Nominal rewriting

Murdoch J. Gabbay

23/1/2006, Innsbruck, Austria

Thanks for inviting me (at short notice).

I'll talk about nominal rewriting. . .

. . . and the broader framework of my research, if I have time.

The issue

Consider the term $\lambda x.t$.

x is a variable symbol and t is a meta-level variable, ranging over λ-terms.

Instantiation of t does not avoid capture: if we set t to be x, we get $\lambda x.x$.

The issue

Consider the term $(\lambda x.t)u$.

This reduces

$$(\lambda x.t)u \rightsquigarrow t[x \mapsto u]$$

Let's specify how substitution distributes through t:

$$\begin{aligned}
x[x \mapsto t] &= t \\
y[x \mapsto t] &= y \\
(tt')[x \mapsto u] &= (t[x \mapsto u])(t'[x \mapsto u]) \\
(\lambda z.t)[x \mapsto u] &= \lambda z.(t[x \mapsto u]) && z \notin u
\end{aligned}$$

The issue

x, y, and z are variable symbols, or more precisely meta-level variable symbols varying over object-level variable symbols.

t and u are meta-level variable, ranging over λ-terms.

t itself is not a λ-term!

Instantiation of t does not avoid capture: if we set t to be x, we get $\lambda x.x$.

The issue

The definition of substitution has side-conditions (so as a rewrite system we would need conditional reductions:

$$(\lambda z.t)[x \mapsto u] = \lambda z.(t[x \mapsto u]) \qquad\qquad z \notin u$$

The issue

Substitution of ‘strong’ (meta-level; t) variables for ‘weak’ (object-level; x) variables does not avoid capture.

Substitution of variables of the same level does avoid capture. That’s what we specify when we ‘specify substitution’ $[x \mapsto u]$.

Nominal rewriting is a rewriting framework which faithfully represents the intuition and informal practice of writing $\lambda x.t$, including the capturing behaviour of instantiation of t.

Syntax and sorts

Nominal rewriting has nominal terms.

It is abstract syntax trees, with sorts and term-formers.

$$t, u ::= a, b, c, \ldots \mid X, Y, Z, \ldots \mid [a]t \mid \mathsf{f}(t, \ldots, t) \mid \ldots$$

$a, b, c, \ldots$ are atoms. They represent object-level variable symbols. They have a sort of ... 'object-level variable symbols'. So object-level variable symbols are data.

$X, Y, Z, \ldots$ are variables or unknowns. They represent unknowns and may have any sort (usually elided).

$[a]t$ is an abstraction. Think of it as $\lambda a.t$, but without β-equivalence.

Sorts for the λ-calculus

Take a sort $\mathbb{T}$ of λ-terms and a sort $\mathbb{A}$ of atoms.

Note: we represent the terms of the λ-calculus as nominal terms of sort $\mathbb{T}$.

Nominal rewrite system for the λ-calculus

Take $\cdot$ (application) a binary term-former arity $(\mathbb{T}, \mathbb{T})\mathbb{T}$.

Write $\cdot(t, u)$ as tu and associate to the left, as usual.

Take λ (abstraction) arity $([\mathbb{A}]\mathbb{T})\mathbb{T}$.

Write $\lambda([a]t)$ as $\lambda[a]t$.

Take sub (explicit substitution) arity $([\mathbb{A}]\mathbb{T}, \mathbb{T})\mathbb{T}$.

Write $\mathsf{sub}([a]t, u)$ as $t[a{\mapsto}u]$.

Nominal rewrite system for the λ-calculus

Rewrite rules are:

$$(\lambda[a]X)Y \;\rightarrow\; X[a \mapsto Y] \qquad\qquad \big(\cdot(\lambda[a]X, Y) \;\rightarrow\; \mathsf{sub}([a]X, Y)\big)$$

and...

