Nominal techniques: present, and future prospects

Murdoch J. Gabbay, Heriot-Watt University, Scotland

TU/E, Eindhoven, Netherlands
Monday, 14 November 2007

Thanks to Adam Koprovski

Nominal techniques

Thanks to Aad Mathijssen for a productive collaboration developing Nominal Algebra. I wish him well for the future.

Thanks to Jan-Friso Groote; this time two years ago I was living here in Eindhoven. I remember that time fondly.

Thanks also to MohammadReza Mousavi. We never did manage to finish writing that paper but I hope we'll keep trying.

Nominal techniques

What are nominal techniques? Good question. When I finished my PhD (in 2001), ‘nominal techniques’ meant a semantics for ‘syntax trees up to α-equivalence’.

Syntax trees do have a semantics — trees. It’s not terribly complicated.

The semantics of syntax trees up to α-equivalence is important because so many proofs need induction on syntax (of terms, of derivations). They require renaming of bound variables. I studied how to do that.

This led to the discovery of equivariance and the Gabbay-Pitts $И$-quantifier and model of α-abstraction.

Nominal techniques

I propose that nominal techniques are/can be used for two things:

- A theory of name-like entities in syntax and semantics.
- A theory of truth at the informal meta-level.

It turns out that there is substantial (and unanticipated) overlap between these two things.

At this point I can expect some listeners to be confused. So let's get some overview of some logics.

What's good for what?

Propositional logic, a theory of truth and implication:

$$it\ is\ a\ belgian\ beer \Rightarrow it\ is\ good$$

Modal logic, a theory of truth and implication over time, space, or possibility:

$$\Box(it\ is\ a\ belgian\ beer \Rightarrow it\ is\ good)$$

First-order logic, a theory of truth and implication over elements (of some domain of discourse):

$$\forall x.(beer(x) \wedge belgian(x) \Rightarrow good(x))$$

What's good for what?

Second-order logic, a theory of truth over sets of elements:

$$\forall U.\Big((\forall x \in U.beer(x) \wedge belgian(x)) \Rightarrow excellent(U)\Big)$$

Higher-order logic, a theory of truth over functions.

Nominal techniques: a formal theory of truth in the informal meta-theory of the logics above — of many 'formal methods' in computer science.

It turns out that names play a large part in this informal meta-theory. The model of names provided by my thesis can be applied here, to give a formal theory of truth in the informal meta-level of the logics above.

Nominal algebra

Nominal algebra is one part of this formal theory. I can show you a few axioms.

Nominal algebra axioms for substitution

$$
\begin{array}{rlrcl}
(\mathbf{var}{\mapsto}) & & a[a{\mapsto}T] & = & T \\
(\#{\mapsto}) & a\#X \vdash & X[a{\mapsto}T] & = & X \\
(\Rightarrow{\mapsto}) & & (P \Rightarrow Q)[a{\mapsto}T] & = & (P[a{\mapsto}t]) \Rightarrow (Q[a{\mapsto}t]) \\
(=\!{\mapsto}) & & (U = V)[a{\mapsto}T] & = & (U[a{\mapsto}t]) = (V[a{\mapsto}t]) \\
(\forall{\mapsto}) & b\#T \vdash & (\forall[b]P)[a{\mapsto}T] & = & \forall[b](P[a{\mapsto}T]) \\
(\mathbf{sub}{\mapsto}) & b\#T \vdash & X[b{\mapsto}U][a{\mapsto}T] & = & X[a{\mapsto}T][b{\mapsto}U[a{\mapsto}T]] \\
(\mathbf{ren}{\mapsto}) & b\#X \vdash & X[a{\mapsto}b] & = & (b\,a) \cdot X
\end{array}
$$

$s[a{\mapsto}t]$ is shorthand for $\mathsf{sub}([a]s, t)$.

