A (partial) survey of nominal techniques

Murdoch J. Gabbay

September 18, 2013

This will will be a non-technical talk.

If you come away from this with some idea of what nominal techniques are, why they are useful, and what they do, then I will be very happy.

Though I have written slides for this talk, I am happy to follow the audience. So feel free to ask questions, including stupid questions and technical questions.

What are nominal techniques?

Based on nominal sets by Gabbay and Pitts.

(Connections to Fraenkel-Mostowski sets, Schanuel Topos & set theories with urlemente or atoms $a, b, c, \ldots$.)

A nominal set $\mathcal{X}$ is like an ordinary set. It has elements: $x \in \mathcal{X}$.

But $x \in \mathcal{X}$ also has a supporting set supp(x) of atoms.

This is a new concept; examples on next slide.

Note: names and atoms are used synonymously in this talk.

Concrete examples of nominal structures

Syntax up to α-equivalence is a nominal set $\mathcal{S}$. Elements $t \in \mathcal{S}$ are terms. Variable symbols are atoms a. Support supp(t) is 'free variables of t':

- $\forall a.a = b$.
- $\lambda a.\lambda b.fab$.
- `typedef struct { int x, y; } Position;`

Pointers are nominal. Pointers are atoms. Support of a machine state is 'memory usage':

```
int j; int *ptr; int main(void) { j = 2; ptr = &j; }
```

Process calculi are nominal. Channel names are atoms. Support of a process is 'free channels': $\nu a.\bar{a}b$.

Examples of nominal structures

Tensors are nominal. Atoms a, b, are indexes ranging over basis vectors. $K_a^b \delta_a^b$.

Similarly for nonces in security protocols, and so on.

Nominal sets capture abstractly what is common to these concrete examples: $x \in \mathcal{X}$ and supp(x) + abstract properties.

In summary: nominal techniques are about atoms.

Atoms (see later discussion) are characterised by symmetry, and parametricity.

So nominal techniques are the study of

- atoms + extra structure, which lead to
- structures with symmetries, and thus to
- parametricity (e.g. over choice of atom, but there's more).

Nominal theory

Phenomena in different fields are identified as instances of a general theory.

This theory can be stated, verified, implemented, and automated.

What does that mean?

Numbers and names

Consider numbers.

- You don't expect to reimplement arithmetic every time you encounter a number. If you calculate on apples today, you don't expect to implement an apples-library distinct from the oranges-library you implemented yesterday.
- You assume a single theory of numbers which matches what you studied in school & uni.
- You expect abstract interfaces, efficient implementations, and guarantees of performance and accuracy.
- You expect the hard work to have been done once and for all.

Nominal techniques try to do this for names, whose essence is symmetry and parametricity. Contrast with numbers, whose essence is well-orderableness.

Q. What is a name? (Clue: symmetry)

What is the essential feature that makes an atom or name be an atom or name?

Permutations: can permute atoms. E.g. $(b\ a) : \mathbb{A} \to \mathbb{A}$ swaps b and a.

Compare and contrast numbers $\mathbb{N}$ with names $\mathbb{A}$.

- Numbers are totally ordered: given i and j we can always ask whether $i < j$.
- Names are totally permutable: given a nominal element x (which might be a term, but not necessarily) we can always form $(b\ a)\cdot x$. Permutation action.

These properties are opposite.

- Numbers are the canonical totally ordered set.
- Names are the canonical totally symmetric/permutable set.

What is a name? How is a name different from a number?

If x is a valid program or predicate, $(b\ a)\cdot x$ is another valid program or predicate with the same meaning except that b and a are swapped. We call this equivariance.

Thus, these are all 'the same' up to permuting names:

- $\lambda a.\lambda b.ab$
- $\lambda a'.\lambda b'.a'b'$
- $\lambda b.\lambda a.ba$

This is not true of numbers: $1 \leq 2$ is true and $2 \leq 1$ is false.

Nominal sets

- Ordinary sets are built using numbers $0, 1, 2, 3, \ldots$.
- Nominal sets are built using numbers and atoms $a, b, c, \ldots$.

- Set theorists: von Neumann cumulative hierarchy $V = \bigcup_\alpha \mathcal{P}^\alpha(\varnothing)$.
- Type theorists use $\tau ::= \{\bot, \top\} \mid \mathbb{N} \mid \tau \to \tau$.

For this talk, these are just fancy ways of building collections of orderable data (cf. axioms of choice: every set can be well-ordered).

Atoms are totally unordered and symmetric up to permutations. We can permute atoms and compare them for equality . . . and that's all.

Effects of culture

We are slaves to the foundational assumptions of the giants on whose shoulders you stand.

Existing foundations—$\bigcup_\alpha \mathcal{P}^\alpha(\varnothing)$ or $\tau ::= \cdots \mid \tau \to \tau$—are positional.

