Lambda context calculus

Murdoch J. Gabbay, Heriot-Watt University, Scotland

LFMTP, CADE, Bremen
Sunday, 15 July 2007

Joint work with Stéphane Lengrand, St Andrew's University, Scotland

Context of contexts

Amongst other things I'm interested in contexts in the λ-calculus and logic.

By context I mean the stuff that 'surrounds' terms and which may bind variables in them:

$$\lambda a.t$$

is context surrounding the λ-term t.

$$\forall a.\phi$$

is a context surrounding the first-order logic predicate ϕ.

Put a term in a context, and you get another term.

Contexts on contexts

Contexts are used in (informal specifications of) rewrite and derivation rules. β- and η-reduction for example refer to the top-level structure of a term, as does the derivation rules $\forall R$:

$$(\lambda a.s)t \longrightarrow s[a \mapsto t] \qquad a\#s \Rightarrow \lambda a.(sa) = s \qquad \frac{\Gamma \vdash \phi \quad [a \notin \Gamma]}{\forall a.\phi}$$

Contexts on contexts

Rewrites and derivation rules are usually understood to operate on terms — but they use contexts to do so.

This shows up directly in the theory:

Context of contexts

$$\dfrac{\dfrac{A{\Rightarrow}B{\Rightarrow}C \quad [A]^i}{B{\Rightarrow}C} \qquad \dfrac{?}{B}}{\dfrac{C}{A{\Rightarrow}C}\, i} \qquad\qquad \dfrac{\dfrac{A{\Rightarrow}B{\Rightarrow}C \quad [A]^i}{B{\Rightarrow}C} \qquad \dfrac{A{\Rightarrow}B \quad [A]^i}{B}}{\dfrac{C}{A{\Rightarrow}C}\, i}$$

Both derivations above are of $A{\Rightarrow}B{\Rightarrow}C, A{\Rightarrow}B \vdash A{\Rightarrow}C$ but the left-hand one is incomplete.

Discharge means that we have to be able to instantiate $?$ in an incomplete derivation for an assumption which will be discharged. Discharge corresponds in the Curry-Howard correspondence to λ-abstraction. Instantiation corresponds to capturing substitution.

Context on contexts

So contexts have to do with capturing substitution.

Just a little bit more context on contexts

At its most simple I want a direct model of what happens when we write:

> 'Let t be a in $\lambda a.t$. We get $\lambda a.a$.'

Call this instantiation (non-capture-avoiding substitution).

Instantiation is central to informal mathematics — as Randy said, the mathematics where we mean what we say and we say what we mean — so this is an interesting and important question.

λ-abstraction and function application aren't it. $\lambda f.(\lambda a.f)a =_\beta \lambda a'.a$.

Lambda context calculus

Suppose disjoint infinite sets of variables $\mathbb{A}_1$, $\mathbb{A}_2$, ...

$i, j, k \in \{1, 2, 3, \ldots\}$ are levels.

$a_i \in \mathbb{A}_i$ is a meta-variable ranging over elements of $\mathbb{A}_i$; we say a_i has level i. Similarly for $b_j \in \mathbb{A}_j$. If $j > i$ call a_i weaker than b_j.

We use a permutative convention that $a_i, b_j, c_k, \ldots$ are a_i, b_j, and c_k are always distinct variables.

x, y, z are particular elements of $\mathbb{A}_1$.
X, Y, Z are particular elements of $\mathbb{A}_2$.

Lambda context calculus syntax

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

It looks just like lambda-calculus with explicit substitutions.

Let $\mathsf{fv}(t)$ be defined as usual. For example:

$$\mathsf{fv}(s[a_i \mapsto t]) = (\mathsf{fv}(s) \setminus \{a_i\}) \cup \mathsf{fv}(t)$$

Levels and $\#$ (technical)

Let $\mathsf{level}(t)$ be the level of the strongest variable in t, free or bound. For example:

$$\mathsf{level}(\lambda a_i.t) = \mathsf{max}(a_i, \mathsf{level}(t))$$
$$\mathsf{level}(s[a_i \mapsto t]) = \mathsf{max}(\mathsf{level}(s), a_i, \mathsf{level}(t))$$

Finally if S is a set of variables write $a_i \# S$ when

- $a_i \notin S$ and
- $i \geq k$ for every $c_k \in S$.

For example $a_i \# \{b_i\}$ but not $a_i \# \{b_j\}$ if $j > i$.

The following reduction rules took me (and then Stéphane) about three years to find; perhaps two. I lost count. They're not terribly hard.

