

Extended Nominal Rewriting: two models of binding

Murdoch J. Gabbay

Joint work with Maribel Fernández

PPDP 2005, Lisbon, 10/7/2005

First: here are the 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.

$$\begin{aligned} \bar{a}b \mid a[c].Y &\rightarrow Y\{c\mapsto b\} & \nu[c]X &\rightarrow \forall c.X & !X &\rightarrow !X \mid X \\ a@P \vdash P \mid (\forall a.Q) &\rightarrow \forall a.(P \mid Q). \end{aligned}$$

$X\{c\mapsto b\}$, 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

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

This has been implemented by Christophe Calves.

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 (an Abstractor) and ν (a Restrictor)

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 a 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

Briefly...

‘ $C[\nu a.P]$ ’ 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 a s are fresh. Copying interacts with context: we do not mean

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

but

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

Extended nominal rewriting...

... provides two constructs, one to model ‘abstractors’ (like λ) and the other to model ‘restrictors’ (like 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!

Abstructor (think 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. β -reduction *not* a structural congruence of terms — implement with rewrites.
4. α -equivalence is also *not* a structural congruence of terms — so $[a]a \not\equiv [b]b$.
5. *However, do* introduce \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.

Restrictor (think \mathbb{N})

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

We *do* want $\mathbb{N}a.\mathbb{N}a.P \approx \mathbb{N}a.P$ and $\mathbb{N}a.P \approx P$ if $a \notin P$.

It is not convenient for α -equivalence to move term-formers up and down a term, so make \mathbb{N} not be a term-former — annotate terms at every level with a **tag** of local names.

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

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

Restrictor

We *do not* want $\forall a.a \approx \forall b.b$.

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

However, extended nominal rewriting has rule F :

$$b@X \vdash \forall a.X \rightarrow \forall b.(b\ a)X.$$

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

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

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

Terms are

$$\begin{aligned} s, t & ::= \mathbb{N}A.u \\ u, v & ::= a \mid \pi \cdot X \mid \langle t_1, \dots, t_n \rangle \mid [a]t \mid (ft) \end{aligned}$$

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

Substitution

$$(\forall A.a)[X \mapsto s] \equiv \forall A.a \quad (\forall A.(ft))[X \mapsto s] \equiv \forall A.(f(t[X \mapsto s]))$$

$$(\forall A.\langle t_1, \dots, t_n \rangle)[X \mapsto s] \equiv \forall A.\langle t_1[X \mapsto s], \dots, t_n[X \mapsto s] \rangle$$

$$(\forall A.[a]t)[X \mapsto s] \equiv \forall A.[a](t[X \mapsto s])$$

$$(\forall A.\pi \cdot X)[X \mapsto \forall B.u] \equiv \forall (A \cup \pi \cdot B).\pi \cdot u.$$

Swapping

$$(a\ b) \cdot \forall A.u \equiv \forall (a\ b) \cdot A.(a\ b) \cdot u$$

$$(a\ b) \cdot a \equiv b \quad (a\ b) \cdot b \equiv a \quad \text{and} \quad (a\ b) \cdot c \equiv c \quad (c \neq a, b)$$

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

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

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

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

So:

$$(a\ b) \cdot \lambda[a] \forall b, c. abX \equiv \lambda[b] \forall a, c. ba(a\ b) \cdot X$$

$$\begin{aligned} \left((a\ b) \cdot \lambda[a] \forall b, c. abX \right) [X \mapsto a] &\equiv \lambda[b] \forall a, c. bab \\ &\equiv \left(\lambda[b] \forall a, c. ba(a\ b) \cdot X \right) [X \mapsto a] \end{aligned}$$

Fresh

$$\frac{a\#u}{a\#\forall A.u} \quad \frac{}{a\#b} \quad \frac{a\#s}{a\#fs} \quad \frac{a\#s_1 \cdots a\#s_n}{a\#\langle s_1, \dots, s_n \rangle}$$
$$\frac{}{a\#[a]s} \quad \frac{a\#s}{a\#[b]s} \quad \frac{\pi^{-1} \cdot a\#X}{a\#\pi \cdot X} \quad \frac{a\#a}{P} (\#\perp)$$

Local

$$\begin{array}{c}
 \frac{}{a@IA.u} \quad a \in A \qquad \frac{a@u}{a@IA.u} \quad a \notin A \\
 \\
 \frac{}{a@b} \qquad \frac{a@s}{a@fs} \qquad \frac{a@s}{a@[b]s} \qquad \frac{}{a@[a]s} \\
 \\
 \frac{a@s_1 \cdots a@s_n}{a@<s_1, \dots, s_n>} \qquad \frac{\pi^{-1} \cdot a@X}{a@\pi \cdot X} \qquad \frac{a\#X}{a@X} (\#@) \qquad \frac{a@a}{P} (@\perp)
 \end{array}$$

Alpha-equivalence

$$\begin{array}{c}
 \frac{u \approx v \quad B \setminus A @ u \quad A \setminus B @ v}{\forall A. u \approx \forall B. v} \\
 \frac{}{a \approx a} \quad \frac{s_1 \approx t_1 \cdots s_n \approx t_n}{\langle s_1, \dots, s_n \rangle \approx \langle t_1, \dots, t_n \rangle} \quad \frac{s \approx t}{fs \approx ft} \\
 \frac{s \approx t}{[a]s \approx [a]t} \quad \frac{s \approx (ab) \cdot t \quad a \# t}{[a]s \approx [b]t} \quad \frac{ds(\pi, \pi') \# X}{\pi \cdot X \approx \pi' \cdot X}
 \end{array}$$

where

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

Scope-extrusion for $|$

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

$$a@Y \vdash (\lambda a.X) | Y \rightarrow \lambda a.(X | Y).$$

This works in collaboration with rule F :

$$b@X \vdash \lambda a.X \rightarrow \lambda b.(b a)X$$

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

Unification, confluence, etc.

λ 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, λ may ‘swallow’ names. For example (on the face of it) $\lambda a.X \approx \lambda a.a$ has two solutions: X maps to a , and X maps to $\lambda a.a$. 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 (*open*) 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 and similar conditions to neatly express reduction strategies.

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.

$$\begin{aligned} \bar{a}b \mid a[c].Y &\rightarrow Y\{c\mapsto b\} & \nu[c]X &\rightarrow \forall c.X & !X &\rightarrow !X \mid X \\ a@P \vdash P \mid (\forall a.Q) &\rightarrow \forall a.(P \mid Q). \end{aligned}$$

$X\{c\mapsto b\}$, 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

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

This has been implemented by Christophe Calves.