A NEW calculus of contexts

Murdoch J. Gabbay

1/2/2006, St Andrews, Scotland

The issue

I'd like to talk about the λ-calculus.

How original.

No, wait! I have something NEW to say.

The issue

Consider the term $\lambda x.t$.

x is a variable symbol and t is a meta-level variable, ranging over λ-terms.

Instantiation of t does not avoid capture: if we set t to be x, we get $\lambda x.x$.

The issue

Claim: This is the essence of the meta-level.

Substitution of ‘strong’ (meta-level) variables for ‘weak’ (object-level) variables does not avoid capture.

Substitution of variables of the same level does avoid capture.

The issue

There are many things we can do with this idea.

1. Semantics.
2. Logic with proof-theory.
3. Algebra.
4. λ-calculus.

I'm trying to get at a greater truth, but I can't hang around for ten years till I get it 'just right'.

Let's base a calculus on this idea.

The issue

Suppose x is weak (level 1, say) and X is stronger (level 2, say), then

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

This is important.

Yes, important!

Why formalise the meta-level?

It’s what we use to make programs, do logic, etcetera; whether we do this formally or not, it’s there.

A formal framework which accurately represents our intention when we write ‘$\lambda x.t$’, including how t is instantiated, would be valuable.

Difficulty: α-equivalence

If $\lambda x.X = \lambda y.X$ then $(\lambda X.\lambda x.X)x \leadsto \lambda y.x$.

This is bad.

Some capture-avoidance remains legitimate, to be able to reduce terms like

$$(\lambda y.\lambda x.y)x \qquad \text{to} \qquad \lambda x'.x.$$

Technically, I shall use ideas originating from work with Urban and Pitts (just after my thesis), later developed further with Fernández, and investigated subsequently to this paper with Mathijssen, to control this.

The syntax

Suppose sets of variables $a_i, b_i, c_i, n_i, \ldots$ for $i \geq 1$.

a_i has level i. Syntax is given by:

$$s, t ::= a_i \mid tt \mid \lambda a_i.t \mid t[a_i \mapsto t] \mid \text{И} a_i.t.$$

- $s[a_i \mapsto t]$ is explicit substitution.
- $\lambda a_i.t$ is abstraction.
- $\text{И} a_i.t$ a binder.

Equate up to И-binding, nothing else.

Call b_j stronger than a_i when $j > i$.

E.g. b_3 is stronger than a_1.

Example terms and reductions

x, y, z have level 1. X, Y, Z have level 2.

$(\lambda x.x)y \rightsquigarrow x[x \mapsto y] \rightsquigarrow y$	Ordinary reduction
$(\lambda x.X)[X \mapsto x] \rightsquigarrow \lambda x.(X[X \mapsto x]) \rightsquigarrow \lambda x.x$	Context substitution
$x[X \mapsto t] \rightsquigarrow x$	X stronger than x
$x[x' \mapsto t] \rightsquigarrow x$	Ordinary substitution
$x[x \mapsto t] \rightsquigarrow t$	Ordinary substitution
$X[x \mapsto t] \not\rightsquigarrow$	Suspended substitution

Records

Fix constants 1 and 2.

l and m have level 1, X has level 2.

A record:

$$X[l\mapsto 1][m\mapsto 2]$$

A record lookup:

$$\begin{aligned} X[l\mapsto 1][m\mapsto 2][X\mapsto m] &\rightsquigarrow X[l\mapsto 1][X\mapsto m][m\mapsto 2] \\ &\rightsquigarrow X[X\mapsto m][l\mapsto 1][m\mapsto 2] \\ &\rightsquigarrow m[l\mapsto 1][m\mapsto 2] \\ &\rightsquigarrow m[m\mapsto 2] \\ &\rightsquigarrow 2. \end{aligned}$$

In-place update

$$
\begin{aligned}
X[l\mapsto 1][m\mapsto 2][X\mapsto X[l\mapsto 2]] &\leadsto X[l\mapsto 1][X\mapsto X[l\mapsto 2]][m\mapsto 2] \\
&\leadsto X[X\mapsto X[l\mapsto 2]][l\mapsto 1][m\mapsto 2] \\
&\leadsto X[l\mapsto 2][l\mapsto 1][m\mapsto 2] \\
&\leadsto X[l\mapsto 2][m\mapsto 2]
\end{aligned}
$$

Substitution-as-a-term

$(\lambda X.X[l \mapsto \lambda n.n])$ applied to lm

$$(\lambda X.X[l \mapsto \lambda n.n])(lm) \leadsto X[l \mapsto \lambda n.n][X \mapsto lm] \leadsto^{*} (\lambda n.n)m$$

In-place update as a term

$$\lambda\mathcal{W}.\mathcal{W}[X{\mapsto}X[l{\mapsto}2]] \quad \text{applied to} \quad X[l{\mapsto}1][m{\mapsto}2]$$

...and so on ($\mathcal{W}$ has level 3).

