# Nominal Rewriting

Murdoch J. Gabbay

Joint work with Maribel Fernández

Leicester, 9/8/2005

Thank you for having me over (and for the splendid lunch, with tablecloths!).

This talk has 17 slides including the two so far.

— Jamie 'no time pressure' Gabbay

# The issue

Rewriting is a framework to express computation, logic, and processes. Anybody can ‘do rewriting’! Just pick your favourite formal grammar, and write down rewrite rules describing how it evolves.

For example:

- The $\lambda$-calculus is a rewrite system with terms *blah for $\lambda$-terms* and rewrites *blah for reductions*.
- The $\pi$-calculus is a rewrite system with terms *blah for $\pi$-terms* and rewrites *blah for reactions*.
- First-order logic is a rewrite system with terms *blah for judgements* and rewrites *blah for derivation rules, read bottom-up*.

## For example: a $\lambda$-calculus

Terms are given by: $t ::= a \mid \lambda a.t \mid tt \mid t[a{\mapsto}t]$.

Reductions are given by:

$$(\lambda a.X)Y \leadsto X[a{\mapsto}Y] \qquad (XY)[a{\mapsto}U] \leadsto (X[a{\mapsto}U])(Y[a{\mapsto}U])$$

$$(\lambda b.X)[a{\mapsto}U] \leadsto \lambda b.(X[a{\mapsto}U]) \quad (a \notin FV(X))$$

- $X$, $Y$, and $U$ are meta-variables standing for unknown terms (alternative: one rewrite rule for every term, analagous to *axiom schemes* in logic).
- $a$ and $b$ are (what I shall call) names or atoms; variable symbols of the object-language (alternative: use meta-variables to represent names).
- Names get abstracted whence capture-avoidance side-conditions (with alternative: use $\lambda$-abstraction to represent object-level abstraction).

# Basic idea

- Unknowns (meta-variables) $X, Y, Z, U$ are distinct from atoms (object-level variables) $a, b, c$.
- There is an abstraction operator $[a]s$ which does not bind, in the sense that substitution of $t$ for $X$ in $[a]s$ is first-order textual substitution and does not avoid capture.
- There is a context of freshness assumptions $a\#X$ in which rewriting takes place. We are *not allowed* to substitute $t$ for $X$ if $a\#t$ is not provable.

I will define 'not allowed' later.

$(\lambda[a]X)$ is '$\lambda$ of abstract $a$ in $X$'. $\lambda$ is an operator; all abstractors are (for sorts, see later). So similarly, $\nu[a]P$ is '$\nu$ of abstract $a$ in $P$'.

# $\#$ Freshness-as-a-logical-notion

$$\frac{a\#s_1 \quad \cdots \quad a\#s_n}{a\#\langle s_1, \ldots, s_n \rangle} \qquad \frac{a\#s}{a\#f\,s} \qquad \frac{a\#s}{a\#[b]s}$$

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

Write $\Delta$ for a set of apartness assumptions $a\#X$. Write $\Delta \;\vdash\; a\#s$ when assumptions $\Delta$ prove $a\#s$.

$$a\#X \;\vdash\; a\#\langle X, [a]Y \rangle$$

$$a\#X,\; b\#Y \;\vdash\; a\#\langle (a\,b) \cdot X, (b\,c) \cdot Y \rangle$$

$\pi$ is a atoms-permutation, e.g. $(a\ b)$ swaps $a$ and $b$. We may use them to rename atoms to avoid capture, e.g. when we deduce equality:

# $\approx$ Equality-as-a-logical-notion

$$\frac{s_1 \approx t_1 \quad \cdots \quad s_n \approx t_n}{\langle s_1, \ldots, s_n \rangle \approx \langle t_1, \ldots, t_n \rangle} \quad \frac{s \approx t}{fs \approx ft} \quad \frac{}{a \approx a} \quad \frac{t \approx t'}{t' \approx t}$$

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

$ds(\pi, \pi') = \left\{a \;\middle|\; \pi(a) \neq \pi'(a)\right\}$ the difference set.

