Permissive nominal logic and all that

Murdoch J. Gabbay
Joint work with Gilles Dowek

Thanks to Eike Ritter

September 3, 2010

Nominal techniques

Nominal techniques were introduced by myself and Pitts in a 1999 LICS paper.

See 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:

- nominal sets,
- atoms and atoms-abstraction,
- permutations,
- equivariance,
- the $И$-quantifier, and
- 'nominal' abstract syntax.

Nominal techniques

In my thesis the mathematical ideas above were used to separate α-equivalence from abstract syntax.

Names and α-binding became properties of (nominal) sets and we could use nominal sets to define inductive datatypes of abstract syntax with binding.

I won't talk about that application in this talk. See the nominal Isabelle datatypes package by Urban and others, for an extensive implementation.

A gripe on terminology, appropriate for the BLC audience

Some authors use ‘nominal logic’ as a catch-all phrase for the collection of tools listed above and for the systems derived from them.

I deprecate this, not least because for me, a ‘logic’ refers to some kind of derivation system.

I call the ensemble of ideas above nominal techniques.

Nominal techniques in the literature

Dominic Mulligan keeps an online bibliography of 'nominal' papers. He's indexed over a hundred.

The word 'nominal' is used independently by a group of mostly Italians to refer to π-calculi-like languages and HD-automata. It emerges that HD automata were also nominal in the sense of my thesis. (Or: nominal sets are a natural generalisation of HD automata.)

We had been doing the same thing.

Nominal techniques in the literature

Marcelo Fiore and others developed a presheaf approach that was also basically the same thing. He published it in the same conference, LICS'99.

Alexander Kurz and others have recently come up with a first-order presentation of nominal sets. Dale Miller and others have their own version of the И-quantifier and talk about 'nominal constants'.

Cheney, Levy and Villaret, and Dowek and myself have connected nominal unification and higher-order pattern unification; higher-order patterns are, up to a quadratic expansion of syntax, also nominal.

Thsi list is not exhaustive.

Nominal techniques in the literature

In retrospect, different people have all been doing the same thing at about the same time.

Back then these connections were not apparent. What Pitts and I did was considered iconoclastic and quite risky.

But I think we accidentally hit on something important—and it wasn't just about abstract syntax.

Nominal sets and the foundations of mathematics

Something special about what Pitts and I did was that we made a connection with set-theoretic urelemente and thus with the foundations of mathematics.

This is useful because from there, it was very natural to move to languages including λ-calculi and first-order logic.

Permissive-nominal logic (PNL) is an outcome of that process and belongs to a family of increasingly sophisticated systems.

More on that shortly.

What makes nominal techniques special: names as data

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.

Here's a slogan: names are data.

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, we do not just get a new datatype. We also get a new meta-theory. We can reflect this meta-theory back into the logics and programming languages we design to manipulate names.

Uses of names

Names appear in abstract syntax as variable symbols. The first and still most important application of nominal techniques is to programming on abstract syntax.

Names also appear in informal mathematical specifications. Permissive-nominal logic is a first-order language for capturing informal mathematical specifications faithfully in a first-order language with names (it can also program on abstract syntax).

Names appear elsewhere, e.g. as channel names in 'nominal' calculi, or as references, or as arbitrary/secret information in games or in security protocols.

Names appear as markers for 'holes' in proof-search and other situations where we have incomplete information.

Permissive-nominal logic . . .

. . . is the outcome of a thread of research:

- nominal unification (with Urban and Pitts),
- nominal rewriting (with Fernández),
- nominal algebra (with Mathijssen),
- and finally permissive-nominal logic (with Dowek).

Informal specifications

Here are some informal specifications ('informal' in the sense of being written in rigorous English):

- If $x \notin fv(t)$ then $\lambda x.(tx) = t$.
- If $x \notin fv(\Gamma)$ then $\Gamma \vdash \phi$ implies $\Gamma \vdash \forall x.\phi$.
- If $y \notin fv(t)$ then $\int_x t = \int_y t[y/x]$.
- If for every ϕ, $\phi[0/z]$ and $\forall z.\phi \Rightarrow \phi[z{+}1/z]$—then $\forall x.\phi$.

In what first-order logic can these informal specifications can be finitely represented?

Yes: permissive-nominal logic.

