

A NEW calculus of contexts

Murdoch J. Gabbay

PPDP 2005, 10/7/2005, Lisbon

The issue

Context=‘term with a hole’: $C[-] = \lambda x.[-]$.

$[-]$ may be filled in a capturing manner: $C[x] = \lambda x.x$.

This is **not** modelled by β -reduction since it avoids capture; consider $C = (\lambda y.\lambda x.y)$. Then $Cx \rightsquigarrow^* \lambda x'.x$ — **wrong!**

$C[-]$ **is** modelled by β -reduction **if** you have types and application: write $C = \lambda F.\lambda x.Fx$. Then $C\lambda y.y \rightsquigarrow^* \lambda x.x$.

The issue

My vision: something **NEW**.

Suppose a hierarchy of **levels** of variables of increasing **strength**.

Abstraction and application are (more-or-less) as before. However, substitution for a variable **avoids** capture for stronger variables under weaker variables, and **does not avoid capture** for weaker variables under stronger variables.

For example, if x is weak (level 1, say) and X is stronger (level 2, say), then $C = (\lambda X. \lambda x. X)$ and

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

The issue

Problem: α -equivalence.

If $\lambda x.X = \lambda y.X$ then $(\lambda X.\lambda x.X)x \rightsquigarrow \lambda y.x$. This would be bad!

Dropping α -equivalence entirely is too drastic. Some capture-avoidance, as in $(\lambda y.\lambda x.y)x$, should be legitimate.

Result: We solve these issues and obtain not just a ‘calculus for contexts’, but a calculus for **Records, Objects, Modules, Partial Evaluation, Dynamic Binding**, and possibly Staged Computation . . . — and with good meta-properties including confluence, preservation of strong normalisation, Hindley-Milner types, and an applicative characterisation of contextual equivalence.

Syntax

Suppose countably infinite set of disjoint infinite **sets of variables** $a_i, b_i, c_i, n_i, \dots$ for $i \geq 1$. Say a_i **has level** i . Syntax is given by:

$$s, t ::= a_i \mid tt \mid \lambda a_i.t \mid t[a_i \mapsto t] \mid \forall a_i.t.$$

Call $s[a_i \mapsto t]$ an **explicit substitution**, $\lambda a_i.t$ an **abstraction**, and $\forall a_i.t$ a **binder**.

Terms are equated up to binding by \forall and nothing else.

Call a variable b_j **stronger** than another a_i when $j > i$ (when it has strictly higher level). b_3 is stronger than a_1 .

Example terms and reductions

Let x, y, z have level 1 and 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.

Here is a record:

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

Here is 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

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

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

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

I'm **telling** you we can proceed to global state (the world is a big hole with state suspended on it, just like a record), and Abadi-Cardelli imp- ϵ object calculus. For details, see the paper.

Records (again, using λ)

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

Here is a record:

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

This is just as before, but now we must use an application, to m say, to retrieve the value stored at m :

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

(same as before on Slide 7).

But what about

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

\mathcal{W} is a level 3 variable, so it beats X , l , and m .

If we apply $[\mathcal{W} \mapsto X]$ we obtain (after some reduction)

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

Apply this to l and we obtain 2 . Is that wrong?

$$\begin{aligned} (\lambda X. X [l \mapsto X] [m \mapsto 2])l &\rightsquigarrow X [l \mapsto 1] [m \mapsto 2] [X \mapsto l] \\ &\rightsquigarrow^* l [l \mapsto m] [m \mapsto 2] \rightsquigarrow^* 2 \end{aligned}$$

Maybe, maybe not. It depends. This kind of thing makes the Abadi-Cardelli ‘self’ variable work. But perhaps we do not want this. The problem is, λ does not **bind**, it only **abstracts**. We still need a binder. No problem.

Introduce \mathbb{N}

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

Then

$$\begin{aligned} & (\mathbb{N}X.\lambda X.X[l \mapsto \mathcal{W}][m \mapsto 2])[\mathcal{W} \mapsto X] \\ & \rightsquigarrow^* \mathbb{N}X'.(\lambda X'.X'[l \mapsto \mathcal{W}][m \mapsto 2][\mathcal{W} \mapsto X]) \\ & \rightsquigarrow^* \mathbb{N}X'.\lambda X'.X'[l \mapsto X][m \mapsto 2] \end{aligned}$$

Good! Apply **this** to l and we get X .

$$\begin{aligned} \mathbb{N}X'.(\lambda X'.X'[l \mapsto X][m \mapsto 2]) X & \rightsquigarrow \mathbb{N}X'.(\lambda X'.X'[l \mapsto X][m \mapsto 2] X) \\ & \rightsquigarrow \mathbb{N}X'.X'[l \mapsto X][m \mapsto 2][X' \mapsto X] \rightsquigarrow^* X \end{aligned}$$

\mathbb{N} behaves like the π -calculus ν ; it floats to the top (extrudes scope).

