a-logic, the λ-calculus, and internalising **meta** using names and binding.

Murdoch J. Gabbay

LMU, Munich, 26/7/2005

Thanks to LMU for inviting me

There are 28 more slides until the Conclusions.

(I can always skip a couple of technical ones, depending on what interests you.)

I'll talk about how to put assertions which normally live at the meta-level, into the object-level.

This makes the object-level more expressive (of course), and also makes the assertions susceptible to object-level proof-principles (if they survive the extension).

Lots of other people do this. The distinguishing feature of my work is that I put assertions about syntax into the object-level, not assertions about the syntax's denotation

So:

- a-logic has a predicate $\mathbf{at}\ s$ which is true when s is a variable symbol.
- The NEW calculus of contexts has meta-variables as first-class data values.

(Also: Nominal Logic, Nominal Rewriting, Nominal Unification, . . .)

So... what exactly does the title mean?

'Internalise the meta-level' means 'enrich the formal language/system/programming environment with formal assertions about structure of the syntax or semantics'. For example:

- Intuitionistic logic has a standard semantics using Kripke structures. Modal logic introduces modalities, such as $\Box P$, to make assertions about the Kripke structure (such as $\Box\Box P \supset \Box P$).
- First-order logic has function symbols f which can be applied to terms, as in $f(t)$. Higher-order logic internalises this with λ, so we can write $\lambda x.fx$.
- Syntax often needs assertions of the form '$a \notin fv(t)$'. Nominal Unification and Nominal Rewriting internalise this with a judgement $a\#t$.

Nominal Techniques

Nominal techniques are a little different, because they internalise assertions about the syntax of an assertion, rather than its semantics — compare $\Box P$ and $\lambda x.fx$, which relate firmly to the semantics, and $a\#t$, which seems to relate to the syntactic structure of t.

(Historically, this has been know to provoke allergic reactions in some academics. I take this as an encouraging sign in the long term. If people saw it coming, it wouldn't be NEW.)

Why this is good

Internalising meta-levels is often useful, simply because it exposes extra structure to object-level proof-principles. (The trick is, to not lose the proof-principles we care about, in the extension.)

What is the meta-level? Quite possibly, a formal language.

So if we take a more syntactic slant on this ‘internalisation’ process, we might get nice clean proofs, because of a nice clean internalisation.

For example: the difference (in usability) between the λ-calculus $\lambda x.x$ and combinatory logic skk. Both ‘internalise’ functions to first-order algebra, but I know which one ML and Haskell are based on.

One example: a-logic

Syntax is that of First-Order Logic (FOL) enriched with a unary predicate $\mathbf{at}$ along with a derivation rule

$$\frac{(s \text{ not a variable symbol})}{\Gamma, \mathbf{at}\, s \vdash \Delta}(\mathbf{at}\, L) \qquad \frac{\Gamma, \mathbf{at}\, a \vdash \Delta}{\Gamma, \Delta}(Fresh)$$

(In $(Fresh)$, $a \notin \Gamma, \Delta$.)

Intuitively, $\neg\mathbf{at}\, s$ says 's is a term'.

Note that if s is a term, so is $s[a{\mapsto}u]$ (s with a replaced by u).
Therefore, the Substitution Lemma still holds

$$\Gamma \vdash \Delta \text{ derivable implies } \Gamma[a{\mapsto}u] \vdash \Delta[a{\mapsto}u] \text{ is derivable.}$$

Theorem: Cut-elimination $\checkmark$

Lemma: $(\mathbf{at}\, L)$ equivalent to $\forall\overline{x}.\ \neg\mathbf{at}\, f\overline{x}$ for each term-former f.
$(Fresh)$ is just $\exists a.\ \mathbf{at}\, a$. So hey, it really easy.