Explicit substitution

$$
\begin{array}{rcl}
a[a\mapsto X] & \to & X \\
a\#Z \vdash Z[a\mapsto X] & \to & Z \\
\mathsf{f}(X_1,\ldots,X_n)[a\mapsto X] & \to & \mathsf{f}(X_1[a\mapsto X],\ldots,X_n[a\mapsto X]) \\
b\#X \vdash ([b]Y)[a\mapsto X] & \to & [b](Y[a\mapsto X])
\end{array}
$$

For example:

$$(\lambda[a]a)b \;\longrightarrow\; a[a\mapsto b] \;\longrightarrow\; b$$

$$(\lambda[a]aab)b \;\longrightarrow\; (aab)[a\mapsto b] \;\longrightarrow\; (aa)[a\mapsto b](b[a\mapsto b]) \;\longrightarrow^{*}\; bbb$$

For example:

$$(\lambda[a]\lambda[b]a)b \rightarrow (\lambda[b]a)[a\mapsto b] \rightarrow \lambda(([b']a)[a\mapsto b]) \stackrel{b'\#b}{\rightarrow} \lambda[b'](a[a\mapsto b]) \rightarrow \lambda[b']b$$

$$(\lambda[a]\lambda[b]Z)X \rightarrow (\lambda[b]Z)[a\mapsto X] \rightarrow \lambda(([b'](b'\ b)\cdot Z)[a\mapsto X]) \stackrel{b'\#X,Z}{\rightarrow} \lambda[b']((b'\ b)\cdot Z[a\mapsto X]).$$

If we also know $a\#Z$ we can further reduce

$$\lambda[b']((b'\ b)\cdot Z[a\mapsto X]) \rightarrow \lambda[b'](b'\ b)\cdot Z.$$

α-equality and freshness

What is $a\#t$?

$$\frac{a\#t_1 \quad \cdots \quad a\#t_n}{a\#\mathsf{f}(s_1,\ldots,t_n)} \qquad \frac{a\#t}{a\#[b]t} \qquad \frac{}{a\#b} \qquad \frac{}{a\#[a]t} \qquad \frac{\pi^{\text{-}1}(a)\#X}{a\#\pi\cdot X}$$

$a\#[a]t$ always holds.

$a\#X$ only holds if you've assumed . . . $a\#X$.

$b\#a$ always holds.

$a\#a$ never holds.

$a\#\pi\cdot X$ holds if and only if $\pi^{\text{-}1}(a)\#X$ holds.

α-equality and freshness

What is $(a\ b) \cdot X$?

Well, note that it is not possible for $[a]X \approx_\alpha [b]X$.

Then (since rewrites and thus equality should be closed under instantiating unknowns) $[a]a \approx_\alpha [b]a$, which is like $\lambda a.a = \lambda b.a$ (but without the functions, i.e. β-equivalence!).

But we still want to rename atoms, to avoid capture, etc.

So we write $[a]X \approx_\alpha [b](b\ a) \cdot X$.

Nominal rewriting is such that rewrites are equivalent up to the least symmetric transitive reflexive congruence $\approx_\alpha$ such that

$$a, b\#t \ \vdash \ (a\ b) \cdot t \approx_\alpha t.$$

α-equality and freshness

$\approx_\alpha$ is decidable, in linear time:

$$\frac{s_1 \approx_\alpha t_1 \quad \cdots \quad s_n \approx_\alpha t_n}{\mathsf{f}(s_1,\ldots,s_n) \approx_\alpha \mathsf{f}(t_1,\ldots,t_n)} \qquad \frac{}{a \approx_\alpha a} \qquad \frac{t \approx_\alpha t'}{t' \approx_\alpha t}$$

$$\frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \qquad \frac{a\#t \qquad s \approx_\alpha (a\ b)\cdot t}{[a]s \approx_\alpha [b]t} \qquad \frac{ds(\pi,\pi')\#X}{\pi\cdot X \approx_\alpha \pi'\cdot X}$$

(Here $ds(\pi,\pi') \stackrel{\text{def}}{=} \{n \mid \pi(n) \neq \pi'(n)\}$. For example, $ds((a\ b),\mathbf{Id}) = \{a,b\}$.)

Example derivation

$$\dfrac{\dfrac{}{a\#\lambda[a]ba} \qquad \dfrac{\dfrac{a \approx_\alpha a \quad b \approx_\alpha b}{ab \approx_\alpha ab}}{\lambda[b]ab \approx_\alpha (b\ a)\cdot(\lambda[a]ba) \equiv \lambda[b]ab}}{\lambda[a]\lambda[b]ab \approx_\alpha \lambda[b]\lambda[a]ba}$$

Looks like $\lambda f.\lambda x.fx = \lambda x.\lambda f.xf$.

