A NEW calculus of contexts

Murdoch J Gabbay

Tel Aviv University, Israel 14/5/2006

This talk. . .

. . . is the third in a series of about four talks in the framework of a mini-course describing (some? most?) of the mathematics I've done over the past six years (since I got my PhD).

Motivation

In this talk I'll discuss the NEW calculus of contexts, see my webpage `www.gabbay.org.uk` for the paper [PPDP'05].

Motivation

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

How original.

No, wait! I have something NEW to say.

Motivation

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 essence of the meta-level

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

- Substitution of ‘strong’ (meta-level) variables for ‘weak’ (object-level) variables does not avoid capture. Call this instantiation.
- Substitution of variables of the same level does avoid capture.

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, is worthy of serious mathematical investigation.

Why formalise the meta-level?

In this course we have already seen the following based on this philosophy and accompanying mathematics:

1. Semantics.

2. Logic with proof-theory.

3. Algebra.

Let's now look at a calculus, i.e. do programming.

An example

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

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

Difficulty: α-equivalence

If $\lambda x.X$ and $\lambda y.X$ are equivalent then

$$(\lambda X.\lambda x.X)x \rightsquigarrow \lambda y.x.$$

This is undesirable. Yet some capture-avoidance remains legitimate, e.g. we still want $\lambda x.x$ to be equivalent to $\lambda y.y$.

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:

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

- $s[a_i \mapsto t]$ 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:

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

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

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

Substitution-as-a-term

$$(\lambda X.X[l \mapsto \lambda n.n]) \quad \text{applied to} \quad lm$$

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

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)

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

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.

И

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

Then

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

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

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

И

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. И binds — it floats upwards avoiding capture.

Implementation of the untyped λ-calculus

Terms of the untyped λ-calculus:

$$s ::= a \mid ss \mid \lambda a.s$$

quotiented by α-equivalence as usual.

Translation into the NEWcc is:

$$[\![a]\!] \equiv a \qquad [\![ss']\!] = [\![s]\!][\![s']\!] \qquad [\![\lambda a.s]\!] = \text{И}a.\ \lambda a.[\![s]\!].$$

Theorem: NEWcc reductions simulate λ-calculus reductions, and they preserve strong normalisation.

Reduction rules

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

Graphs

Here is a fun NEW calculus of contexts program:

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

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

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

What is the ‘geometry’ of a NEWCC term?

Partial evaluation

Write

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

in untyped λ-calculus. Then calculate

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

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

A more intelligent method may recognise that the program will always return $\mathtt{false}$ (with types etc.).

Partial evaluation

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

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

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

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

Partial evaluation

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:

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

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?

Types (briefly)

$$\frac{x : \sigma \in \Gamma \quad \tau \preceq \sigma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma, a_i : \tau \vdash s : \tau'}{\Gamma \vdash \lambda a_i.s : \tau \rightarrow \tau'}$$

$$\frac{\Gamma \vdash s' : \tau' \quad \Gamma, a_i : \forall\overline{\alpha}.\tau' \vdash s : \tau \qquad \overline{\alpha} = \textit{tyv}\,(\tau') \setminus \textit{tyv}\,(\Gamma)}{\Gamma \vdash s[a_i \mapsto s'] : \tau}$$

$$\frac{\Gamma, n_j : \alpha \vdash s : \tau \quad n_j, \alpha \notin \Gamma}{\Gamma \vdash \text{И} n_j.\, s : \tau} \qquad \frac{\Gamma \vdash s : \tau \rightarrow \tau' \quad \Gamma \vdash t : \tau}{\Gamma \vdash st : \tau'}$$

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

With the NEWcc, we really can meta-program.

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.