Nominal algebra axioms of first-order logic

Usual rules for boolean algebra, plus axioms for substitution, plus

$$
\begin{array}{lll}
(\mathbf{Q1}) & \forall[a]P \Rightarrow P[a{\mapsto}T] & = \top \\
(\mathbf{Q2}) & \forall[a](P \wedge Q) \Leftrightarrow \forall[a]P \wedge \forall[a]Q & = \top \\
(\mathbf{Q3}) & a\#P \vdash \forall[a](P \Rightarrow Q) \Leftrightarrow (P \Rightarrow \forall[a]Q) & = \top \\
 & & \\
(\mathbf{E1}) & U = T \wedge P[a{\mapsto}T] \Rightarrow P[a{\mapsto}U] & = \top \\
(\mathbf{E2}) & T = T & = \top
\end{array}
$$

Nominal algebra axioms for the λ-calculus

$$
\begin{array}{rrrrcl}
(\mathbf{var}{\mapsto}) & & \vdash & a[a{\mapsto}X] & = & X \\
(\#{\mapsto}) & a\#Z & \vdash & Z[a{\mapsto}X] & = & Z \\
(\mathbf{app}{\mapsto}) & & \vdash & (Z'Z)[a{\mapsto}X] & = & (Z'[a{\mapsto}X])(Z[a{\mapsto}X]) \\
(\mathbf{abs}{\mapsto}) & b\#X & \vdash & (\lambda b.Z)[a{\mapsto}X] & = & \lambda b.(Z[a{\mapsto}X]) \\
(\mathbf{ren}{\mapsto}) & b\#Z & \vdash & Z[a{\mapsto}b] & = & (b\,a)\cdot Z
\end{array}
$$

Abstraction

Abstraction is a feature of nominal terms. $[a]X$ is 'abstract a in X'. X is a level 2 name, a is a level 1 name. Substitution for X does not avoid capture by abstraction by a level 1 name. So $([a]X)[a/X] \equiv [a]a$.

This gives X the behaviour of a meta-variable, and a the behaviour of an object-level variable.

$[a]X$ is formal syntax, and it behaves like the following expressions at the informal meta-level: $\lambda x.t$ or $\forall x.\phi$ or $\nu x.P$.

α-equivalence

The freshness side-condition $a\#X$ lets us write formal 'not free in' side-conditions which behave like the following expressions at the informal meta-level: 'a is not free in t', 'a is not free in ϕ', 'a is not free in P'.

A theory of α-equivalence is built in to nominal algebra, via permutations like $(b\ a)$ (swap b and a). So

$$b\#X \vdash [b](b\ a) \cdot X = [a]X$$

exactly captures α-equivalence. It turns out that this is a special case of

$$a\#X,\ b\#X \vdash (b\ a) \cdot X = X.$$

α-equivalence

Substitute $[a]X$ for X

$$a\#[a]X,\ b\#[a]X \vdash (b\ a)\cdot[a]X = [a]X$$

and use the facts that

- $(b\ a)\cdot[a]X \equiv [b](b\ a)\cdot X$,
- $a\#[a]X$ always, and
- $b\#[a]X$ implies $b\#[a]X$.

Permutations are ‘naturally capture-avoiding’; they distribute through binders as pairs, renaming also abstracted level 1 names. This use of permutations is one of the original surprises of nominal techniques.

Nominal techniques

To what extent does nominal algebra illustrate my claims about name-like entities?

Much of the power of nominal algebra comes from the use of two levels of name, a and X.

Freshness side-conditions allow us to assert formal relationships between these names; $a\#X$ or 'a is fresh for X'.

Substitution actions make the names behave like variables, but as we have seen substitution can be axiomatised; no further apparatus is needed to include it.

Nominal techniques

Permutations (bijections on names) are built-in. These are used to capture α-equivalence of level 1 abstraction in the presence of level 2 variables;

$[a]X =_\alpha [b]X$ is wrong because if we substitute a for X we get $[a]a =_\alpha [b]a$.

Recall previous axiom. $[b](b\ a) \cdot a \equiv [b]b$ and $[b]b =_\alpha [a]a$ is not wrong.

Nominal algebra

Nominal algebra scores because it is ϵ away from informal practice.

The informal meta-level has two levels of variable, it also has freshness side-conditions.

The nominal algebra theory of α-equivalence does not appear in the informal meta-level — it is the missing ingredient necessary to make it into a formal theory.

Nominal algebra

α-equivalence and unification of nominal terms is decidable.

Equality of the nominal theory of substitution is also decidable; decidability of unification is an open problem.

The ability of nominal algebra to specify theories, such as logics and λ-calculi (and also process calculi), is well-established.

The usefulness of nominal techniques in reasoning about syntax with binding (the original application in my thesis) is now as well-accepted as anything can be in this field.

I believe that what we have seen so far is only a fraction of what can be achieved.