Atoms, permission sets, permutations

Fix two disjoint countably infinite set of atoms $\mathbb{A}^<$ and $\mathbb{A}^>$. Write $\mathbb{A} = \mathbb{A}^< \cup \mathbb{A}^>$.

$a, b, c, \ldots$ will range over distinct atoms (we call this the permutative convention).

A permission set has the form $(\mathbb{A}^< \cup A) \setminus B$ where $A \subseteq \mathbb{A}^>$ and $B \subseteq \mathbb{A}^<$ are finite. S, T, and U will range over permissions sets.

A permutation is a bijection π on atoms such that

$$\mathit{nontriv}(\pi) = \{a{\in}\mathbb{A} \mid \pi(a){\neq}a\}$$

is finite (we say π has finite support). π will range over permutations.

Syntax of permissive-nominal logic (PNL)

For each permission set S fix a countably infinite set of unknowns of that sort and permission set.

X, Y, Z will range over distinct unknowns. Write $p(X)$ for the permission set of X.

Recall the informal specification

$$\textit{If } x \notin \textit{fv}(t) \textit{ then } \lambda x.(tx) = t.$$

In PNL the variable symbol x is modelled by an atom a. The meta-variable t is modelled by an unknown X. The condition $x \notin \textit{fv}(t)$ is modelled by a permission-sorting condition $a \notin p(X)$. Any α-renamings are handled using permutative renamings of atoms.

$\lambda x.(tx) = t$ is modelled by an equality between permissive-nominal terms introduced below, in which λ and application are just term-formers.

Terms and propositions

Terms r and propositions ϕ are defined as follows:

$$r ::= a \mid (r, \ldots, r) \mid \mathsf{f(r)} \mid \mathsf{[a]r} \mid \pi\cdot\mathsf{X}$$
$$\phi ::= \bot \mid \phi \Rightarrow \psi \mid \mathsf{P(r)} \mid \forall \mathsf{X}.\phi$$

There's a sort system that I won't worry about in this talk.

Note that this is just an extension of first-order logic.

The variables of this logic are X. The atoms give us names as data, so we have atoms a, and also abstraction $[a]r$ and α-renaming $\pi\cdot X$.

Recall the informal specification "If $x \notin fv(t)$ then $\lambda x.(tx) = t$".
A PNL axiom for it is:

$$\forall X.\lambda([a]\mathsf{app(X, a)}) = \mathsf{X} \quad \text{where} \quad \mathsf{a} \notin p(\mathsf{X})$$

I'll show more theories in a moment but first I want to talk about free atoms.

Free atoms in PNL

Define free atoms $fa(r)$ by:

$$\begin{array}{rr} fa(\pi{\cdot}X) = \pi{\cdot}p(X) & fa([a]r) = fa(r) \setminus \{a\} \\ fa(\mathrm{f}(\mathrm{r})) = fa(r) & fa((r_1, \ldots, r_n)) = \bigcup fa(r_i) \\ fa(a) = \{a\} & \end{array}$$

A central innovation of PNL is the use of permission sets $S = (\mathbb{A}^< \setminus A) \cup B$ for $A, B \subseteq \mathbb{A}$ finite. Recall that $p(X)$ is a permission set.

So $fa(r)$ is always infinite, and so is its complement $\mathbb{A} \setminus fa(r)$. We can always pick a fresh name for r.

We can always α-convert a in $[a]r$ to fresh b to obtain $[b](b\ a){\cdot}r$.

We can also α-convert unknowns X, as usual.

Some example theories: Substitution

Note that these axiomatisations are finite and first-order (no infinite axiom-schemes). Note the use of atom-abstraction in terms (impossible in first-order logic). But it's all first-order, too.

$r[a{\mapsto}t]$ is sugar for sub([a]r, t).