Applying a-logic: axiomatisation of substitution

$a\#u$ is sugar for $\mathbf{at}\, a \wedge \forall x.\, u\langle a \mapsto x\rangle = u$

$$\mathbf{at}\, a \supset u\langle a \mapsto a\rangle = u \qquad \mathbf{at}\, a \supset a\langle a \mapsto x\rangle = x$$

$$\mathbf{at}\, a \wedge \mathbf{at}\, b \supset (a \neq b \Leftrightarrow a\#b)$$

$$\mathbf{at}\, a \wedge b\#u \supset u\langle a \mapsto b\rangle\langle b \mapsto y\rangle = u\langle a \mapsto y\rangle$$

$$\mathbf{at}\, a \wedge a\#x \supset a\#u\langle a \mapsto x\rangle$$

$$\mathbf{at}\, b \wedge a\#b \wedge a\#y \supset u\langle a \mapsto x\rangle\langle b \mapsto y\rangle = u\langle b \mapsto y\rangle\langle a \mapsto x\langle b \mapsto y\rangle\rangle$$

Provably closed: $\bullet s$

Write

$$\bullet s \quad \text{for} \quad \forall a.\, \mathbf{at}\, a \supset a\#s$$

This says 's is (provably) closed'.

$\bullet s$ does not imply that s is actually closed viewed as syntax.

For example, $\bullet x \;\vdash\; \bullet x$ ('from $\bullet x$ we may derive $\bullet x$) but x is a variable.

Similarly, if $\bullet x$ then $\forall a.\, \mathbf{at}\, a \supset \bullet(a\langle a \mapsto x\rangle)$, but $a\langle a \mapsto x\rangle$ clearly has free variables a and x.

$$
\begin{array}{lrl}
(\alpha) & \forall a,b,x,y. & \mathbf{at}\, a \wedge b\#x \supset \lambda a.x = \lambda b.x\langle a{\mapsto}b\rangle \\
(\beta) & \forall a,x,y. & \mathbf{at}\, a \wedge \bullet y \supset (\lambda a.x)y = x\langle a{\mapsto}y\rangle \\
(\xi) & \forall x,y. & \bullet x \wedge \bullet y \\
 & & \qquad \supset (\forall z.\, \bullet z \supset xz = yz) \supset x = y \\
(\sigma\lambda) & \forall a,b,x,y. & \mathbf{at}\, b \wedge a\#y \supset (\lambda a.x)\langle b{\mapsto}y\rangle = \lambda a.(x\langle b{\mapsto}y\rangle) \\
(\sigma app) & \forall a,x,y,z. & \mathbf{at}\, a \supset (xy)\langle a{\mapsto}z\rangle = (x\langle a{\mapsto}z\rangle)(y\langle a{\mapsto}z\rangle) \\
(\#app) & \forall x,y.\, \exists a. & a\#x \wedge a\#y \\
 & & \qquad \wedge \forall b.\, (b\#axy \Leftrightarrow (b\#a \wedge b\#x \wedge b\#y))
\end{array}
$$

(Recall that $a\#x$ implies $\mathbf{at}\ a$.)

This axiomatisation is correct in a suitable formal sense:

Lemma: From a FOL model of alogic + sub + lambda we obtain a λ-model by taking $\{p \mid \bullet p\}$.

Lemma: An extensional λ-model can be canonically extended to a model of lambda.

(An 'extensional λ-model' is a model of the FOL theory

$$\forall x, y.\ \mathsf{k}xy = x \qquad \mathsf{s}xyz = (xy)(xz)$$
$$\forall x, y.\ (\forall z.\ xz = yz) \supset x = y. \quad)$$

Corollary: lambda is consistent and has non-trivial models. Provably closed terms up to provable equivalence are models of the λ-calculus.

Other possible theories: internalising meta-variables

By tweaking the axioms, we can get 'friends' of the λ-calculus. Here is just one possibility:

Introduce a binary predicate $\leq$ and axioms:

$$\mathbf{at}\, a \wedge \mathbf{at}\, b \wedge a{\leq}b \wedge b{\leq}a \;\supset\; a{=}b$$

$$\mathbf{at}\, a \wedge \mathbf{at}\, b \wedge \mathbf{at}\, c \wedge a{\leq}b \wedge b{\leq}c \;\supset\; a{\leq}c \qquad \mathbf{at}\, a \supset a{\leq}a.$$

Write $a < b$ for $a \leq b \wedge a \neq b$.

