Closed nominal rewriting and efficiently computable nominal algebra equality

Murdoch J. Gabbay
Joint work with Maribel Fernández

July 14, 2010

Nominal techniques then

Nominal techniques were introduced by myself and Pitts in a 1999 LICS paper. See also the journal paper "A New Approach to Abstract Syntax with Variable Binding", Formal Aspects of Computing, 13(3-5): 341-363 (2002).

This introduced 'nominal' machinery like atoms and atoms-abstraction, (equivariant) Fraenkel-Mostowski sets, permutations, equivariance, the $\mathcal{N}$-quantifier, and 'nominal' abstract syntax.

At the time, what Pitts and I did was considered pretty risky.

I remember Andrew telling me as a PhD student "Jamie, this might not work—and if it doesn't work I can't rescue you and you're screwed".

Maybe he didn't use exactly these words.

Nominal techniques now

By my count there are seven 'nominal' talks at FLOC 2010:

- "Nominal Unification - Hitting a Sweet Spot" by Christian Urban in UNIF, apparently happening right now 14:00-15:00.
- "Nominal Theory as an Extension of First-Order Theory" by Christophe François Olivier Calvès at 15:30 today, also in UNIF.
- "An Efficient Nominal Unification Algorithm" by Jordi Levy and Mateu Villaret in RTA (16:30-17:00 yesterday).
- "(Nominal) Unification by Recursive Descent with Triangular Substitutions" by Ramana Kumar and Michael Norrish in ITP (10:30-11:00 last Sunday).
- "Proof Pearl: A New Foundation for Nominal Isabelle" by Brian Huffman and Christian Urban also in ITP (15:30-16:00 last Sunday).
- "Closed nominal rewriting: properties and applications" by Maribel Fernández in HOR (09:00-10:00 this morning).
- This talk.

Yes, but what are nominal techniques?

I speak for myself here, not my coauthor Maribel.

What makes nominal techniques different is that we build our mathematical universe using atoms (urelemente).

For set theorists: we use a cumulative hierarchy of ZFA sets (we start from atoms $\mathbb{A} = \{a, b, c, \ldots\}$ instead of from $\varnothing$).

For category theorists: instead of building everything over the category of sets, we build it over the category of nominal sets.

Names are data; atoms.

This is a non-trivial mathematical step.

Symmetry properties of atoms—e.g. that atoms are symmetric under permutation—directly import into the nominal metatheory.

Thus, if we take names to be data we do not only get a new datatype; we get a new meta-theory.

Equivariance: symmetry of atoms under permutations

Because atoms are atomic, we can permute them without affecting truth. Truth is symmetric under permuting atoms.

This translates directly to 'nominal' logic and programming principles.

Nominal techniques are the theory of symmetric programming on names.

Syntax-with-binding vs. metamathematical reasoning

This can be applied in various ways.

When Christian Urban introduced Nominal Isabelle in his talk, he described nominal techniques as being for syntax-with-binding.

Yes. We can apply symmetric programming on names to program on syntax-with-binding.

I see another remit.

I am developing nominal techniques for symmetric metamathematical reasoning with names.

I want to see people using 'nominal' languages tomorrow, like we use first- and higher-order logic today.

So I designed nominal rewriting (with Fernández), nominal algebra (with Mathijssen), and permissive-nominal logic (with Dowek). All are based on nominal terms.

Nominal rewriting and nominal algebra

The syntax of nominal terms:

$$r ::= a \mid \pi{\cdot}X \mid [a]r \mid \mathsf{f}(\mathsf{r},\ldots,\mathsf{r})$$

- a is an atom—the same atom we built into our universe.
- $\pi{\cdot}X$ is a permutation acting on a variable X—the π comes from the symmetry of our universe under permutation (it allows us to be very abstract about what the symbol X ranges over).
- $[a]r$ is an atoms-abstraction—this corresponds to generating a local fresh atom.
- $\mathsf{f}(\mathsf{r},\ldots,\mathsf{r})$ is a term-former as in first-order syntax.

Nominal rewriting and nominal algebra

Nominal rewriting is a theory of rewriting (directed equality) on nominal terms.

Nominal algebra is a theory of algebra ('ordinary' equality) on nominal terms.

These are both powerful (metamathematical) languages. E.g. they can axiomatise the λ-calculus and first-order logic.

Permissive-nominal logic (PNL) extends this story to a first-order framework, with $\forall X$ and $\exists X$. We use PNL to axiomatise arithmetic. That is another story.

Nominal rewriting, nominal algebra

A nominal rewrite rule is basically a pair $l \to r$. For example:

$$a \notin fa(X) \ \vdash \ \lambda([a](Xa)) \to X$$

A nominal algebra axiom is basically a pair $l = r$. For example:

$$a \notin fa(X) \ \vdash \ \lambda([a](Xa)) = X$$

So far, so good.

