Hierarchical Nominal Rewriting

Murdoch J Gabbay, Heriot Watt University, UK
Kindly presented by James Cheney

LFMTP, Seattle USA, 16 August 2006

Thanks. . .

. . . to the workshop organisers.

. . . to James for agreeing to present this work. I'm sorry I can't be here with you but I'm on e-mail at:

`murdoch.gabbay@gmail.com`

Take it away, James. . .

The meta-level

The meta-level is a wonderful thing.

No matter how much formal reasoning we may do in predicate logic, and no matter how much programming we may do in lambda calculus, two things cannot be avoided:

- We must specify that logic or calculus before we begin.
- We must make a meta-level judgement about the result of our calculations, when we end.

Example: first-order predicate logic

Fix $a, b, c, \ldots$ a countably infinite set of atoms (variable symbols). Then predicates are inductively defined by:

$$P, Q = (a = b) \quad | \quad (a = a) \quad | \quad P \Rightarrow P \quad | \quad \bot \quad | \quad \forall a.P$$

We equate predicates up to α-equivalence in the usual way.

Example: first-order predicate logic

Let Φ and Ψ range over finite sets of predicates and call these contexts. Call a pair $\Phi \vdash \Psi$ a sequent. Then the derivable sequents are inductively specified by:

$$\frac{}{P, \Phi \vdash \Psi, P} \qquad \frac{P, \Phi \vdash \Psi, Q}{\Phi \vdash \Psi, P \Rightarrow Q} \qquad \frac{Q, \Phi \vdash \Psi \qquad \Phi \vdash \Psi, P}{P \Rightarrow Q, \Phi \vdash \Psi}$$

$$\frac{\Phi \vdash \Psi, P}{\Phi \vdash \Psi, \forall a.P} \ (a \text{ fresh for } \Phi \text{ and } \Psi) \qquad \frac{P[a \mapsto a'], \Phi \vdash \Psi}{\forall a.P, \Phi \vdash \Psi}$$

$$\frac{P[a \mapsto a'], \Phi \vdash \Psi}{a' = a'', P[a \mapsto a''], \Phi \vdash \Psi} \qquad \frac{}{\Phi \vdash \Psi, a = a}$$

Example: the meta-level in specification

What are P and Q?

Meta-variables ranging over predicates.

What are a, b, a', a'', a'''?

Meta-variables ranging over atoms.

What is $P[a \mapsto b]$?

A meta-level operation defined which given a real predicate and two atoms, gives a predicate.

What is 'a fresh for Φ and Ψ'?

A meta-level condition only defined when given a real context and cocontext.

Example: making judgements

A sequent $\Phi \vdash \Psi$ is just syntax. It acquires meaning when we make the judgement '$\Phi \vdash \Psi$ is/is not derivable'.

Mathematical writing is replete with such meta-level reasoning. Our claim in this work is: the meta-level has real mathematical content.

Just like we can go from counting to number theory, so we can go from specifying formal syntax (like the valid sequents above) to . . . something else.

Hierarchical nominal rewriting tries to capture some part of that 'something else'.

Infinite hierarchy

But now we have a problem. Suppose we capture the meta-level in some way. We still need another meta-level in order to reason about it.

So is a mathematics of the meta-level impossible?

Must the logic we reason in (for example first-order logic) look different from the semi-formal reasoning system we use to make judgements in (such as the specification of the derivation rules)?

No!

It suffices to consider a system with an infinite hierarchy of increasingly ‘strong’ atoms (think: variable symbols).

Let us explain . . .

Types vs hierarchical nominal rewriting

In some sense type theory is already a theory of the meta-level. A function f from a type A to a type B can be thought of as an element of B with an A-shaped hole in it, to be filled by function application.

The infinite hierarchy of types plays the rôle of an infinite hierarchy of increasingly strong meta-levels. For example a function from a type C to the type $A \to B$ of functions from A to B, is like 'an f with a C-shaped hole in it'.