Other possible theories: internalising meta-variables

Now change the axioms above as follows:

$$\mathbf{at}\, b \wedge a\#b \wedge a\#y \wedge b \leq a \supset u\langle a\mapsto x\rangle\langle b\mapsto y\rangle = s\langle b\mapsto y\rangle\langle a\mapsto x\langle b\mapsto y\rangle\rangle$$

$$\mathbf{at}\, b \wedge \mathbf{at}\, b \wedge a < b \supset u\langle a\mapsto x\rangle\langle b\mapsto y\rangle = s\langle b\mapsto y\rangle\langle a\mapsto x\langle b\mapsto y\rangle\rangle$$

$$\mathbf{at}\, b \wedge a\#y \wedge b \leq a \supset (\lambda a.x)\langle b\mapsto y\rangle = \lambda a.(x\langle b\mapsto y\rangle)$$

$$\mathbf{at}\, b \wedge \mathbf{at}\, a \wedge a < b \supset (\lambda a.x)\langle b\mapsto y\rangle = \lambda a.(x\langle b\mapsto y\rangle)$$

Hey presto, $a < b$ means 'b is a meta-variable' (with respect to a).

The NEW calculus of contexts (NewCC)

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

[-] 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!

$C[\text{-}]$ 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

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

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

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.

All this 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 $a_i, b_i, c_i, n_i, \ldots$ 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

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:

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

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

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

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

In-place update as a term

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

...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:

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

(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?

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

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.

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

Then

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

Good! Apply this to l and we get X.

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

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

The future

I see a fundamental change in computer science.

We're much more interested in bits, and their connections, and how these connections change when the bits move around.

This is for two reasons:

- As the problems/programs get bigger, we slice them into interconnected bits, solve the bits, then put them back together.
- The modern computing landscape is inherently component-based, because it's networked.

We need to develop a new mathematics to describe these things.

Modern computer science is full of clues about what that mathematics should be.

Names and binding are one of them, because people use syntax to describe stuff, names to describe their interrelations, substitution to move things around, and binding to make them local.

So, this turns up in OO programming. I didn't set out to model OO programming in the NewCC — but once I'd internalised meta-variables, it happened anyway. I don't think this is a coincidence.

There are lots of systems out there describing components and their connections, right now.

Important: I'm not going to 'solve OO programming', or 'solve mobile processes', or 'solve security'. That's obviously too much. But they're trying to tell us something and if we are to make mathematics that lasts, we need to listen.

Case study (influence of geology on higher-order logic)

Consider Higher-Order Logic. Thirty years ago, the extra power over FOL was often used for logical purposes, e.g. impredicativity, least fixedpoints, axiomatising arithmetic.

Nowadays, as likely as not we will use β-reduction to 'pipe' arguments around, and λ-abstraction to wrap up the components. Pfenning, Miller, Hofmann (tried to go back to semantics with 'a semantical analysis of HOAS'), ...

Type systems control resources, garbage collection, modules, state. Completely different from 'simple types'.

This suggests that higher-order logic (and techniques in general) have been exposed to a 'geological shift'.

Names and binding

Names and binding are a (one) distillation of something that's happening across a broad front in many different areas of mathematics.

Substitution becomes a model of 'plugging things together'. Binding is the interface. It is possible to define different substitutions, and different bindings, tailored to different situations.

Much more is possible.

Names and binding

What will I do? Add assertions about syntax to object-level systems.
People haven't done that before.

Why do that? To set about modelling a broad trend in computer science.
Get it right, and we could make a big difference.

And this is what mathematicians are supposed to do.

The field is wide open.

There are interesting problems around.

Thanks for listening.

Asking questions, are we?

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

Graphs

Here is a fun NEW calculus of contexts program:

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

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

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. Free variables behave like dangling ‘edges’.