Extended Nominal Rewriting: two models of binding

Murdoch J. Gabbay

Joint work with Maribel Fernández

LIX, Paris, 5/2005

Thanks to Dale Miller for organising this talk.

Conclusions

Nominal Rewriting internalises α-equivalence as a logical derivation.

Extended Nominal Rewriting plays with this by internalising two notions of binding and putting them side-by-side.

$$\overline{a}b \mid a[c].Y \rightarrow Y\{c\mapsto b\} \quad \nu[c]X \rightarrow \text{И}c.X \quad !X \rightarrow !X \mid X$$

$$a@P \vdash P \mid (\text{И}a.Q) \rightarrow \text{И}a.(P \mid Q).$$

$X\{c\mapsto b\}$, actually $\Sigma([c]X, b)$, is explicit substitution with reactions $(X \mid Y)\{c\mapsto b\} \rightarrow X\{c\mapsto b\} \mid Y\{c\mapsto b\}$ and $(\nu[a]X)\{c\mapsto b\} \rightarrow \nu[a](X\{c\mapsto b\})$.

And we can express reduction strategies like

$$\bullet Y \vdash (\lambda a.X)Y \rightarrow X\{a\mapsto Y\}.$$

Introduction

Rewriting is a general framework of logic and computation — just define term-formers for the syntax of your system and then reason or compute by rewriting.

Logic and computation display binding behaviour. Here are two binders:

Two binders: λ and ν

ν from the π-calculus satisfies:

- $\nu a.P_a \approx \nu b.P_b$.
- $\nu a.\nu b.P \approx \nu b.\nu a.P$.
- $\nu a.P \approx P$ (if $a \notin P$).
- $C[\nu a.P] \approx \nu a.C[P]$ if $a \notin C$ and C cannot copy P ($!$ does).

λ from the λ-calculus satisfies:

- $\lambda a.P_a \approx \lambda b.P_b$.
- $\lambda a.\lambda b.P \not\approx \lambda b.\lambda a.P$.
- $\lambda a.P \not\approx P$ (if $a \notin P$).
- $C[\lambda a.P] \not\approx \lambda a.C[P]$ unless C is the identity context.

λ

‘λa’ copies an unknown argument to as in its scope.

Position is important because that is where the application is to be attached. If the λ is moved or duplicated, applications may have an extra, a missing, or an unintended, mate.

The ‘problem of binding’ in $\lambda a.P$ is caused by interactions between internal lambdas in P; in $P[a \mapsto Q]$ names in Q should not be captured by other binders in P.

ν

'$C[\nu a.P]$' generates a name fresh for names mentioned outside P, i.e. in the external C.

Repositioning and duplication of νa are OK; so long as the as are fresh. Copying interacts with context: we do not mean

$$2(\nu a.a) \longrightarrow 2(a) \longrightarrow a \mid a$$

but

$$2(\nu a.a) \longrightarrow \nu a.a \mid \nu a.a \longrightarrow a \mid \nu a.a \longrightarrow a \mid b.$$

Extended nominal rewriting. . .

. . . provides two constructs, one to model λ and the other to model ν.

In this talk I will stress the foundational mathematical aspects. In another talk we might also want to use it — the space between λ- and π-calculi takes in quite a bit!

λ

Treated, in our style, by Nominal Rewriting [PPDP'04]. The recipe is:

1. Separate meta- and object-variables to X and a respectively.
2. Introduce a term-former $[a]X$ to model abstraction (λ).
3. Leave actual β-reduction to the user to program up (so β-reduction is *not* a structural congruence of terms).
4. Do *not* even quotient terms by α-equivalence (so $[a]a \not\equiv [b]b$).
5. *Do* introduce a binary relation $\approx$ and make it primitive, in that α-equivalent terms have the same rewrites.

In view of point 5, why have point 4? Because it separates the definition of substitution from the definition of α-equivalence.

И

So now we come back to ν. Introduce a construct $\text{И}a.P$ to mean ‘generate a fresh a and get P’.

We *do* want $\text{И}a.\text{И}a.P \approx \text{И}a.P$ and $\text{И}a.P \approx P$ if $a \notin P$. It is not convenient for α-equivalence to move term-formers up and down a term, so make И not be a term-former — annotate terms at every level with a tag of local names.

We *do* want $\text{И}a.\text{И}b.P \approx \text{И}b.\text{И}a.P$ so make the tag a set.