This forces us, slaves that we are, to adopt 'positional'/'ordered'/'numbered' repesentations. Symmetry properties cannot be represented.

Specifications are more difficult because they must include explicit 'swapping' lemmas on a theory-by-theory basis.

Example: inductive syntax with binding. When we go under a binder, decomposing $\lambda a.s$ into a and s, we must symmetrically choose a fresh name for a. Ordinary sets cannot directly express that 'the choice of a is symmetric' or 'choose some/any fresh name a for the bound atom' (cf. Gabbay-Pitts NEW quantifier).

Effects of culture

Stronger logics are often required to compensate (e.g. HOL instead of FOL or algebra).

Models are larger, more complex, not transferrable between theories, and tend to be polluted by ‘exotic elements’.

Some natural operations cannot be expressed.

Effects of culture

It's a serious mathematical issue, and a serious engineering issue too.

Nominal techniques are based on Fraenkel-Mostowski set theory (FM sets).

A different foundation.

It has significantly different engineering properties, because FM sets admit names.

Summary

- Names are ubiquitous.
- They are absent from ordinary sets and their characteristic symmetry properties are not present.
- This is a lacuna, it matters, and it is a research opportunity.
- Nominal sets = ordinary sets + names. We get all our existing programming and logic + new ones for atoms and their symmetries.
- Nominal sets come with a high-level mathematical vocabulary and theory giving uniform libraries and operations, just as numbers (i.e. ordered datatypes) do for the asymmetric data we are all used to.
- Low-hanging fruit. Yum!

Applications of nominal techniques

This list is only partial:

- Polish group studying automata with symmetries. This cuts down on search space and permits study of 'infinite' automata that are symmetric up to groups of permutations, and thus finitely representable.
- Similar work by Italians studying symmetries of process calculi.
- Nominal Isabelle: an implementation of the 'nominal' ideas discussed above, in Isabelle. Currently developed in King's College, London. Many applications.
- Theoretical work in Cambridge led by Andrew Pitts amongst others.
- Me.

My own research interests

Let me pick out two recent highlights:

- Nominal algebra, lattices, topology, and duality for first-order logic and the λ-calculus.
- Permissive-nominal logic: first-order reasoning over nominal sets + finite first-order axiomatisation of arithmetic + translation to HOL.

My own research interests

In practice, names usually come equipped with domain-specific properties.

- Variables in logic have a substitution action $[a \mapsto t]$, can be quantified $\forall a$, and exist in a lattice of truth-values $x \leq y$.
- Variables in the λ-calculus have a substitution $[a \mapsto s]$ action and can be λ-abstracted $\lambda a.s$.
- Indexes in tensors come with properties describing summation over indexes $K_a^b \phi$.
- Channel names come with properties describing channel-passing.

. . . and so on.

These properties can be equationally specified.

Nominal algebra and duality

Axiomatise a name-carrying system such as first-order logic.
Example axioms (in nominal algebra):

$$a \notin \mathrm{supp}(y) \Rightarrow \forall a.x = x$$
$$a \notin \mathrm{supp}(y) \Rightarrow \forall a.(x \vee y) = (\forall a.x) \vee y$$

Then build lattice and topological models of these axioms. Prove soundness, completeness, and topological duality.

This is algebraic logic and duality theory, but using nominal sets.

Next, do the same for the λ-calculus, the π-calculus, tensor calculi, and so on.

Such things are known for non-name-carrying systems such as Boolean algebra and distributive lattices. Doing it for name-carrying systems without nominal techniques is very difficult, or impossible.

Permissive-nominal logic (PNL)

Axiomatise a name-carrying system such as arithmetic. Example axioms (in permissive-nominal logic):

$$a \notin \mathrm{supp}(Y) \vdash \forall X.\forall a.X = X$$
$$a \notin \mathrm{supp}(Y) \vdash \forall X, Y.\forall a.(X \vee Y) = (\forall a.X) \vee Y$$

Note that the axiomatisation is finite: so we have a finitely specification of Peano arithmetic in (nominal) first-order logic.

Thus, PNL is stronger than non-nominal first-order logic, but it is weaker than higher-order logic (HOL).

Give translation of PNL into HOL identifying exactly what fragment of HOL PNL corresponds to. This works, but what goes missing in the translation is . . . name-symmetries (beautiful result).

Some research opportunities

- More on duality.
- Dependent type theories: reconcile symmetry (in particular name-generation) with dependent types.
- Categories. Categories are inherently positional (just like ZF sets and HOL).
  Typed σ-algebras are a candidate nominal generalisation.
- Coalgebras and behaviour. Nominal techniques for inductive types are well-understood; coinductive types, less so.
- Parametricity: nobody understands in abstract terms what the category of nominal sets actually is.