Nominal Unification

Murdoch J. Gabbay

September 10, Imperial College London

Nominal Unification

Work by Urban, Pitts, and Gabbay.

Nominal terms are similar to first-order terms but the theory of equality is not just literal equality on syntax trees, but α-equivalence $\approx_\alpha$. This allows us to talk about unification of terms up to binding, but not up to full β-equivalence.

We obtain a framework with much of the flavour of first-order unification (where equality is literal equality of abstract syntax trees), but a useful fragment of the power of higher-order systems.

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 $\tau_1 \longrightarrow \tau_2$. 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

$$\texttt{var} : \nu \longrightarrow exp \qquad \texttt{app} : exp \times exp \longrightarrow exp$$
$$\texttt{lam} : [\nu]exp \longrightarrow exp \qquad \texttt{let} : exp \times [\nu]exp \longrightarrow exp$$
$$\texttt{letrec} : [\nu]((([\nu]exp) \times exp) \longrightarrow exp$$

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:

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

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	$\mathtt{let}\langle t, [a]t'\rangle$
$\mathtt{letrec}\ (f\,a){=}t\ \mathtt{in}\ t'$	for	$\mathtt{letrec}[f]\langle [a]t, t'\rangle$.

a, $(\lambda[a]aa)(\lambda[a]aa)$, and $\mathtt{letrec}\ (f\,a)\ {=}a\ \mathtt{in}\ fb$ 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

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

The action extends to terms:

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

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

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

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

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

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

Write $\langle a\#s\rangle_{\#sol}$ for the least apartness context such that $\langle a\#s\rangle_{\#sol} \vdash a\#s$. $\langle a\#s\rangle_{\#sol}$ is well-defined, unique, and always

exists.

$$\begin{aligned} \langle a\#\langle X, [a]Y\rangle\rangle_{\#sol} &= \{a\#X\} \\ \langle a\#\langle (a\ b)\cdot X, (b\ c)\cdot Y\rangle\rangle_{\#sol} &= \{b\#X, a\#Y\}. \end{aligned}$$

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

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

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

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

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

$$a, b\#X \vdash (a\,b)\cdot X \approx_\alpha X$$
$$b\#X \vdash \lambda[a]X \approx_\alpha \lambda[b](b\,a)\cdot X$$

Unification and Matching

The matching and unification algorithms invert these rules. . .

$$
\begin{aligned}
\langle l_1, \ldots \rangle \stackrel{?}{=} \langle s_1, \ldots \rangle, P &\rightarrow l_1 \stackrel{?}{=} s_1, \ldots, P \\
f l \stackrel{?}{=} f s, P &\rightarrow l \stackrel{?}{=} s, P \\
[a]l \stackrel{?}{=} [a]s, P &\rightarrow l \stackrel{?}{=} s, P \\
[b]l \stackrel{?}{=} [a]s, P &\rightarrow (a\ b) \cdot l \stackrel{?}{=} s, a\#l, P \\
a \stackrel{?}{=} a, P &\rightarrow P \\
\pi \cdot X \stackrel{?}{=} \pi' \cdot X, P &\rightarrow ds(\pi, \pi')\#X, P \\
a\#s, P &\rightarrow \langle a\#s \rangle_{\#sol}, P \\
& \qquad (s \not\equiv X)
\end{aligned}
$$

P is a possibly empty list of atomic problems $s \stackrel{?}{=} t$ and $a\#t$.

. . . and include a substitution step to solve $X \approx_\alpha t$.

$$(Matching) \quad \pi \cdot Y \stackrel{?}{=} s, P \quad \stackrel{Y \mapsto \pi^{-1} \cdot s}{\longrightarrow} \quad P[Y \mapsto \pi^{-1} \cdot s] \quad (Y \notin s)$$

$$(Unification) \quad l \stackrel{?}{=} \pi \cdot X, P \quad \stackrel{X \mapsto \pi^{-1} \cdot l}{\longrightarrow} \quad P[X \mapsto \pi^{-1} \cdot l] \quad (X \notin l)$$

The input to the algorithm is a problem P. The output is a pair Δ, σ of some apartness conditions $a\#X$ and a substitution.

- Solution of $a\#\langle X, Y\rangle$ is $\{a\#X, a\#Y\}$.
- Solution of $[a]X \stackrel{?}{=} [b]\langle Y, Y\rangle$ is $\{a\#Y\}$ and $X \mapsto \langle (a\ b) \cdot Y, (a\ b) \cdot Y\rangle$.

More examples

Suppose a signature with an atomic sort ν, a sort of data τ, and constructors application of arity $\tau \times \tau \to \tau$ and abstraction of arity $[\nu]\tau \to \tau$, written in standard λ-notation.

- $\lambda a.\lambda b.Xb \stackrel{?}{=}_? \lambda b.\lambda a.aX$ has no solution.
- $\lambda a.\lambda b.Xb \stackrel{?}{=}_? \lambda b.\lambda a.aY$ has solution $[X \mapsto a,\ Y \mapsto b]$.
- $\lambda a.\lambda b.bX \stackrel{?}{=}_? \lambda b.\lambda a.aY$ has solution $[X \mapsto (a\ b) \cdot Y]$.
- $\lambda a.\lambda b.bX \stackrel{?}{=}_? \lambda a.\lambda a.aY$ has solution $\{b\#Y\}$ and $[X \mapsto (b\ a) \cdot Y]$.

Properties of Nominal Unification and the algorithm

If a unifier exists the algorithm will find one in linear time, and it will be most general in a natural sense.

If no unifier exists the algorithm will fail in linear time.

Possible applications to Inductive Logic Programming

Nominal Terms are suitable to describe first-order logic and the lambda-calculus—not up to β-conversion. Unification could be used directly to learn in these languages, by guessing generalizations and then unifiying them.

I see no barrier in principle to a similar algorithm to directly compute least general generalisations of finite sets of nominal terms.

Possible applications to Inductive Logic Programming

A useful way to proceed may be to consider existing systems such as L_λ and M_λ, and their generalisation algorithms, and see if they can be encoded in Nominal Terms; if not, what needs to be put in—probably a restricted explicit substitution $([\nu])\tau \times \tau \rightarrow \tau$ written $X[a \mapsto t]$.

The syntactic restrictions on L_λ and M_λ terms could correspond to restrictions on t and possibly X. L_λ and M_λ are recovered by considering closed (no X) terms, quotiented by $(\lambda a.t)t' =_\beta t[a \mapsto t']$.

An advantage of Nominal approaches is that they can express possibly capturing substitution; for example $([a]X)[X \mapsto t]$ can substitute X for a t containing a. They can also express capture-avoiding substitution, by simultaneously solving the problem $a\#t$.

We can talk about "X in which t and t' occur *somewhere*" just by writing $X[a \mapsto t,\ b \mapsto t']$. In a Higher-Order setting we might prefer $(\lambda a, b.Fab)tt'$.

Matching

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

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

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 $l \stackrel{?}{=} s$.

Rewriting

Given $R = \nabla \vdash l \longrightarrow r$ 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 $(\nabla \vdash l) \stackrel{?}{=} (\Delta \vdash s|_p)$.
- $\Delta \vdash s[r\theta]_p \approx_\alpha t$.

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 $(\nabla \vdash (s[l]_p, s[r]_p)) \stackrel{?}{=} (\Delta \vdash (s,t))$ 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. $l_1|_p \stackrel{?}{=} l_2$ 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 `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:

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

The translation is:

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

A CRS rule defining a differentiation operator is:

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

The translation is:

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

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.