We *do not* want $c(\text{И}a.a, P) \approx \text{И}a.(c(a, P))$, unless we have a mechanism to commit c to not copy its first argument. Good, tag sets give us that behaviour.

И

We *do not* want $\text{И}a.a \approx \text{И}b.b$.

Hang on. $\nu a.\overline{a}a \approx \nu b.\overline{b}b$ holds, does it not? Yes, but we have abstraction for that. И just generates the name, what you do with it then is up to you.

However, extended nominal rewriting has rule F:

$$b@X \ \vdash \ \text{И}a.X \longrightarrow \text{И}b.(b\ a)X.$$

Maximum confusion point.

Syntax

Fix term variables X, Y, Z and atoms a, b, c, n, x. Write A, B for finite sets of atoms.

A swapping is a pair of atoms $(a\ b)$. Permutations π are lists of swappings

$$\pi ::= \textbf{Id} \mid (a\ b) \cdot \pi$$

Call **Id** the identity permutation. Write $\pi \cdot X$ for a swapping suspended on a variable.

Terms are

$$\begin{array}{lll} s, t & ::= & \text{И}A.u \\ u, v & ::= & a \mid \pi \cdot X \mid \langle t_1, \ldots, t_n \rangle \mid [a]t \mid (f t) \end{array}$$

A term with only empty tags is a(n unextended) nominal term. We may write $\text{И}\emptyset.u$ as u, and $\textbf{Id} \cdot X$ as X.

Substitution

$$(\text{И}A.a)[X\mapsto s] \equiv \text{И}A.a \quad (\text{И}A.(ft))[X\mapsto s] \equiv \text{И}A.(f(t[X\mapsto s]))$$

$$(\text{И}A.\langle t_1,\ldots,t_n\rangle)[X\mapsto s] \equiv \text{И}A.\langle t_1[X\mapsto s],\ldots,t_n[X\mapsto s]\rangle$$

$$(\text{И}A.[a]t)[X\mapsto s] \equiv \text{И}A.[a](t[X\mapsto s])$$

$$(\text{И}A.\pi\cdot X)[X\mapsto \text{И}B.s] \equiv \text{И}(A\cup\pi\cdot B).\pi\cdot s.$$

Swapping

$$(a\ b)\cdot \text{И}A.u \equiv \text{И}(a\ b)\cdot A.(a\ b)\cdot u$$

$$(a\ b)\cdot a \equiv b \qquad (a\ b)\cdot b \equiv a \quad \text{and} \quad (a\ b)\cdot c \equiv c\ (c \not\equiv a, b)$$

$$(a\ b)\cdot (ft) \equiv (f(a\ b)\cdot t)$$

$$(a\ b)\cdot \langle t_1,\ldots,t_n\rangle \equiv \langle (a\ b)\cdot t_1,\ldots,(a\ b)\cdot t_n\rangle$$

$$(a\ b)\cdot [n]t \equiv [(a\ b)\cdot n](a\ b)\cdot t$$

$$(a\ b)\cdot \pi\cdot X \equiv ((a\ b)\cdot \pi)\cdot X.$$

So:

$$(a\ b)\cdot \lambda[a]\text{И}b,c.abX \equiv \lambda[b]\text{И}a,c.ba(a\ b)\cdot X$$

$$\Big((a\ b)\cdot \lambda[a]\text{И}b,c.abX\Big)[X\mapsto a] \equiv \lambda[b]\text{И}a,c.bab.$$

Fresh

$$\frac{a\#u}{a\#\text{И}A.u} \qquad \frac{}{a\#b} \qquad \frac{a\#s}{a\#f\,s} \qquad \frac{a\#s_1 \quad \cdots \quad a\#s_n}{a\#\langle s_1, \ldots, s_n\rangle}$$

$$\frac{}{a\#[a]s} \qquad \frac{a\#s}{a\#[b]s} \qquad \frac{\pi^{-1} \cdot a\#X}{a\#\pi \cdot X} \qquad \frac{a\#a}{P}\,(\#\bot)$$

Local

$$\frac{}{a@\mathsf{N}A.u}\ a \in A \qquad \frac{a@u}{a@\mathsf{N}A.u}\ a \notin A$$

$$\frac{}{a@b} \qquad \frac{a@s}{a@fs} \qquad \frac{a@s}{a@[b]s} \qquad \frac{}{a@[a]s}$$

$$\frac{a@s_1 \quad \cdots \quad a@s_n}{a@\langle s_1, \ldots, s_n\rangle} \qquad \frac{\pi^{-1} \cdot a@X}{a@\pi \cdot X} \qquad \frac{a\#X}{a@X}\ (\#@) \qquad \frac{a@a}{P}\ (@\bot)$$