(**sub**var)	$\forall X.$	$\mathrm{var}(a)[a{\mapsto}X]$	$\approx X$
(**sub**#)	$\forall X, Z.$	$Z[a{\mapsto}X]$	$\approx Z$
(**sub**succ)	$\forall X', X.$	$\mathrm{succ}(X')[a{\mapsto}X]$	$\approx \mathrm{succ}(X'[a{\mapsto}X])$
(**sub***op*)	$\forall X'', X', X.$	$(X'' \mathbin{op} X')[a{\mapsto}X]$	$\approx (X''[a{\mapsto}X] \mathbin{op} X'[a{\mapsto}X])$ $(op \in \{+, *, \dot{\Rightarrow}, \dot{\approx}\})$
(**sub**$\dot{\forall}$)	$\forall X, Z.$	$(\dot{\forall}([b]Z))[a{\mapsto}X]$	$\approx \dot{\forall}([b](Z[a{\mapsto}X]))$
(**subid**)	$\forall X.$	$X[a{\mapsto}\mathrm{var}(a)]$	$\approx X$

$a \in \mathbb{A}^<$ and $b \notin \mathbb{A}^<$. The permission set of X'', X', and X is equal to $\mathbb{A}^<$. The permission set of Z is equal to $(b\ a){\cdot}\mathbb{A}^<$.

Some example theories: First-order logic

$$
\begin{array}{ll}
(\dot{\Rightarrow}) & \forall Z', Z.\ \epsilon(Z' \mathrel{\dot{\Rightarrow}} Z) \Leftrightarrow (\epsilon(Z') \Rightarrow \epsilon(Z)) \\
(\dot{\forall}) & \forall Z.\ \big(\epsilon(\dot{\forall}([a]Z)) \Leftrightarrow \forall X.\epsilon(Z[a{\mapsto}X])\big) \\
(\dot{\bot}) & \epsilon(\dot{\bot}) \Rightarrow \bot \\
(\dot{\approx}) & \forall X', X.\ X' \approx X \Rightarrow \epsilon(X' \mathrel{\dot{\approx}} X)
\end{array}
$$

Here Z' and Z have sort o and permission set $\mathbb{A}^<$; X' and X have sort ι and permission set $\mathbb{A}^<$; and $a \in \mathbb{A}^<$.

Some example theories: arithmetic

$$
\begin{array}{ll}
(\mathbf{PS0}) & \forall X.\ \mathsf{succ}(\mathsf{X}) \approx 0 \Rightarrow \bot \\
(\mathbf{PSS}) & \forall X', X.\ \mathsf{succ}(\mathsf{X}') \approx \mathsf{succ}(\mathsf{X}) \Rightarrow \mathsf{X}' \approx \mathsf{X} \\
(\mathbf{P{+}0}) & \forall X.\ X + 0 \approx X \\
(\mathbf{P{+}succ}) & \forall X', X.\ X' + \mathsf{succ}(\mathsf{X}) \approx \mathsf{succ}(\mathsf{X}') + \mathsf{X} \\
(\mathbf{P{*}0}) & \forall X.\ X * 0 \approx 0 \\
(\mathbf{P{*}succ}) & \forall X', X.\ X' * \mathsf{succ}(\mathsf{X}) \approx (\mathsf{X}' * \mathsf{X}) + \mathsf{X} \\
(\mathbf{PInd}) & \forall Z.\ (\epsilon(Z[a{\mapsto}0]) \Rightarrow \\
 & \qquad \big(\forall X.(\epsilon(Z[a{\mapsto}X]) \Rightarrow \epsilon(Z[a{\mapsto}\mathsf{succ}(\mathsf{X})]))\big) \Rightarrow \\
 & \qquad \forall X.\epsilon(Z[a{\mapsto}X]))
\end{array}
$$

All variables have permission set $\mathbb{A}^<$, and $a \in \mathbb{A}^<$.

A sketch of the paper on PNL in PPDP 2010

- Introduce permissive-nominal terms and PNL predicates.
- Axiomatise equality, substitution, first-order logic, and arithmetic.
- Prove soundness with respect to a nominal sets semantics.
- Prove by a construction on models that arithmetic as a PNL theory is correct.
- Claim that PNL is also complete for the semantics and satisfies cut-elimination (proofs written out but not polished and peer-reviewed).
- Discuss other axioms in PNL, including for inductive datatypes, for freshness, and for the И-quantifier.

The motivation behind this is to propose PNL as an extension of first-order logic that can do a lot of what higher-order logic is currently pressed into service for, but without the higher-order semantics or syntax. What makes it all work is the permissive-nominal semantics.

