Nominal Rewriting Systems

Murdoch J. Gabbay

Work with Maribel Fernández and Ian Mackie

August 26, Verona, PPDP’2004

The talk in a slide:

- Rewriting is an encompassing framework for expressing logic and computation. Real logics and computing languages (e.g. FOL, λ-calculus) have binding (α/β-equivalence). β-equivalence is undecidable, can cause problems in higher-order systems.
- We have a decidable theory of α-equivalence, based on Fraenkel-Mostowski sets.
- Cross it with a first-order theory of rewriting.
- Get a theory of Nominal Rewriting—decidable, and with binding.
- Verify some good properties of the system (critical pairs lemma, linear time decidability, as expressive as Combinatory Rewriting).

In more detail:

Urban, Pitts, and Gabbay presented a decidable linear time unification algorithm for Nominal Terms.

Nominal terms are similar to first-order terms but the theory of equality is not just literal equality on syntax trees, but α-equivalence $\approx_\alpha$ with respect to a special abstraction operator (examples below) on atoms $a, b, c \in \mathbb{A}$, written $[a]t$.

Nominal terms may contain unknowns X (representing unknown nominal terms). These may occur under abstraction $[a]X$. The unification algorithm finds a substitution σ of Xs for ss in t and t' such that $t\sigma \approx_\alpha t'\sigma$.

Nominal Rewriting is a natural extension of first-order rewriting with respect to nominal terms and the matching algorithm obtained by rewstricting nominal unification. The payoff is a first-order-like treatment of binding in syntax.

Signatures and Sorts

A **Nominal Signature** Σ is some sorts of atoms ν, base data sorts s (e.g. $\mathbb{N}$, $\mathbb{B}$), and function symbols f of arity $\tau_1 \rightarrow \tau_2$. If τ_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 ν, one sort of data exp, and function symbols with arities

$$\mathtt{var} : \nu \rightarrow exp \qquad \mathtt{app} : exp \times exp \rightarrow exp$$

$$\mathtt{lam} : [\nu]exp \rightarrow exp \qquad \mathtt{let} : exp \times [\nu]exp \rightarrow exp$$

$$\mathtt{letrec} : [\nu](([\nu]exp) \times exp) \rightarrow exp$$

Terms

Fix Σ. For each τ fix countably infinite term variables $X, Y, Z \in \mathcal{X}_\tau$ meta-level unknowns. For each ν fix countably infinite atoms $a, b, c, f, g, h, \ldots \in \mathcal{A}_\nu$ object-level variable symbols.

Nominal Terms are:

$$t ::= \quad a_\nu \mid (\pi \cdot X)_\tau \mid \langle t_{1\tau_1}, \ldots, t_{n\tau_n} \rangle_{\tau_1 \times \ldots \times \tau_n} \mid ([a_\nu]t_\tau)_{[\nu]\tau} \mid (f_{\tau_1 \to \tau_2} t_{\tau_1})_{\tau_2}$$

and called resp. atoms, moderated variables, tuples, abstractions and function applications. Ground terms are terms without variables.

a is abstracted in $[a]t$, not under $[a]$- it is free.

These terms have a notion of position as usual in first-order rewriting, only the position of X in $\pi \cdot X$ is ϵ.

For example

For our example Σ, write

a	for	$\mathtt{var}(a)$
tt'	for	$\mathtt{app}\langle t, t' \rangle$
$\lambda[a]t$	for	$\mathtt{lam}([a]t)$
$\mathtt{let}\ a{=}t\ \mathtt{in}\ t'$	for	$\mathtt{let}\langle t, [a]t' \rangle$
$\mathtt{letrec}\ (fa){=}t\ \mathtt{in}\ t'$	for	$\mathtt{letrec}[f]\langle [a]t, t' \rangle.$

a, $(\lambda[a]aa)(\lambda[a]aa)$, and $\mathtt{letrec}\ (fa)\ {=}a\ \mathtt{in}\ fb$ are terms.

f is abstracted in t and t', and a in t, in $\mathtt{letrec}\ f\ a{=}t\ \mathtt{in}\ t'$.

Swappings

$(a\ b)$ a swapping is a pair of atoms. Permutations $\pi ::= \mathbf{Id} \mid (a\ b) \cdot \pi$ are lists of swappings. (**Id** is the identity.)

Swappings (and thus permutations) act on atoms

$$(a\ b)(a) \stackrel{\mathrm{def}}{=} b \qquad (a\ b)(b) \stackrel{\mathrm{def}}{=} a \quad \text{and} \quad (a\ b)(c) \stackrel{\mathrm{def}}{=} c\ (c \neq a, b).$$

The action extends to terms:

$$(a\ b)(X) = (a\ b) \cdot x \qquad\qquad (a\ b)[n]t = [(a\ b)(n)](a\ b)(t)$$

Syntactic equality $\equiv$ is not modulo α-equivalence: $[a]a \not\equiv [b]b$.