Alpha-equivalence

$$\frac{u \approx v \quad B\backslash A@u \quad A\backslash B@v}{\text{И}A.u \approx \text{И}B.v}$$

$$\frac{}{a \approx a} \qquad \frac{s_1 \approx t_1 \quad \cdots \quad s_n \approx t_n}{\langle s_1, \ldots, s_n \rangle \approx \langle t_1, \ldots, t_n \rangle} \qquad \frac{s \approx t}{fs \approx ft}$$

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

where

$$ds(\pi, \pi') \stackrel{\text{def}}{=} \{n \mid \pi \cdot n \neq \pi' \cdot n\}.$$

For example

For example, $ds((a\,b), \mathbf{Id}) = \{a, b\}$, so (using the rules above) we can deduce $(a\,b)\cdot X \approx X$ from assumptions $a\#X$ and $b\#X$ and we also have as expected $[a]a \approx [b]b$.

Consistent with intuitions discussed in the Introduction, $\text{И}a.b \approx b$, and $\text{И}a, b.a \approx \text{И}a.a$, but note that $\text{И}a.a \not\approx \text{И}b.b$ and $\text{И}a, a.u \equiv \text{И}a.u$.

We can deduce $\text{И}a.s \approx s$ from $a@s$.

Scope-extrusion for $|$

Suppose some binary term-former $|$. Then scope-extrusion rule for it is

$$a\#Y \vdash (\text{И}a.X) \mid Y \to \text{И}a.(X \mid Y).$$

This works in collaboration with rule F:

$$b\#X \vdash \text{И}a.X \to \text{И}b.(b\ a)X$$

which lets us guarantee $a\#t$ for any particular choice of t to replace Y.

Unification, confluence, etc.

И creates some interesting issues. Clearly with rule F no rewrite system can be terminating, so we need a notion of termination up to renaming local names. Not a problem.

Also, И may ‘swallow’ names. For example (on the face of it) $\text{И}a.X \approx \text{И}a.a$ has two solutions: X maps to a, and X maps to $\text{И}a.a$. Neither is a substitution instance of the other because substitution is for meta-variables X and Y, not for a. So we need a notion of unifier up to accidental swallowing by tags. Again, not a problem.

Alpha-equivalence

Some people say that Nominal Rewriting, in its various flavours, is ‘just about α-equivalence’.

But that misses that it’s about substitution for strong (meta-) variables and unification of incomplete terms, in the presence of α-equivalence for weak (object-) variables.

Thread the machinery of determining $\approx$ into the mechanics of the system, as we do by making it a logical derivation distinct from the fact of syntactic equivalence $\equiv$, and you have structural proof principles you can use to get results, and you have a design space e.g. to have more than one kind of binder.

We have access to a new design space. I *like* design spaces.

Closed, is-in

Actually, I've missed out something important.

We can introduce new judgements $\bullet s$ for 's is closed' or $a \in s$ for 'a occurs (precisely once?) in s'.

We can use these to express reduction strategies, but still benefit from meta-theorems of the rewriting framework such as confluence results.

That is; if your favourite reduction strategy can be expressed within the nominal rewriting framework in a certain formal sense, then you get nominal rewriting meta-theorems for free.

Conclusions

Nominal Rewriting internalises α-equivalence as a logical derivation.

Extended Nominal Rewriting plays with this by internalising two notions of binding and putting them side-by-side.

$$\overline{a}b \mid a[c].Y \rightarrow Y\{c\mapsto b\} \quad \nu[c]X \rightarrow \text{И}c.X \quad !X \rightarrow !X \mid X$$

$$a@P \vdash P \mid (\text{И}a.Q) \rightarrow \text{И}a.(P \mid Q).$$

$X\{c\mapsto b\}$, actually $\Sigma([c]X, b)$, is explicit substitution with reactions $(X \mid Y)\{c\mapsto b\} \rightarrow X\{c\mapsto b\} \mid Y\{c\mapsto b\}$ and $(\nu[a]X)\{c\mapsto b\} \rightarrow \nu[a](X\{c\mapsto b\})$.

And we can express reduction strategies like

$$\bullet Y \vdash (\lambda a.X)Y \rightarrow X\{a\mapsto Y\}.$$