Likewise global state (world = a big hole), and Abadi-Cardelli imp-ε object calculus.

Records (again, using λ)

Fix constants 1 and 2.

l and m have level 1. X has level 2.

A record:

$$\lambda X.X[l\mapsto 1][m\mapsto 2].$$

Now we use application to retrieve the value stored at m:

$$(\lambda X.X[l\mapsto 1][m\mapsto 2])m \leadsto X[l\mapsto 1][m\mapsto 2][X\mapsto m]$$

Records (again, using λ)

$$\lambda X.X[l{\mapsto}\mathcal{W}][m{\mapsto}2]$$

Here $\mathcal{W}$ has level 3. It beats X, l, and m.

Apply $[\mathcal{W}{\mapsto}X]$:

$$\Big(\lambda X.X[l{\mapsto}\mathcal{W}][m{\mapsto}2]\Big)[\mathcal{W}{\mapsto}X] \rightsquigarrow^* \lambda X.X[l{\mapsto}X][m{\mapsto}2].$$

Apply to (lm) and obtain $(l2)2$:

$$\Big(\lambda X.X[l{\mapsto}X][m{\mapsto}2]\Big)(lm) \rightsquigarrow^* lm[l{\mapsto}lm][m{\mapsto}2] \rightsquigarrow^* (l2)2$$

Records (again, using λ)

$$\Big(\lambda\mathcal{W}.\lambda X.X[l{\mapsto}\mathcal{W}][m{\mapsto}2]\Big)X(lm) \leadsto^* (l2)2$$

Is that wrong?

Depends what you want.

This kind of thing makes the Abadi-Cardelli ‘self’ variable work. The issue is that λ does not bind — it abstracts.

$$\text{И}X.\big(\lambda X.X[l{\mapsto}\mathcal{W}][m{\mapsto}2]\big).$$

Then

$$\big(\text{И}X.\lambda X.X[l{\mapsto}\mathcal{W}][m{\mapsto}2]\big)[\mathcal{W}{\mapsto}X]$$
$$\rightsquigarrow^* \text{И}X'.(\lambda X'.X'[l{\mapsto}\mathcal{W}][m{\mapsto}2][\mathcal{W}{\mapsto}X])$$
$$\rightsquigarrow^* \text{И}X'.\lambda X'.X'[l{\mapsto}X][m{\mapsto}2]$$

Apply to lm:

$$\text{И}X'.(\lambda X'.X'[l{\mapsto}X][m{\mapsto}2])\ (lm)$$
$$\rightsquigarrow \text{И}X'.((\lambda X'.X'[l{\mapsto}X][m{\mapsto}2])\ (lm))$$
$$\rightsquigarrow \text{И}X'.X'[l{\mapsto}X][m{\mapsto}2][X'{\mapsto}lm] \rightsquigarrow^* (X[m{\mapsto}2])2$$

И behaves like the π-calculus ν; it floats to the top (extrudes scope).

How the different ‘bits’ fit together

1. λ abstracts — it stays put and β-reduces.
2. $[x \mapsto s]$ substitutes — it floats downwards capturing x until it runs out of term or gets stuck on a stronger variable.
3. N binds — it floats upwards avoiding capture.