We develop an explicit theory of α-equivalence in context.

Fresh

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

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

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

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

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

$\approx_\alpha$, a notion of α-equivalence in context

$$\frac{s_1 \approx_\alpha t_1 \quad \cdots \quad s_n \approx_\alpha t_n}{\langle s_1, \ldots, s_n \rangle \approx_\alpha \langle t_1, \ldots, t_n \rangle} \qquad \frac{s \approx_\alpha t}{f s \approx_\alpha f t} \qquad \frac{}{a \approx_\alpha a} \qquad \frac{t \approx_\alpha t'}{t' \approx_\alpha t}$$

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

$ds(\pi, \pi') = \{a \mid \pi(a) \neq \pi'(a)\}$ the difference set.

Write $\Delta \vdash s \approx_\alpha t$ when Δ proves $s \approx_\alpha t$.

$$\begin{aligned} a, b\#X \vdash \quad (a\, b) \cdot X &\approx_\alpha X \\ b\#X \vdash \quad \lambda[a]X &\approx_\alpha \lambda[b](b\, a) \cdot X \end{aligned}$$

The matching/unification algorithms invert these rules and include a substitution step to solve $X \approx_\alpha t$. We omit details.

Terms-in-context

Because the useful notion of equality, $\approx_\alpha$, is in a context, we work with terms-in-context $\Gamma \vdash t$. For example:

1. $\emptyset \vdash a$.
2. $a\#X \vdash [a]X$.
3. $a\#X, b\#Y \vdash \langle X, Y\rangle$.

We may write $\emptyset \vdash t$ as just t.

Rewrite rules

Write $V(s)$ for $X \in \mathcal{X}$ mentioned in s and $A(s)$ for atoms mentioned in s (free or abstracted). Similarly write $V(\nabla)$.

A nominal rewrite rule over Σ is a tuple (∇, l, r), we write it $\nabla \ \vdash \ l \rightarrow r$, such that $V(r) \cup V(\nabla) \subseteq V(l)$.

We may write $l \rightarrow r$ for $\emptyset \ \vdash \ l \rightarrow r$.

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

Examples

We discuss matching, then rewriting, in a moment. Here are some examples:

1. X rewrites with $\emptyset \vdash X \rightarrow \langle X, X\rangle$ to $\langle X, X\rangle$.
   Y rewrites to $\langle Y, Y\rangle$.

2. a rewrites with $\emptyset \vdash a \rightarrow a$ to a. b does not rewrite.

3. $a\#X \vdash \langle X, X\rangle$ rewrites with $a\#Z \vdash \langle Z, Z\rangle \rightarrow \langle Z, a\rangle$ to $\langle Z, a\rangle$. $\langle X, X\rangle$ does not rewrite, neither does $\langle a, b\rangle$, but $a\#X \vdash \langle b, X\rangle$ rewrites to $\langle b, a\rangle$.

4. $[a]a$ rewrites with $\emptyset \vdash [b]b \rightarrow [b]c$ to $[a]c$, to $[b]c$, and $[d]c$, but not $[c]c$. The former are all provably α-equivalent in the context $\emptyset$.
   $a\#X \vdash [a]X$ also rewrites with the same rule to $[a]c$.

Matching

Call a term in context a pair $\Gamma \vdash t$.

A matching problem is a pair of them, $(\nabla \vdash l) \ {}_?{=} \ (\Delta \vdash s)$.

A solution is a substitution θ such that

- $\theta X \equiv X$ for X in $V(\Delta \vdash s)$.
- $\Delta \vdash l\theta \approx_\alpha s$.
- $\Delta \vdash \nabla\theta$.

If a solution exists then a most general one is the θ from (θ, Γ) solving $l \ {}_?{=} \ s$.

Rewriting

Given $R = \nabla \vdash l \to r$ say s rewrites with R to t, in a context Δ, or just $\Delta \vdash s \xrightarrow{R} t$, when:

- $V(R) \cap V(\Delta, s) = \emptyset$ (wlog).
- There exists a position p in s and a solution θ to $(\nabla \vdash l) \;_{?}{=} (\Delta \vdash s|_p)$.
- $\Delta \vdash s[r\theta]_p \approx_\alpha t$.

Two basic lemmas of $\approx_\alpha$, and a corollary

Lemma: If $\Delta \vdash t \approx_\alpha s|_p$ then $\Delta \vdash s[t]_p \approx_\alpha s$.

Lemma: If $\Delta \vdash t \approx_\alpha t'$ and if p is a position in s, then $\Delta \vdash s[t]_p \approx_\alpha s[t']_p$.

E.g. $\emptyset \vdash [a]a \approx_\alpha [b]b$ and $\emptyset \vdash [a][a]a \approx_\alpha [a][b]b$.