Example derivation

$$\dfrac{\dfrac{}{a\#\lambda[a](b\;a)\cdot X} \qquad \dfrac{\dfrac{}{X \approx_\alpha (b\;a)\circ(b\;a)\cdot X}\,(\#X)}{\lambda[b]X \approx_\alpha (b\;a)\cdot(\lambda[a](b\;a)\cdot X) \equiv \lambda[b](b\;a)\circ(b\;a)\cdot X}}{\lambda[a]\lambda[b]X \approx_\alpha \lambda[b]\lambda[a](b\;a)\cdot X}$$

Looks like?

Note permutation treats open terms (terms with unknowns). Parametric treatment of abstraction.

Global context

Nominal rewriting [PPDP'04] is like first-order rewriting:

If nontrivial critical pairs are joinable: local confluence.

Orthogonal rewrite system: confluence.

Interesting extensions [PPDP'05] as rewrite system.

Equality

Instead of considering $\longrightarrow$, a directed equality. . .

. . . we can throw out the direction and consider nominal algebra (Nominal Algebraic Specifications).

Substitution (again)

$$
\begin{array}{rrcll}
(\#\mapsto) & a\#X & \vdash & X[a\mapsto T] & = X \\
(f\mapsto) & & \vdash & \mathsf{f}(X_1,\ldots,X_n)[a\mapsto T] & = \mathsf{f}(X_1[a\mapsto T],\ldots,X_n[a\mapsto T]) \\
(abs\mapsto) & b\#T & \vdash & ([b]X)[a\mapsto T] & = [b](X[a\mapsto T]) \\
(var\mapsto) & & \vdash & \mathsf{var}(a)[a\mapsto T] & = T \\
(ren\mapsto) & b\#X & \vdash & X[a\mapsto \mathsf{var}(b)] & = (b\ a)\cdot X
\end{array}
$$

(var has sort $(\mathbb{A})\mathbb{T}$.)

These axioms are ω-complete — if $t\sigma = u\sigma$ for all closing σ then $t = u$.

This is not at all an easy result.

Logic (first-order)

(Props) $$P \Rightarrow Q \Rightarrow P \;=\; \top \qquad \neg\neg P \Rightarrow P \;=\; \top$$

$$(P \Rightarrow Q) \Rightarrow (Q \Rightarrow R) \Rightarrow (P \Rightarrow R) \;=\; \top \qquad \bot \Rightarrow P \;=\; \top$$

(Quants)

$$\forall[a]P \Rightarrow P[a{\mapsto}T] \;=\; \top \qquad \forall[a](P \wedge Q) \Leftrightarrow \forall[a]P \wedge \forall[a]Q \;=\; \top$$

$$a\#P \;\vdash\; \forall[a](P \Rightarrow Q) \Leftrightarrow P \Rightarrow \forall[a]Q \;=\; \top$$

(Eq) $$T \approx T = \top \qquad T \approx U \Rightarrow P[a{\mapsto}T] \Leftrightarrow P[a{\mapsto}U] = \top$$

Further work

Atoms are data. That is, $a \neq b$ is derivable.

So in a semantics, e.g. for substitution or logic, variable symbols are first-class elements of the denotation.

What does that denotation look like?

Further work

In a sense the only difference between X and a is that $([a]X)[X{\mapsto}t] \equiv [a]t$, i.e. substitution of t for X does not avoid capture.

$([a]X)[b{\mapsto}t]$ does avoid capture.

What if we allow abstraction by $[X]$ in the syntax, and introduce a hierarchy of levels of variables a_1 (a), a_2 (X), a_3 (t?), and so on, what do we get [PPDP'05b].

Further work

Graphs with abstraction for name-generation (work with Joe Wells)?

Logics and lambda-calculi with hierarchies of variables (instead of simple types)?

Feasibility study of mechanised formal proof system (like Isabelle) but with iconoclastic treatment of functions?

. . . and much more, of course.