Reduction rules

(β)	$(\lambda a_i.s)u \leadsto s[a_i{\mapsto}u]$	
(σa)	$a_i[a_i{\mapsto}u] \leadsto u$	$\forall c.c\#a_i{\Rightarrow}c\#u$
$(\sigma\#)$	$s[a_i{\mapsto}u] \leadsto s$	$a_i\#s$
(σp)	$(a_i t_1 \ldots t_n)[b_j{\mapsto}u] \leadsto (a_i[b_j{\mapsto}u]) \ldots (t_n[b_j{\mapsto}u])$	
$(\sigma\sigma)$	$s[a_i{\mapsto}u][b_j{\mapsto}v] \leadsto s[b_j{\mapsto}v][a_i{\mapsto}u[b_j{\mapsto}v]]$	$j{>}i$
$(\sigma\lambda)$	$(\lambda a_i.s)[c_k{\mapsto}u] \leadsto \lambda a_i.(s[c_k{\mapsto}u])$	$a_i\#u, c_k\ k{\leq}i$
$(\sigma\lambda')$	$(\lambda a_i.s)[b_j{\mapsto}u] \leadsto \lambda a_i.(s[b_j{\mapsto}u])$	$j > i$
(σtr)	$s[a_i{\mapsto}a_i] \leadsto s$	
$(\text{И}p)$	$(\text{И}n_j.s)t \leadsto \text{И}n_j.(st)$	$n_j{\notin}t$
$(\text{И}\lambda)$	$\lambda a_i.\text{И}n_j.s \leadsto \text{И}n_j.\lambda a_i.s$	$n_j{\neq}a_i$
$(\text{И}\sigma)$	$(\text{И}n_j.s)[a_i{\mapsto}u] \leadsto \text{И}n_j.(s[a_i{\mapsto}u])$	$n_j{\notin}u\ n_j{\neq}a_i$
$(\text{И}{\notin})$	$\text{И}n_j.s \leadsto s$	$n_j{\notin}s$

Graphs (if I have time)

Here is a fun NEW calculus of contexts program:

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

Observe $s(xy) \rightsquigarrow^* (yy)(xx)$.

Free variables behave like dangling edges in graphs; stronger variables behave like holes.

What is the ‘geometry’ of a NEWCC term?

Partial evaluation (if I have time)

Write

$$\texttt{if} = \lambda a, b, c.abc \quad \texttt{true} = \lambda ab.a \quad \texttt{false} = \lambda ab.b$$
$$\texttt{not} = \lambda a.\texttt{if}\ a\ \texttt{false}\ \texttt{true}.$$

in untyped λ-calculus. Then calculate

$$s = \lambda f, a.\texttt{if}\ a\ (f\ a)\ a \qquad \text{specialised to} \qquad s\ \texttt{not}$$

by β-reduction. We obtain $\lambda a.\texttt{if}\ a\ (\texttt{not}\ a)\ a$.

A more intelligent method may recognise that the program will always return `false` (with types etc.).

Partial evaluation (if I have time)

Choose level 1 variables a, b and level 2 variables and B, C and define

$$\mathtt{true} = \lambda ab.a \quad \mathtt{false} = \lambda ab.b$$

$$\mathtt{if} = \lambda a, B, C.\, a(B[a{\mapsto}\mathtt{true}])(C[a{\mapsto}\mathtt{false}])$$

$$\mathtt{not} = \lambda a.\mathtt{if}\ a\ \mathtt{false}\ \mathtt{true}.$$

So if we get to B, $a = \mathtt{true}$. Consider

$$s = \lambda f, a.\mathtt{if}\ a\ (f\ a)\ a \qquad \text{specialised to} \qquad s\ \mathtt{not}.$$

We obtain:

$$\begin{aligned} s\ \mathtt{not} &\rightsquigarrow^* \lambda a.a\ ((\mathtt{not} B)[a{\mapsto}\mathtt{true}][B{\mapsto}a])\ (C[a{\mapsto}\mathtt{false}][C{\mapsto}a]) \\ &\rightsquigarrow^* \lambda a.a\ ((\mathtt{not} a)[a{\mapsto}\mathtt{true}])\ (a[a{\mapsto}\mathtt{false}]) \\ &\rightsquigarrow^* \lambda a.(a\ \mathtt{false}\ \mathtt{false}). \end{aligned}$$

More efficient!

Other applications

Dynamic (re)binding.

Staged computation. Our calculus is a pure rewrite system. However, a programming language based on it can model staged computation (I think).

Complexity. Can we write more efficient programs?

Geometry. What is a notion of Böhm tree (or similar), in the presence of strong variables?

Meta-properties

- Confluence.
- Preservation of strong normalisation (for untyped lambda-calculus).
- Hindley-Milner type system. Explicit substitution rule is like that for `let`.
- Applicative characterisation of contextual equivalence.

Conclusions

The meta-level lives in the same world as the object calculus. So does the meta-meta-level. And so on.

Scope separate from abstraction; necessary for proper control of α-equivalence in the presence of the hierarchy.

Hierarchy of strengths of variables in common with work by Sato et al. But we have different control of α-equivalence.

Explicit substitution calculus.

Unexpectedly: model of state, unordered datatypes, objects, graphs, and more.