Write $\Delta \vdash s \approx t$ when $\Delta$ proves $s \approx t$.

$$\begin{array}{rcccc} a, b\#X & \vdash & (a\ b) \cdot X & \approx & X \\ b\#X & \vdash & \lambda[a]X & \approx & \lambda[b](b\ a) \cdot X \end{array}$$

Nominal matching/unification algorithms invert these rules and include a substitution step to solve $X \approx t$.

# Unification and matching I

$$
\begin{array}{rcll}
a\#b, Pr & \to & Pr & \\
a\#f s, Pr & \to & a\#s, Pr & \\
a\#\langle s_1, \ldots, s_n\rangle, Pr & \to & a\#s_1, \ldots, a\#s_n, Pr & \\
a\#[b]s, Pr & \to & a\#s, Pr & \\
a\#[a]s, Pr & \to & Pr & \\
a\#\pi \cdot X, Pr & \to & \pi^{-1} \cdot a\#X, Pr & \qquad \pi \not\equiv \textbf{Id}
\end{array}
$$

Thus:

$$a\#\langle X, [a]Y\rangle \xrightarrow{*} a\#X \qquad\qquad a\#f a \xrightarrow{*} a\#a$$

$$a\#\langle (a\ b) \cdot X, (b\ c) \cdot Y\rangle \xrightarrow{*} b\#X, a\#Y$$

# Unification and matching II

$$
\begin{array}{rcl}
a \approx a, Pr & \longrightarrow & Pr \\
\langle l_1, \ldots, l_n \rangle \approx \langle s_1, \ldots, s_n \rangle, Pr & \longrightarrow & l_1 \approx s_1, \ldots, l_n \approx s_n, Pr \\
f\,l \approx f\,s, Pr & \longrightarrow & l \approx s, Pr \\
[a]l \approx [a]s, Pr & \longrightarrow & l \approx s, Pr \\
[b]l \approx [a]s, Pr & \longrightarrow & (a\ b) \cdot l \approx s, a\#l, Pr \\
\pi \cdot X \approx \pi' \cdot X, Pr & \longrightarrow & ds(\pi, \pi')\#X, Pr
\end{array}
$$

Thus:

$$[a]X \approx [b]X \xrightarrow{*} a\#X,\ b\#X \qquad\qquad a \approx b \not\xrightarrow{*}$$

$$[b]Y \approx [a]X \ \longrightarrow\ a\#Y,\ (a\ b) \cdot Y \approx X \ \xrightarrow{X \mapsto (a\ b) \cdot Y}\ a\#Y$$

- Urban, Pitts, Gabbay ‘Nominal Unification’.
- Fernández, Gabbay ‘Nominal Rewriting’, ‘Extensions of Nominal Rewriting’.
- Gabbay, ‘NEW calculus of contexts’.
- Gabbay, Mousavi, ‘Nominal SOS’.
- Pitts, Shinwell, and others, ‘FreshML’.
- Cheney, Urban, ‘$\alpha$-prolog’.

All on the web.

# Brief summary

Nominal Techniques typically:

- Separate meta-level unknowns from object-level variable symbols.
- Separate syntactic identity $\equiv$ from $\alpha$-equivalence $\approx$, and therefore also binding ($\alpha$-renaming preserves identity) from abstraction (only preserves $\alpha$-equivalence).

  $\alpha$-equivalence is the useful notion of equivalence, we just do not call $\alpha$-equivalent terms identical.
- Enrich the context with assumptions about freshnesses $a\#X$.
- Enrich terms themselves with permutations suspended on unknowns $\pi \cdot X$ and abstractions $[a]X$.

Write $V(s)$ for the $X$ in $s$ and $A(s)$ for the $a$ in $s$. Write $V(\nabla)$ for $\nabla$ a set of freshness assertions.

A nominal rewrite rule (over a signature $\Sigma$) is a tuple $(\nabla, l, r)$, we write it $\nabla \ \vdash \ l \to r$, such that $V(r) \cup V(\nabla) \subseteq V(l)$. We may write $l \to r$ for $\emptyset \ \vdash \ l \to r$.