The condition $a \notin fa(X)$ is a freshness side-condition, more usually written $a\#X$. This too comes out of the ambient 'nominal' universe.

Nominal rewriting, nominal algebra

We go from rules and axioms to rewrites and derivable equalities by taking the least transitive reflexive α-equivalence relation such that:

- Rewriting, assuming an axiom $l \to r$:
  $s \equiv C[s']$ and $s' =_\alpha \pi\cdot(l\theta)$ and $C[\pi\cdot(r\theta)] =_\alpha t$ imply $s \to t$.
- Equality, assuming an axiom $l = r$:
  $s =_\alpha C[\pi\cdot(l\theta)]$ and $C[\pi\cdot(r\theta)] =_\alpha t$ imply $s = t$.

(One nice thing about this paper is our simplified presentations of nominal rewriting and algebra.)

π is a permutation. θ is a substitution. $=_\alpha$ is α-equivalence; this is the native notion of equality on nominal terms. $\equiv$ is syntactic identity.

Rewriting is geared towards traversing a term and generating rewrites efficiently. Algebra is geared towards generating derivable theories.

Nominal rewriting, nominal algebra

However, rewriting turns out to be expensive.

The reason for this is that it uses equivariant matching;

- $s \equiv C[s']$ and $s' =_\alpha \pi\cdot(l\theta)$ and $C[\pi\cdot(r\theta)] =_\alpha t$ imply $s \to t$.
- $s =_\alpha C[\pi\cdot(l\theta)]$ and $C[\pi\cdot(r\theta)] =_\alpha t$ imply $s = t$.

Finding π is expensive, as was observed in "The complexity of equivariant unification" by Cheney in 2004.

This matters for practical applications. If we want to decide equalities up to a theory, one standard way is to orient the equalities and try to rewrite to normal form.

We want it to be cheap to rewrite terms.

Nominal rewriting, nominal algebra

Also, rewriting and equality do not match up!

For example in the rule $a\#X \vdash X \to \mathsf{f}(\mathsf{X})$—if a is fresh for X then rewrite X to $\mathsf{f}(\mathsf{X})$—the term X in the empty freshness context does not rewrite to $\mathsf{f}(\mathsf{X})$.

The empty freshness context provides no fresh a for X. Nominal rewriting does not allow you to 'just generate' fresh atoms.

Nominal algebra does. There is a rule

$$\frac{\Delta, a\#X \vdash s = t \quad (a \text{ not in } s, t)}{\Delta \vdash s = t}\ (\mathbf{fr})$$

which lets you 'expand the freshness context'.

Permissive-nominal techniques eliminate this issue; I do not see it as an essential theoretical issue. But it certainly matters for practical applications.

Closed rules

Closed terms were introduced in "Nominal rewriting systems" by myself and Maribel in 2004.

Intuitively, a term r is closed when if we freshen all X and a in r to $X^{\prime}$ and $a^{\prime}$, to obtain a 'freshened' term $r^{\prime}$, then $r^{\prime}$ unifies with r.

a is not closed; there is no substitution on variables X making $a = a^{\prime}$ for some fresh $a^{\prime}$.

Atoms are not variables. They cannot be substituted for; only permuted.

X is closed; the substitution $[X^{\prime}{:=}X]$ makes X equal to $X^{\prime}$.

$[a]X$ is closed; $[X{:=}(a^{\prime}\ a){\cdot}X^{\prime}]$ makes $[a]X$ equal to $[a^{\prime}]X^{\prime}$; we are allowed to assume $a^{\prime}\#X$. The punchline of the paper is:

> *If we restrict to rewrites and equalities between closed rules then nominal rewriting becomes cheap and we do not have to generate fresh atoms so nominal rewriting is equal to nominal equality.*

In summary

- Nominal techniques = Symmetric metaprogramming on names as data.
- Nominal rewriting and nominal algebra instantiate this idea to rewriting and equational reasoning.
- They don't match up (nominal rewriting is a bit too weak). Furthermore, rewriting is too expensive (for n atoms, there are $n!$ permutations to consider).
- If we restrict to an already-identified syntactic subclass of closed terms—then this all goes away and everything lines up perfectly.

Implications to implementation

If you design a metamathematical framework and write some axioms down, then if those axioms involve only closed nominal terms then you can begin to try to decide equalities via common normal forms.

Normal forms do not have to exist. That depends on the rules. But this is just as in first-order rewriting, and is not a new issue.

Should we restrict all nominal terms to be closed? This is probably too enthusiastic.

Theories for first-order logic and the λ-calculus are closed. But, theories for the π-calculus are not closed. Kristoffer Rose has examples of natural specification arising in compiler design that are not closed.

Being closed is a nice case but is not the only one of interest.

Thank you for listening!