Sequent derivation rules of PNL

$$\frac{}{\Phi,\ \phi \vdash \phi,\ \Psi}\ (\mathbf{Ax}) \qquad \frac{}{\Phi,\ \bot \vdash \Psi}\ (\bot\mathbf{L}) \qquad \frac{\Phi,\ \phi \vdash \Psi}{\Phi,\ \pi{\cdot}\phi \vdash \Psi}\ (И)$$

$$\frac{\Phi \vdash \phi,\ \Psi \quad \Phi,\ \psi \vdash \Psi}{\Phi,\ \phi \Rightarrow \psi \vdash \Psi}\ (\Rightarrow\mathbf{L}) \qquad \frac{\Phi,\ \phi \vdash \psi,\ \Psi}{\Phi \vdash \phi \Rightarrow \psi,\ \Psi}\ (\Rightarrow\mathbf{R})$$

$$\frac{\Phi,\ \phi[X{:=}r] \vdash \Psi \quad (fa(r){\subseteq}p(X))}{\Phi,\ \forall X.\phi \vdash \Psi}\ (\forall\mathbf{L}) \qquad \frac{\Phi \vdash \phi,\ \Psi \quad (X{\notin}fV(\Phi,\Psi))}{\Phi \vdash \forall X.\phi,\ \Psi}\ (\forall\mathbf{R})$$

Connection with nominal logic

Pitts published on ‘nominal logic’. The subtitle of the paper was ‘a first-order theory of names and binding’.

Nominal logic is a first-order axiomatisation sound for nominal sets; a fragment of Fraenkel-Mostowski set theory, to be precise.

Nominal sets = Fraenkel-Mostowski set theory, by the way.

This matters. It is a logic. PNL has a proof-theory and a distinctive syntax. It puts names and binders into terms.

Permissive-nominal logic’s name is inspired by the name of nominal logic, but PNL is a syntax and derivation rules with a sound and complete semantics attached.

Both PNL and nominal logic take semantics in nominal sets; they are ‘the same’ in this sense. This is a weak equivalence; lots of things take semantics in nominal sets.

. . . and all that

Where is this all going?

PNL could and should be implemented. It might be a useful mathematical foundation for theorem-provers alternative to HOL.

I can imagine a nominal version of Isabelle based on PNL instead of HOL.

This matters because HOL unification is undecidable, yet heuristically theorem-provers seem to stay in decidable fragments.

It's just my opinion, but I suspect this is because they are really doing PNL.

. . . and all that

If you want names in your favourite system (games/graphs/domains/. . .) take its representation in sets and transfer to nominal sets. This may be non-trivial; nominal sets provide a methodology.

I and others are doing research along those lines.

Semantics of unknowns, semantics of holes

However, PNL raises another interesting question. It has two levels of variable: atoms a and unknowns X.

Atoms a map to themselves. Unknowns don't; they are variables and are given a denotation via a valuation.

Unknowns are functional variables. They are functionally abstracted by the valuation (in the semantics which I haven't shown you).

But in PNL we don't just have a theory of names. We have a theory of names with holes. The unknowns are also holes.

Holes

Holes are interesting. They turn up e.g. in studies of modules, incompleteness, and proof-search.

I have spent the past year trying to construct a nominal-style semantics for the holes in permissive-nominal logic. But I didn't do it for the sake of PNL, which already has a good denotation.

I did it to extend the 'nominal' story from names to names-and-holes.

I call it going from level 1 to level 2. We are taking the nominal ideas and compounding them with natural, non-functional, structure.

The idea that seems to work is this: if names are atoms then unknowns are well-orderings on infinite lists of atoms.

We identify X with a well-ordering on the permission set $p(X) = fa(X)$.

Holes

Why is it intuitively reasonable to identify X with a well-ordering of its free atoms?

The permutation action $\pi \cdot X$ just permutes the well-ordering. $\pi \cdot X$ is the atoms in the order given by X, permuted.

Also, a non-obvious fact is that well-orderings behave like atoms in that they can be permuted, chosen fresh, and even abstracted. You can build $[X]r$; $[X]r$ behaves like a 'big' version of $[a]r$ and has all the right properties.

Conclusions

Nominal techniques are a rich source of interesting mathematical questions.

They might also support some things that are happening in the logic and semantics of computer science, wherever names and incompleteness arise.

Because of the symbolic and computational nature of many of the interesting questions in computer science, that is very common.

PNL is a 'nominal' first-order logic. I have mentioned syntax-with-binding, holes, and a finite and elementary axiomatisation of arithmetic.