Hierarchical nominal rewriting replaces the hierarchy of types with a hierarchy of 'stronger' or 'weaker' atoms. Strong atoms behave like

- unknowns towards weak atoms, and like
- constant symbols towards even stronger atoms.

Syntax of hierarchical nominal terms

Fix a set of term-formers f. For each number $i \geq 1$ fix disjoint countably infinite sets of atoms $a_i, b_i, c_i, \ldots$. Say that a_i has level i.

The syntax of hierarchical nominal terms is inductively defined by

$$t ::= a_i \quad | \quad X \quad | \quad [a_i]t \quad | \quad \mathsf{f}(t_1, \ldots, t_n).$$

Syntax of hierarchical nominal terms

- Call a_i an atom of level i. It is a hole which behaves like a variable to c_k for $k < i$, and like a constant to b_j for $i < j$.
- $[a_i]t$ is an abstraction. This is t with the atom a_i abstracted.
  $[a_i]b_j$ looks like $\forall a.\phi$ (ϕ might mention a).
  $[a_i]c_k$ looks like $\forall a.(0 = 0)$ (0 cannot mention a).
- $\mathsf{f}(t_1, \ldots, t_n)$ is a term-former applied to some terms. We use this to create interesting classes of terms to rewrite. The t_i may be atoms, abstractions, or other term-formers applied to terms.

λ and $\forall$ are example term-formers, but also $+$ and 2, and $\lambda[a_2]c_1$, $\lambda[a_2]b_3$, $2 + 2$, and $a_1 + 2$, are valid hierarchical nominal terms.

Syntax of hierarchical nominal terms

X is an unknown.

It is a true variable symbol and represents an unknown term. Its behaviour is similar to that of an atom of level ω.

Because we want to do rewriting, we still need X.

The moral of the story is: we are not trying to eliminate the meta-level.

But we are trying to make mathematics out of it.

Some conventions

i, j, k will vary over nonzero natural numbers.

a_i, b_i, c_i range permutatively over atoms of level i.

That is, $a_i \neq b_j$ necessarily. a_i and b_i represent two distinct atoms of the same level.

Unless stated otherwise we may assume $k \leq i < j$.

Freshness assertions

It is typical to have conditions such as 'a fresh for Φ and Ψ'.

We must represent this aspect of the meta-level too, in hierarchical nominal rewriting, so that we can express capture-avoidance conditions.

Call a pair $a_i\#t$ a freshness assertion. The intuition is 'a_i is not free in t' ('$a \notin fv(t)$').

Freshness assertions

Q. When is $a_i \# b_i$?

A. Always. ($x \notin fv(y)$ is always true)

Q. When is $a_i \# c_k$?

A. Always. ($x \notin fv(0)$ is always true)

Q. When is $a_i \# a_i$?

A. Never. ($x \notin fv(x)$ is never true)

Freshness judgements

Q. When is a_i not free in b_j, for $i < j$?

A. When we say so. Because b_j is stronger than a_i, it behaves like t does with respect to a variable symbol x: it depends on what t is.

Similarly for $a\#X$. X represents an unknown term: we do not know $a\#X$, unless we assume it *da capo*.

Freshness assertions

Q. When is $a_i \# \mathsf{f}(t_1, \ldots, t_n)$?

A. When $a_i \# t_1$ and ... and $a_i \# t_n$.

(Obvious motivation from syntax here.)

Q. When is $a_i \# [a_i]t$?

A. Always.

($x \notin fv(\forall x.\phi)$, $x \notin fv(\lambda x.t)$)

Freshness assertions

Q. When is $a_i\#[b_j]t$ (for $i < j$)?

A. When $a_i\#t$ assuming $a_i\#b_j$.

This has no obvious analogue in ordinary syntax, but consider this: we expect

$$a_i\#[b_j]b_j,$$

but $a_i\#b_j$ is only true if we assume it is true.