If $s \approx_\alpha s'$ and $t \approx_\alpha t'$ it is not necessarily the case that $s[t]_p \approx_\alpha s'[t']_p$. For example, $[a]a \approx_\alpha [b]b$ and $a \approx_\alpha a$ but $[a]a \not\approx_\alpha [b]a$.

Corollary: The latter two conditions defining $\Delta \vdash s \xrightarrow{R} t$ can be expressed succinctly as $(\nabla \vdash (s[l]_p, s[r]_p)) \;{}_?{=}\; (\Delta \vdash (s, t))$ for some p.

Critical pair lemma

Call a valid pair of rewrites $\Delta \vdash s \rightarrow t_1, t_2$ a peak.

Suppose

1. $R_i = \nabla_i \vdash l_i \rightarrow r_i$ for $i = 1, 2$ are copies of two rules in $\mathcal{R}$ such that $V(R_1) \cap V(R_2) = \emptyset$ (R_1 and R_2 could be copies of the same rule).
2. p is a position in l_1.
3. $l_1|_p \;{}_?{=}_?\; l_2$ has a solution (Γ, θ), so that $\Gamma \vdash l_1|_p\theta \approx_\alpha l_2\theta$.

Then call the pair of terms-in-context

$$\nabla_1\theta, \nabla_2\theta, \Gamma \vdash (r_1\theta, l_1[r_2\theta]_p)$$

a critical pair. If $p = \epsilon$ and R_1, R_2 are copies of the same rule, or if p is the position of a variable in l_1 then we say the critical pair is trivial.

Theorem: If all critical pairs are joinable, then rewriting is locally confluent.

Simulating Combinatory Reduction Systems (CRS)

First, note that Nominal Rewriting contains First-Order rewriting, just by omitting abstraction $[a]t$ and moderations $\pi \cdot X$.

CRS can be encoded with a little more effort. Fix some CRS over an alphabet A. Define a nominal signature Σ_A with one sort of atoms (ν), one sort of data (δ), the term sorts generated from these, and a set of function symbols which contains the function symbols of the CRS R and a new function symbol $\mathtt{sub}$ representing substitution, which we sugar to $t[a \mapsto s]$ and more generally to $t[a_1 \mapsto s_1, \ldots, a_n \mapsto s_n]$.

We obtain a nominal rewriting system $\mathcal{R}$.

Examples of the translation

β-reduction in the CRSs syntax is:

$$\mathsf{app}(\mathsf{lambda}([a]Z(a)), Z') \to Z(Z')$$

The translation is:

$$a\#Z' \vdash \mathsf{app}(\mathsf{lambda}([a]Z), Z') \to Z[a \mapsto Z']$$

A CRS rule defining a differentiation operator is:

$$\mathsf{diff}([a]\mathsf{sin}(Z(a))) \to [b]\mathsf{mult}(\mathsf{app}(\mathsf{diff}([c]Z(c)), b), \mathsf{cos}(Z(b)))$$

The translation is:

$$b, c\#Z \vdash \mathsf{diff}([a]\mathsf{sin}(Z)) \to [b]\mathsf{mult}\langle \mathsf{app}\langle \mathsf{diff}([c]Z[a \mapsto c]), b\rangle, \mathsf{cos}(Z[a \mapsto b]\rangle)$$

Soundness and completeness

The translation is sound, and complete. Soundness is modulo rewriting some substitutions (CRS elide β-reduction steps). Completeness is direct.

Theorem: Let t be a term in a CRS R (and therefore also in $\mathcal{R}$). If $\vdash t \rightarrow_{\mathcal{R}} u$ then there exists u' such that $\vdash u \rightarrow^*_{\mathcal{R}} u'$ and $t \rightarrow_R u'$.

Theorem: Let t and u be arbitrary terms in the CRS R (and therefore also in $\mathcal{R}$). If $t \rightarrow_R u$ then $\vdash t \rightarrow^*_{\mathcal{R}} u$.

Closed rewriting (briefly)

We have not discussed closed rewriting for efficiency in the presence of equivariance: under certain reasonable reasonableness conditions rewriting is linear time decidable even if the system is equivariant and therefore has infinitely many rules, such as $a \longrightarrow a$, $b \longrightarrow b$, ...

Conclusions

Main results so far:

- Nominal Rewriting has a critical pair lemma,
- is linear time decidable,
- and is as expressive as CRS (the translation of a CRS is reasonable in the sense above).

Future work

- Extend these results to a framework that can express more complex apartness conditions than $a\#X$; for example 'X closed'.
- Prove more powerful confluence results, including criteria on rewrite rules for global confluence.
- Include a И term-former to generate atoms on-the-fly.
- Consider generalization and Inductive Logic Programming.