A NEW calculus of contexts

Murdoch J. Gabbay

Paris, INRIA and PPS, 1/2005

Thank you for fitting me in on short notice!

The issue

Context='term with a hole'. E.g. $C[\text{-}] = \lambda x.[\text{-}]$.

$[\text{-}]$ may be filled, e.g. $C[t] = \lambda x.t$, in a capturing manner, e.g. $C[x] = \lambda x.x$.

This is not modelled by β-reduction since it avoids capture; consider $(\lambda y.\lambda x.y)x \rightsquigarrow^* \lambda x'.x$.

Our vision: 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 variables of the same strength or stronger, and does not avoid capture for weaker variables.

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

The issue

Problem: α-equivalence.

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

Our answer: Separate scope (λ) and binding (И). Introduce freshness context to manage their interaction. Also use explicit substitution, because it is easy to express the reductions.

Result: Not solely a 'calculus for contexts', but also a calculus with good meta-properties *and* unexpected expressivity including things we might not have expected to have anything to do with contexts.

Syntax

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

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

Call $s[a_i \mapsto t]$ an explicit substitution, $\lambda a_i.t$ an abstraction, and $\text{И} a_i.t$ a binder.

Terms are equated up to binding by И 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

More on suspensions: substitution-as-a-term

Let t have level 3.

$$X[x\mapsto t][X\mapsto x] \rightsquigarrow X[X\mapsto x][x\mapsto t[X\mapsto x]]$$
$$\rightsquigarrow x[x\mapsto t[X\mapsto x]] \rightsquigarrow t[X\mapsto x].$$

$b_j[a_i\mapsto t]$ with $i<j$ is a strong hole b_j waiting to be filled so a weaker a_i can substitute in it.

$[a_i\mapsto t]$ is not a term and cannot be made an argument of a function. Using suspensions we can express it:

$$\lambda X.(X[x\mapsto y]) \quad \text{encodes} \quad [x\mapsto y].$$

Reduction is not possible because X is stronger than x. Contrast this with $\lambda X.(X[X\mapsto y])$, which reduces in one step to $\lambda X.y$.

Example of substitution-as-a-term

Let $\mathbf{true} \equiv \lambda x, y.x$, $\mathbf{false} \equiv \lambda x, y.y$, and $\textsf{\textbf{Id}} \equiv \lambda x.x$.

f takes a substitution-as-a-term, a truth value, and an argument, and applies the substitution or not according to the truth value:

$$f \equiv \lambda x, y, z.(yx\textsf{\textbf{Id}})z$$

For example:

$$f\ (\lambda X.X[x\mapsto y])\ \mathbf{true}\ x \rightsquigarrow^* \mathbf{true}(\lambda X.X[x\mapsto y])\textsf{\textbf{Id}}x$$
$$\rightsquigarrow^* (\lambda X.X[x\mapsto y])x \rightsquigarrow^* X[x\mapsto y][X\mapsto x] \rightsquigarrow^* y$$

Similarly, $f\ (\lambda X.X[x\mapsto y])\ \mathbf{false}\ x \rightsquigarrow^* x$.

$\reflectbox{N}$ and freshness

λ does not bind. It is an abstractor but not a binder. So we introduce $\reflectbox{N}$ to get α-equivalence (so, intuitively, $\reflectbox{N}x.\lambda x.blah$ is what we traditionally understand by 'lambda x $blah$').

($\reflectbox{N}$ scope-extrudes like in the π-calculus. λ stays put!)

So consider $\lambda X.\reflectbox{N}y.\lambda y.(X[y{\mapsto}0])$.

Here, y is generated *inside* the function call, so we should *know* y is not in X — and reduce to $\lambda X.\reflectbox{N}y.\lambda y.X$.

Freshness contexts $y\#X$ take care of that. We assume rewrites occur in a context of freshness information with enough fresh variables to satisfy all our needs. Because $\reflectbox{N}$ binds, we can rename y to some fresh variable which the context says is fresh for X. We then proceed.

Confused? *Let's make sure you are. . .*

(β)	$(\lambda a_i.s)u \rightsquigarrow s[a_i{\mapsto}u]$	
(σa)	$a_i[a_i{\mapsto}u] \rightsquigarrow u$	$\forall c.\ c\#a_i{\Rightarrow}c\#u$
$(\sigma\#)$	$s[a_i{\mapsto}u] \rightsquigarrow s$	$a_i\#s$
(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)	$s[a_i{\mapsto}a_i] \rightsquigarrow s$	
$(\text{И}p)$	$(\text{И}n_j.s)t \rightsquigarrow \text{И}n_j.(st)$	$n_j{\notin}t$
$(\text{И}\lambda)$	$\lambda a_i.\text{И}n_j.s \rightsquigarrow \text{И}n_j.\lambda a_i.s$	$n_j{\neq}a_i$
$(\text{И}\sigma)$	$(\text{И}n_j.s)[a_i{\mapsto}u] \rightsquigarrow \text{И}n_j.(s[a_i{\mapsto}u])$	$n_j{\notin}u\ n_j{\neq}a_i$
$(\text{И}{\notin})$	$\text{И}n_j.s \rightsquigarrow s$	$n_j{\notin}s$

WHY??