Derivation rules for valid freshness assertions

$$\frac{k \leq i}{a_i \# c_k}\,(\#diff) \qquad \frac{a_i \# t_1 \ldots a_i \# t_n}{a_i \# \mathsf{f}(t_1, \ldots, t_n)}\,(\#\mathsf{f})$$

$$\frac{a_i \# t}{a_i \# [c_k]t}\,(\#abs{\geq})\quad (i \geq k)$$

$$\frac{}{a_i \# [a_i]t}\,(\#abs{=}) \qquad \frac{\begin{array}{c}[a_i \# b_j] \\ \vdots \\ a_i \# t\end{array}}{a_i \# [b_j]t}\,(\#abs{<})\quad (i < j)$$

Freshness assertions

$[a_i \# b_j]$ denotes discharge in the natural deduction sense.

In sequent style $(\#abs<)$ would be

$$\frac{\nabla, a_i \# b_j \vdash a_i \# t}{\nabla \vdash a_i \# [b_j] t}.$$

Call a freshness assertion primitive if it is of the form $a_i \# b_j$ (yes, for $i < j$). Write ∇ for a set of primitive freshness assertions and call it a (primitive) freshness context.

Hierarchical nominal rewrite rules

A hierarchical nominal rewrite rule is a triple

$$\nabla \vdash l \rightsquigarrow r$$

where ∇ is a primitive freshness context and l and r are terms such that r and ∇ mention only unknowns in l.

Hierarchical nominal rewrite rules

A hierarchical nominal term-in-context is a pair $\nabla \vdash t$.

$\Delta \vdash t \overset{(R)}{\rightsquigarrow} u$ and read it as

$\Delta \vdash t$ rewrites with (R) to u

when

- There is a position C and substitution σ such that
- $\Delta \vdash \nabla\sigma$ and
- $C[l\sigma] = t$, and $C[r\sigma] = u$.

Hierarchical nominal rewrite rules

(Substitutions map X to terms $\sigma(X)$ and act by syntactic replacement; no capture-avoidance.)

So hierarchical nominal rewriting is just like rewriting, except that rewrites may be blocked if their ∇ is not satisfied.

Hierarchical nominal rewrite rules

Write $\rightsquigarrow^*$ for the reflexive transitive closure of $\rightsquigarrow$. So $\Delta \vdash t \rightsquigarrow^* u$ holds when $t = u$ or when there is some sequence of $\rightsquigarrow$-reductions from t to u. If Δ is irrelevant or known we may write $\Delta \vdash t \rightsquigarrow^* u$ as just $t \rightsquigarrow^* u$.

Call a possibly infinite set of hierarchical nominal rewrite rules a hierarchical nominal (term) rewrite system.

Call a hierarchical nominal rewrite system locally confluent when if $\Delta \vdash t \rightsquigarrow u$ and $\Delta \vdash t \rightsquigarrow u'$, then v exists such that $\Delta \vdash u \rightsquigarrow^* v$ and $\Delta \vdash u' \rightsquigarrow^* v$.

Usual theorems hold such as: If all non-trivial critical pairs are joinable then the system is locally confluent. Details in the paper.

Example: substitution

Assume a binary term-former sub and sugar $\mathsf{sub}([a]u, t)$ to $u[a \mapsto t]$.

$$
\begin{array}{lrcll}
(suba) & a_i[a_i \mapsto X] & \rightsquigarrow & X & \\
(sub\#) & a_i \# Z \vdash Z[a_i \mapsto X] & \rightsquigarrow & Z & \\
(subaa) & Z[a_i \mapsto a_i] & \rightsquigarrow & Z & \\
(sub\mathsf{f}) & \mathsf{f}(Z_1, \ldots, Z_n)[a_i \mapsto X] & \rightsquigarrow & \mathsf{f}(Z_1[a_i \mapsto X], \ldots, Z_n[a_i \mapsto X]) & \\
(subabs{>}) & ([c_k]Z)[a_i \mapsto X] & \rightsquigarrow & [c_k](Z[a_i \mapsto X]) & (i > k) \\
(subabs{\leq}) & b_j \# X \vdash ([b_j]Z)[a_i \mapsto X] & \rightsquigarrow & [b_j](Z[a_i \mapsto X]) & (i \leq j)
\end{array}
$$

These are axiom-schemes for all i and j and every n, and for every term-former f.

