)$$

Reductions

$$\frac{}{a[a\mapsto t] \longrightarrow t}\ (\beta\mathbf{a}) \qquad \frac{a\#r}{r[a\mapsto t] \longrightarrow r}\ (\beta\#)$$

$$\frac{}{(\lambda X.r)t \longrightarrow r[X := t]}\ (\beta\mathbf{2}) \qquad \frac{a\#r}{(r'r)[a\mapsto t] \longrightarrow (r'[a\mapsto t])r}\ (\beta\mathbf{2app})$$

$$\frac{\mathsf{level}(r') = 1}{(r'r)[a\mapsto t] \longrightarrow (r'[a\mapsto t])(r[a\mapsto t])}\ (\beta\mathbf{1app})$$

$$\frac{b\#t}{(\lambda b.r)[a\mapsto t] \longrightarrow \lambda b.(r[a\mapsto t])}\ (\beta\lambda\mathbf{1}) \qquad \frac{(X \notin fv(t))}{(\lambda X.r)[a\mapsto t] \longrightarrow \lambda X.(r[a\mapsto t])}\ (\beta\lambda\mathbf{2})$$

Two β-rules

$\text{level}(r') = 1$ means 'r' does not mention any level 2 variables'.

$(\beta\mathbf{1app})$ and $(\beta\mathbf{2app})$ can be viewed as two parts of a single rule:

$$\frac{\text{level}(r') = 1 \quad \text{or} \quad a\#r}{(r'r)[a{\mapsto}t] \to (r'[a{\mapsto}t])(r[a{\mapsto}t])}$$

If $\text{level}(r') = 1$ and $\Delta \vdash a\#r$ we join $(\beta\mathbf{1app})$ and $(\beta\mathbf{2app})$ with $(\beta\#)$.

We know what goes wrong if we relax these conditions (see the paper) but we will probably not fully understand this until we understand a denotational semantics.

Conclusions

I'd like to reiterate the three reasons I'm doing this:

- This is an opportunity to ask some really fundamental mathematical questions. Essentially, the λ-calculus and associated mathematics have studied capture-avoiding substitution half to death, but capturing substitution, its syntax and semantics, is completely virgin territory.
- There should be a theorem-provers offering a 'nominal' model of informal practice.

  Informal practice has two levels of variable and freshness conditions — there should be a theorem-prover that does this, too.

Conclusions

- Nominal terms have been studied (they have good computational properties). The question of mathematical semantics of unknowns X (level 2 variables) has remained an open problem for several years. This paper gives an answer — not the final or only answer but it's the start of something which will run for a while.

Further reading:

- Nominal terms [gabbay:nomu-jv]
- Lambda context calculus [gabbay:lamcc]
- Two-and-a-halfth order lambda-calculus [gabbay:twoaah]
- One-and-a-halfth order logic [gabbay:oneaah-jv]