Semantics

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

Thus, the hierarchy of variables allows us to 'inject' terms into positions where their variables with be captured, either by a lambda or by an explicit substitution.

Well yes, that's what you'd expect of a (λ-)calculus of contexts, really — isn't it?

We can use this power to model many things.

Records

A record is $b_j[a^1_{i_1}\mapsto t_1]\cdots[a^n_{i_n}\mapsto t_n]$ where $j > i_k$ for $1 \leq k \leq n$.

In words, a record is a set of substitutions suspended on a 'big hole' b_j.

Let's be concrete. $R = X[l\mapsto t_l][p\mapsto t_p]$ stores t_l at l and t_p at p.

We retrieve t_l with $[X\mapsto l]$ and update it with $[X\mapsto X[l\mapsto newval]]$.

Call $\text{И}\lambda a_3.a_3[X\mapsto l]$ record lookup at l and $\text{И}\lambda a_3.a_3[X\mapsto X[l\mapsto t'_l]]$ record update at l.

$$
\begin{aligned}
(\text{И}\lambda a_3.a_3[X\mapsto l])R &\overset{(\beta)}{\rightsquigarrow} \text{И}a_3.a_3[X\mapsto l][a_3\mapsto R] \\
&\overset{(\sigma\sigma),(\sigma a)}{\rightsquigarrow^*} \text{И}a_3.R[X\mapsto l] \\
&\overset{(\sigma\sigma)}{\rightsquigarrow^*} \text{И}a_3.l[l\mapsto t_l[X\mapsto l]][p\mapsto t_p[X\mapsto l]] \\
&\overset{(\sigma a),(\text{И}\#)}{\rightsquigarrow^*} t_l[X\mapsto l].
\end{aligned}
$$

Similar reductions for record update.

Protected records

In-place update is also possible. E.g. $[X \mapsto X[l \mapsto l+1]]$ (or as a term, $\text{И}\lambda a_3.a_3[X \mapsto X[l \mapsto l+1]]$) adds 1 to t_l in-place.

Note that X is free in R. If we do not like that, we can use protected records.

$R' = \text{И}\lambda X.(X[l \mapsto t_l][p \mapsto t_p])$.

Protected record lookup at l is encoded by $\text{И}\lambda a_3.a_3 l$ and protected record update at l by $\text{И}\lambda a_3.\text{И}\lambda X.a_3(X[l \mapsto t'_l])$.

Note there is no possibility of substitution for some stronger variable than X being captured by λX, because it is protected by И, which explicitly marks the scope of X.

Global state, and object-oriented programming

in the sense of general references and Abadi-Cardelli **imp-**ε respectively, are easy to encode. For details, see the paper.

Possible applications to the theory of both, e.g. applicative characterisation of contextual equivalence obtained from a theorem which holds of the NEW context calculus (in the paper, a non-trivial result). This is speculative but if it works, it would be great because that kind of theorem is generally *very hard*.

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 $\texttt{false}$ (with types etc.).

Partial evaluation

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:

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

More efficient!

Other applications

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.

Other applications

Just the idea: in $s[a \mapsto t]$ restrict evaluations in t to those involving variables at least as strong as a. For example $a_3[a_1 \mapsto (\lambda a_2.1)0] \rightsquigarrow a_3[a_1 \mapsto 1]$, and $a_3[a_2 \mapsto (\lambda a_1.1)0]$ does not reduce. $a_3[a_2 \mapsto (\lambda a_2.\lambda a_1.a_2)11 \rightsquigarrow^* a_3[a_2 \mapsto \lambda a_1.1]$, because a_2 is strong enough to reduce under a substitution by a_2, but a_1 is not.

This to give enough control of execution flow to encode the *brackets*, *escape*, and *run* of MetaML (as well as other less exotic constructs, such as call-by-name and call-by-value versions of function application). Perhaps also retain good meta-theory. Perhaps compare different staged computation (or other) calculi in common language.

We can explore how well provable properties of the NEW context calculus transfer back along maps into it.

Other applications

There is a-logic, which has a predicate var such that $var(x)$ holds when x is a variable. There is the NEW context calculus, which has contexts. If we put them together, do we get a calculus/logic which can program unification of its own terms?

Meta-properties

- Confluence.
- Preservation of strong normalisation for untyped lambda-calculus.
- Hindley-Milner type system.
- Applicative characterisation of contextual equivalence.

Conclusions

We have ideas of scope as a separate entity from abstraction, from Nominal research, as well as the idea of a freshness context. The calculus can be thought of an operational semantics for heavily souped-up Nominal Terms.

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

Our meta-properties are good and in some directions non-trivial, for example the applicative characterisation of contextual equivalence. The applications seem new, for example our suggestion that calculi of contexts may model state, objects, and even staged computation. Though we make these claims formal, more work clearly remains to be done.

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.