(β)	$(\lambda a_i.s)t \longrightarrow s[a_i \mapsto t]$	
$(\sigma\mathbf{a})$	$a_i[a_i \mapsto t] \longrightarrow t$	
$(\sigma\mathsf{fv})$	$s[a_i \mapsto t] \longrightarrow s$	$a_i \# \mathsf{fv}(s)$
$(\sigma\mathbf{p})$	$(ss')[a_i \mapsto t] \longrightarrow (s[a_i \mapsto t])(s'[a_i \mapsto t])$	$\mathsf{level}(s, s', t) \leq i$
$(\sigma\sigma)$	$s[a_i \mapsto t][b_j \mapsto u] \longrightarrow s[b_j \mapsto u][a_i \mapsto t[b_j \mapsto u]]$	$i < j$
$(\sigma\lambda)$	$(\lambda a_i.s)[b_j \mapsto u] \longrightarrow \lambda a_i.(s[b_j \mapsto u])$	$i < j$
$(\sigma\lambda')$	$(\lambda a_i.s)[c_i \mapsto u] \longrightarrow \lambda a_i.(s[c_i \mapsto u])$	$a_i \# \mathsf{fv}(u)$

Example reductions

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

t ranges over any term.

- $x[X{\mapsto}t] \stackrel{(\sigma\mathsf{fv})}{\longrightarrow} x$, since $X\#\{x\}$.
- $y[x{\mapsto}t] \stackrel{(\sigma\mathsf{fv})}{\longrightarrow} y$, since $x\#\{y\}$.
- $x[x{\mapsto}t] \stackrel{(\sigma\mathbf{a})}{\longrightarrow} t$.

$X[x{\mapsto}t]$ will not reduce with $(\sigma\mathsf{fv})$ (or any other rule) since $x\#\{X\}$ does not hold.

Strong variables distributing under weak ones

$$X[x \mapsto t][X \mapsto x] \xrightarrow{(\sigma\sigma)} X[X \mapsto x][x \mapsto t[X \mapsto x]]$$

$$\xrightarrow{(\sigma\mathbf{a})} x[x \mapsto t[X \mapsto x]]$$

$$\xrightarrow{(\sigma\mathbf{a})} t[X \mapsto x]$$

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

So we have our model of instantiation.

Why are the rules the way they are?

Why (σfv)? We need (σfv) for confluence: substitutions don't always distribute over applications because of the side-condition on $(\sigma\mathbf{p})$.

Why the side-condition on $(\sigma\mathbf{p})$? Any weakening of it we've considered so far, breaks confluence.

α-equivalence is interesting. The correct notion of α-equivalence is such that $\lambda a_i.s =_\alpha \lambda a'_i.(a'_i\ a_i)s$ if $a'_i \# s$.

E.g. $\lambda x.X = \lambda y.(y\ x)X$ if $x \# X$.

This makes things complicated so in the LamCC we approximate it; we can derive $\lambda x.\lambda y.xy = \lambda x'.\lambda y'.x'y'$ but not $\lambda x.X = \lambda y.X$.

The LamCC tries to be simple.

Why infinitely many levels?

Are weak and strong variables always enough; x and X?

One-and-a-halfth-order logic (Gabbay and Mathijssen 2007) does that; it's a variant of first-order logic with predicate unknowns.

But the infinite hierarchy gives useful power.

For example $[X{\mapsto}t]$ is not a term but $\lambda\mathcal{W}.\mathcal{W}[X{\mapsto}t]$ where $\mathcal{W} \in \mathbb{A}_3$, is a term and:

$$(\lambda\mathcal{W}.\mathcal{W}[X{\mapsto}t])s \stackrel{(\beta)}{\longrightarrow} \mathcal{W}[X{\mapsto}t][\mathcal{W}{\mapsto}s]$$

$$\stackrel{(\sigma\sigma)}{\longrightarrow} \mathcal{W}[\mathcal{W}{\mapsto}s][X{\mapsto}t[\mathcal{W}{\mapsto}s]] \stackrel{(\sigma\mathsf{fv})}{\longrightarrow} \mathcal{W}[\mathcal{W}{\mapsto}s][X{\mapsto}t] \stackrel{(\sigma\mathbf{a})}{\longrightarrow} s[X{\mapsto}t].$$

What’s the relation to . . .

New calculus of contexts — same idea and superficially similar syntax, but the LamCC does pretty much the same thing and it’s a lot simpler.

Applications

Incomplete λ-terms, incomplete proofs, that kind of thing.

The LamCC represents instantiation. It requires no special apparatus — e.g. labelling strong variables with weak variables they are allowed to depend on, or raising and lifting operators a la de Bruijn. In my opinion that's a plus.

How well does this help us model/program on/reason about contexts?

Applications

Denotations. I count this as an application.

What new denotations are needed to model instantiation?

Applications

Pattern calculi, logic variables, OO languages, Glasgow Parallel Haskell; can they be usefully compiled into LamCC?

Does instantiation give useful flexibility? We can build λ-terms top-down then dynamically link arguments to the λ-abstractions bottom-up, at run-time.

Extensions of LamCC

Comparison of variables for intensional equality.

At the meta-level (say, level 2) we can compare x and y for intensional equality. I think that we can add an intensional equality to the LamCC.

It's just a constant $==_1$ that doesn't commute with $a_1{\mapsto}t]$.
$x ==_1 y \longrightarrow False$.

Higher-order logic in the LamCC.

Differs from 'ordinary' higher-order logic because we can express instantiation, so we can directly reason . . . on contexts. I'd like to be able to write, for example

$$\forall P.(a\#P \Rightarrow ((\forall a.P) \Leftrightarrow P))$$