Example: substitution

Note $(subabs{\leq})$ and $(subabs{>})$.

We see there that substitution for strong variables does not avoid capture for abstraction by weak variables. This is the behaviour of meta-variables. For example when we write:

Let ϕ be $a = a$ in $\forall a.\phi$

— we get $\forall a.(a = a)$. Substitution for weak variables does avoid capture for abstraction by strong variables (or variables of the same strength). For example when we write:

Let b be $a = a$ in $\forall a.(b = b)$

— we get $\forall a'.(a = a)$.

Example: NEW

Assume a term-former И. Sugar $И[a_i]t$ to $Иa_i.\ t$.

$$(И\#) \qquad b_j\#Z \vdash Иb_j.\ Z \rightsquigarrow Z$$

$$(Исub) \quad b_j\#Y \vdash И(([b_j]Z)[a_i{\mapsto}Y]) \rightsquigarrow И([b_j](Z[a_i{\mapsto}Y]))$$

This behaves much like the π-calculus ν name-restrictor. By wrapping an abstraction in И we guarantee capture-avoidance even if the substitution is for a very strong atom. Thus we recover the usual notion of locality (which avoids capture and knows nothing of hierarchies of atoms) by introducing a special term-former И.

One more example: permutation

Write $(a_i\ b_i)_{c_i} \cdot t$ for $И c_i.\ t[a_i \mapsto c_i][b_i \mapsto a_i][c_i \mapsto b_i]$.

Suppose $c_i \# X$ and $d_i \# X$. Then:

$$(a_i\ b_i)_{c_i} \cdot X \rightsquigarrow^* (a_i\ b_i)_{d_i} \cdot X \qquad (a_i\ b_i)_{c_i} \cdot X \rightsquigarrow^* (b_i\ a_i)_{c_i} \cdot X$$

$$(a_i\ b_i)_{c_i} \cdot (a_i\ b_i)_{c_i} \cdot X \rightsquigarrow^* X.$$

So we may write $(a_i\ b_i) \cdot t$ for $(a_i\ b_i)_{c_i} \cdot t$.

This is a permutation action with the effect of a 'simultaneous substitution of a_i for b_i and vice versa', and this can be made formal by certain rewrites (in the paper).

Simultaneous substitution may be encoded similarly.

Conclusions

Hierarchical nominal rewriting introduces a hierarchy of atoms into a nominal rewriting-like framework.

This is another step towards a mathematics of the meta-level along a path first taken by nominal unification and rewriting.

It is possible to give formal rewrite rules for substitution, which exhibit properties both of instantiation (non-capture-avoiding substitution) and substitution (capture-avoiding substitution), depending on the relationships between levels.

It is possible to give rewrite rules for scoping term-formers. In combination with substitution this is quite powerful.

Conclusions

Much more is possible. Current work is an abstract rewriting semantics for Glasgow Parallel Haskell with Phil Trinder. It is easier to prove equivalences of programs in the rewriting framework than the GPH abstract machine.

The NEW calculus of contexts [PPDP'04] is a lambda-calculus with a hierarchy of variable symbols which is based on similar ideas.

Conclusions

By separating atoms into a hierarchy we give a new account of the 'ϕ' and 't' which pervade mathematical practice. With the hierarchy, we have a choice of approaching atoms

- 'from below', so they behave like variables, or
- 'from above', so we can program on them as data,

as we find convenient.

There is some philosophical content to merging capture-avoiding and non-capture-avoiding substitution. We hope to construct new programming languages and logics which are closer to informal mathematical practice and which can program on their own parameters in new and interesting ways.