Summary

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. \mathcal{N} binds — it floats upwards avoiding capture.

Reduction rules

$$\begin{array}{ll}
 (\beta) & (\lambda a_i . s)u \rightsquigarrow s[a_i \mapsto u] \\
 (\sigma a) & a_i[a_i \mapsto u] \rightsquigarrow u \qquad \forall c. c \# a_i \Rightarrow c \# u \\
 (\sigma \#) & s[a_i \mapsto u] \rightsquigarrow s \qquad a_i \# s \\
 (\sigma p) & (a_i t_1 \dots t_n)[b_j \mapsto u] \rightsquigarrow (a_i[b_j \mapsto u]) \dots (t_n[b_j \mapsto u]) \\
 (\sigma \sigma) & s[a_i \mapsto u][b_j \mapsto v] \rightsquigarrow s[b_j \mapsto v][a_i \mapsto u[b_j \mapsto v]] \qquad j > i \\
 (\sigma \lambda) & (\lambda a_i . s)[c_k \mapsto u] \rightsquigarrow \lambda a_i . (s[c_k \mapsto u]) \qquad a_i \# u, c_k \ k \leq i \\
 (\sigma \lambda') & (\lambda a_i . s)[b_j \mapsto u] \rightsquigarrow \lambda a_i . (s[b_j \mapsto u]) \qquad j > i \\
 (\sigma tr) & s[a_i \mapsto a_i] \rightsquigarrow s \\
 (\forall p) & (\forall n_j . s)t \rightsquigarrow \forall n_j . (st) \qquad n_j \notin t \\
 (\forall \lambda) & \lambda a_i . \forall n_j . s \rightsquigarrow \forall n_j . \lambda a_i . s \qquad n_j \neq a_i \\
 (\forall \sigma) & (\forall n_j . s)[a_i \mapsto u] \rightsquigarrow \forall n_j . (s[a_i \mapsto u]) \qquad n_j \notin u \ n_j \neq a_i \\
 (\forall \notin) & \forall n_j . s \rightsquigarrow s \qquad n_j \notin s
 \end{array}$$

Graphs

Here is a fun NEW calculus of contexts program (hope you like it):

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

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

The hierarchy of variables allows us to inject terms into positions where their variables will be captured, either by a lambda or by an explicit substitution. Free variables behave like dangling 'edges'.

Partial evaluation

Write

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

in untyped λ -calculus. Then calculate

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

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

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

Partial evaluation

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

$$\begin{aligned}\text{true} &= \lambda ab.a & \text{false} &= \lambda ab.b \\ \text{if} &= \lambda a, B, C. a(B[a \mapsto \text{true}])(C[a \mapsto \text{false}]) \\ \text{not} &= \lambda a. \text{if } a \text{ false true}.\end{aligned}$$

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

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

We obtain:

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

More efficient!

Other applications

Dynamic (re)binding. Obviously, we could continue the techniques used to encode objects etc.

Threads. Add concurrency? π -calculus?

Staged computation (MetaML, Template Haskell, Converge) offer control execution; a program can suspend its own execution, compose suspended programs into larger (suspended) programs, pass suspended programs as arguments to functions, and evaluate them. This raises issues similar to those surrounding contexts.

Our calculus is a pure rewrite system. **However**, a **programming language** based on it **can** model staged computation.

Meta-properties

- Confluence.
- Preservation of strong normalisation for untyped lambda-calculus (λ maps to $\mathbb{N}\lambda$, otherwise straight injection).
- Hindley-Milner type system. Explicit substitution rule is like that for `let`.
- Applicative characterisation of contextual equivalence.

Conclusions

We have ideas of **scope** as a separate entity from **abstraction**.

This idea is imported from Nominal research, along with freshness contexts.

The calculus can be thought of a lambda-calculus for Nominal terms (but souped-up ones, because of...)

We have a hierarchy of strengths of variables, in common with work by Sato et al.

We have an explicit substitution calculus. This calculus is deliberately simple-minded treating substitutions, e.g. $(\sigma\sigma)$ and (σp) . However, the interaction with the hierarchy of variables seems interesting.

Conclusions

Technically, this work is a logical extension and application of long traditions and techniques in lambda-calculi, explicit substitution calculi, and calculi of contexts, with Nominal techniques applied in a non-trivial but reasonable manner consistent with obtaining certain desired meta-properties.

It seems to work!

This is not the last word on the subject by any means, but it opens exciting **new** possibilities.

By the way, what's the relation of the hierarchy of variables to types? Can this express graph reductions? What does the rewrite framework with a hierarchy of variables look like? Matching/Unification? Is there a corresponding logic (remove λ , include \forall and logical connectives)?