Metamathematics based on nominal terms: first-order logics over nominal sets

Murdoch J. Gabbay

Thanks to Paul Levy and Achim Jung

April 19, 2011

I promise not to give a hard talk

It's the end of a long day.

Thank you for staying this long—I'll try not to make you regret it.

There will be almost no technical details in this talk.

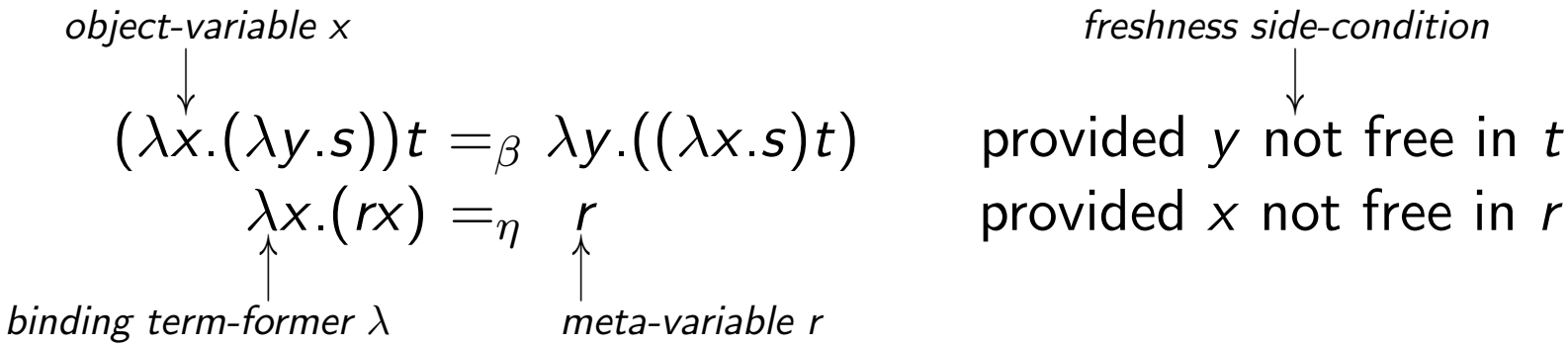
(If you want technical details, you can have them. Just find me and ask.)

I welcome questions, so long as they don't involve getting into technical details.

I want you to come away from the next 20 minutes with some feeling for the overall picture of what I'm trying to do, and why.

What does a nominal meta-language look like?

Suppose you want to specify the λ-calculus. Informally, we write this:



There's a characteristic pattern here: two levels of variable (x and r), freshness side-conditions, and binding term-formers.

Nominal terms make these things formal, so it's natural to try to base logics on nominal terms and use them as meta-mathematical specification languages.

Nominal terms / nominal sets

That's a fancy way of saying: we can try to use nominal terms to write mathematical proofs.

Nominal terms / nominal sets

Nominal terms come with a semantics: nominal sets. (In fact, the nominal sets came first and the nominal terms came later.)

I won't talk much about nominal sets right now.

Just remember this: we have the term language (nominal terms) and the denotation (nominal sets).

All we need is a good predicate language, and we're sorted.

That was pretty much the state of play in 2004.

How do we use nominal terms to do mathematics?

We need a predicate language, some proof theory, soundness and (if possible) completeness, and perhaps a few case studies.

The predicate language I currently propose is permissive-nominal logic (PNL) (developed 2009-, first presented PPDP 2010; joint work with Gilles Dowek; journal version in preparation; see also "Nominal terms and nominal logics" survey paper).

PNL builds on nominal algebra (developed 2005-, first presented in CANS workshop 2005; joint work with Aad Mathijssen), itself based on nominal rewriting (developed 2004-, joint work with Maribel Fernández).

How do we use nominal terms to do mathematics?

Proof-rules of PNL are virtually identical to those of first-order logic (FOL).

Term language is nominal terms. Term-formers can bind ($\lambda[a]r$). It has α-equivalence $[a]r = [b](b\ a)\cdot r$, two levels of variable (a and X), and freshness conditions (as a kind of typing condition).

PNL has universal quantification for meta-variables. So we can formalise statements like 'for all terms r' or 'forall predicates ϕ'. You can finitely axiomatise λ-calculus, first-order logic, and arithmetic in PNL.

Applications similar to higher-order logic (HOL), but it's not based on functions. Quine called second-order logic 'set theory in disguise'. PNL is much more first-order.

How do we use nominal terms to do mathematics?

A translation of PNL into HOL is possible and has been submitted for publication.

The projection of PNL to HOL is really a map from nominal sets to ordinary sets, that encodes names and binding as raising (cf. Levy & Villaret, Dowek & Gabbay).

The symmetry structure of names is lost, and this is reflected in the translation. What PNL really is, is FOL+names+symmetry.

Nominal sets are different from ordinary sets; names and binding do not 'equal' functions; what gets lost is symmetry and 'equivariance' properties—exactly those parts that are needed for α-renaming, which is where this line of research began.

Meta-mathematics based on nominal terms

Let's formalise the specification we saw, in nominal algebra (the equality fragment of PNL):

$$\begin{array}{ll} b\#Y \vdash & (\lambda[a]\lambda[b]X)Y = \lambda[b]((\lambda[a]X)Y) \\ a\#X \vdash & \lambda[a](Xa) = X \end{array}$$

$\beta\eta$-equivalence is a nominal algebra (Gabbay & Mathijssen 2008-2010).

Substitution for X and Y is capturing subject to freshness side-conditions.

We can derive $\lambda[a](ba) = b$ because a is fresh for b, but we cannot derive $\lambda[a](aa) = a$ because a is not fresh for a.

Description language

Nominal algebra was developed in 2005. PNL was developed in 2009.

Why the four-year gap?

The reason is binding! If we are not careful, we lose α-equivalence. Look at this:

$$\vdash [a]X$$

We can't α-rename a to a fresh b, because there is no fresh b. This makes it hard to see what a syntax with 'forall X' looks like.

In $\forall X.\phi([a]X)$ it is not guaranteed that we can α-convert a.

Permissive-nominal techniques

The solution (a solution?) is to split names into two halves, $\mathbb{A}^<$ and $\mathbb{A}^>$, and to view freshness side-conditions as typing or sorting properties of unknowns. This is what permissive-nominal terms add.

So for instance

$$[a]X^{\mathbb{A}^<} =_\alpha [b](b\ a)\cdot X^{\mathbb{A}^<} \qquad \text{where } b \notin \mathbb{A}^<.$$

Applications of PNL

I see PNL being used in applications e.g. like Isabelle/Pure is used today. That's a while away.

But there's no reason not to use PNL today to, say, teach undergraduate maths courses. We've stated and proved correct axioms for many of the major calculi in use.

You don't need to mention nominal sets if you don't want to, and PNL only makes more formal what we do informally anyway.

The future

Aside from a PNL theorem-prover and seeing textbooks written in PNL (like textbooks today are written in FOL, HOL, or ZFA, however implicitly), there are also several theoretical challenges:

- Concrete nominal models of PNL theories like λ-calculus or FOL.
- Applications to meta-programming or types, and computational aspects of nominal sets (e.g. recent interest in type systems for nominal terms, nominal automata, and nominal games).
- Better understanding of meta-variables.

The future

I will conclude with a technical teaser. I think that meta-variables (in the sense of nominal terms) are infinite streams of distinct atoms, possibly with finitely many other elements.

X is actually $(x_1, x_2, x_3, \ldots)$.

Going 'into the meta-level' is actually 'gaining access to larger streams of data'.