- $a\#X \ \vdash \ (\lambda[a]X)Y \to X$ is a form of trivial $\beta$-reduction.
- $a\#X \ \vdash \ X \to \lambda[a](Xa)$ is $\eta$-expansion.
- $XY \to XX$ is strange but quite valid.
- $a \to b$ is a rewrite rule.
- $a\#Z \ \vdash \ X\lambda[a]Y \to X$ is not a rewrite rule; $Z \notin V(X\lambda[a]Y)$. $X \to Y$ is also not a rewrite rule.

I'm telling you we can also do explicit substitutions, the $\pi$-calculus, and lots of similar cases.

A Nominal Signature $\Sigma$ is some sorts of atoms $\mathbb{A}$, base data sorts $s$ (e.g. $\mathbb{N}$, $\mathbb{B}$), and function symbols $f$ of arity $\tau_1 \longrightarrow \tau_2$. If $\tau_1$ is an empty product say $f$ is 0-ary (i.e. a constant) and omit the arrow.

Term sorts are inductively defined by:

$$\tau ::= \nu \mid s \mid \tau \times \ldots \times \tau \mid [\nu]\tau.$$

$\tau_1 \times \ldots \times \tau_n$ is a product sort. $[\nu]\tau$ is an abstraction sort. Terms are defined in the next slide, but first an example:

A nominal signature for a fragment of ML has one sort of atoms $\mathbb{A}$, one sort of data $exp$, and function symbols with arities

$$\mathtt{var} : \mathbb{A} \longrightarrow exp \qquad \mathtt{app} : exp \times exp \longrightarrow exp$$

$$\mathtt{lam} : [\mathbb{A}]exp \longrightarrow exp \qquad \mathtt{let} : exp \times [\mathbb{A}]exp \longrightarrow exp$$

$$\mathtt{letrec} : [\mathbb{A}](([\mathbb{A}]exp) \times exp) \longrightarrow exp$$

# Going further: Extended Nominal Rewriting

Extended nominal terms extend freshness contexts with conditions such as $\bullet X$ for '$X$ is closed' (meaning: $a\#X$ is provable for all $a$).

With such a condition in the context, we can only substitute $t$ for $X$ if $a\#t$ is provable for all $a$. (Actually, you can always do the substitution, but if $\bullet a$ gets in the context you can prove anything.) Because $t$ is finite, it suffices to consider all $a$ in $t$, and one fresh $t$.

The logic thickens but so long as it stays decidable this is not a problem.

We also extend terms with $\text{И}a.t$. И is not an abstractor, so $\text{И}a.a \not\approx \text{И}b.b$. It is not a binder either, so $\text{И}a.a \neq \text{И}b.b$. However, we assume a rewrite rule

(F) $$b\#X \ \vdash\ \text{И}a.X \rightsquigarrow \text{И}b.(b\ a)\cdot X.$$

И models name-generation, as distinct from name-binding or name-abstraction!

## Meta-properties

- Operates on true first-order terms (abstract syntax trees).
- Critical pairs lemma.
- Orthogonal systems are confluent.
- Matching/Unification/Rewriting is (at worst) quadratic.
- Can express rewriting for syntax-with-binders.
- Is implement(able/ed).

# Rewriting: Conclusions

The interplay of the different notions of *binding*, *abstraction*, *name-generation*, closure, and so on, is *non-trivial* and (I think) *illuminating and very interesting*.

This material does *not* seem to fit into the usual category-theoretic frameworks: Gabbay-Pitts $\text{И}$ is useful and has no known universal characterisation (work by Menni aside), neither does the 'finite support' assumption which underlies it, and the $X$ have a different flavour, in the presence of abstraction $[a]X$, than the uniformity induced by functors.

# More conclusions

The logical laws from slides 6 and 7 as semantic equalities, we get a notion of substitution. This suggests (to me) that we can talk about modularity, accessibility, and movement in new ways — independently of the underlying model since (as with logics, which I also investigate) nominal rewriting says nothing about what *is* rewritten, it only presumes it can be described by terms.

Category Theory, $\lambda$-calculi, specification languages, spatial logics ... the way computer science is developing is forcing us to use these frameworks to talk about modularity, accessibility, and movement — so we can slice up our systems and describe inherently mobile ones.

Rich possibilities for collaboration arise from this NEW way of thinking.