Future work: a theory of contexts

Derivation trees are syntax. Quantifier rules abstract the fresh name they generate in the subtree of which they are a head (usually, we think of the fresh name as ‘being free’ in the subtree).

An ‘incomplete derivation’ should be representable using the same nominal terms technology we used, for example, to treat the λ-calculus. $\lambda a.X$ is ‘morally’ the same as $(\forall aR)X$, an unknown derivation concluding in a $\forall$-intro rule:

$$\frac{\begin{array}{c}\vdots\\ X\end{array}}{\vdash \forall a.(a = a)}\,(\forall\mathbf{R})$$

Future work: cylindric techniques

Nominal algebra is in some sense a variant of the so-called ‘cylindric algebras’, commonly applied to first-order logic and also to the λ-calculus.

I would prefer to see more communication with this community. I would like to see their powerful theorems translated to the nominal setting.

I suspect that they will become more powerful in the translation because the nominal terms syntax is more expressive than that of the usual cylindric syntaxes (permutations, freshness side-conditions), and because nominal style semantics assume finite support.

Future work: a theory of incomplete derivations

I know that Geuvers has been interested in this, and I am aware of other groups working on incomplete proof to model top-down proof-search.

I wish that somebody would get on with it and carry out this immediate and relatively easy application of nominal technology — preferably with me.

Future work: a nominal algebra theorem prover

Building a new theorem-prover is a very large undertaking but it might be worth doing.

It could make a good topic for an able and hard-working PhD student.

Nominal algebra demonstrates how close we can get to the informal meta-level using a logic based on nominal terms. It now seems to me inevitable that somebody, somewhere, will have a go at this.

Future work: two-and-a-halfth-order logic

Quantify over the level 2 variables in nominal algebra, internalise the freshness side-conditions using implication.

Pick an axiom at random:

$$b\#T \vdash (\forall[b]P)[a{\mapsto}T] = \forall[b](P[a{\mapsto}T])$$

becomes

$$\forall T.\, b\#T \Rightarrow \forall P.\, \Big((\forall b.P)[a{\mapsto}T] \Leftrightarrow \forall b.(P[a{\mapsto}T])\Big).$$

Future work: denotational models

Nominal techniques are now in use in Oxford (nominal games), in Microsoft Research in Cambridge (nominal pointers), in Cambridge Computer Laboratory (original application to inductive reasoning on syntax; pencil and paper proofs), in Munich (original application to inductive reasoning on syntax; nominal Isabelle), and also in Pisa and Udine (nominal models of HD automata). And of course there's Aad here at TU/e, and perhaps also Mohammad and Michel.

There's more; like everybody else I commit the sin of forgetting to mention half the people in the field.

I am pleased that the ideas started back in 1999 are finding such application.

Conclusions

Nominal techniques are a theory of names.

The examples of nominal games, nominal semantics for pointers, and nominal models of HD automata, demonstrate that name-like entities are not confined to syntax-with-binders, theorem-provers, and variable symbols.

The example of nominal algebra shows that we can make formal logics out of these ideas.

Conclusions

I have written many Introductions/Conclusions to papers (accepted!) and project proposals (not yet!) to the effect that:

Name-like entities are pervasive in computer science. There is a strong case to treat their theory as an independent field of theoretical computer science.

The theory of numbers is independent of arithmetic algorithms (but related to it). The theory of the λ-calculus is independent of the theory of programming languages (but related to it).

In a similar way, the theory of names is independent of variables, pointers, channel names, and so on (but related to them all).

Conclusions

I have emphasised nominal algebra in this talk, in honour of the work Aad did here at TU/e.

In the field of logic, nominal techniques score highly because — as for example nominal algebra demonstrates — we can get ‘ϵ away from informal practice’.

Conclusions

There is a strong dose of what I like to call ‘ergonomic’ motivation in computer science.

Andrew Pitts calls it ‘ϵ away from informal practice’.

De Bruijn wrote

> “I think that in formalizing mathematics, and in particular in preparing mathematics for justification, it is usually elegant as well as efficient to do everything in the natural way.”

(Checking mathematics with computer assistance, AMS, 1991)

Conclusions

A pure mathematician once said to me ‘I don’t know why you [computer scientists] bother with the λ-calculus; just use combinators’. But combinators are not ergonomic; people prefer C to SKI.

Likewise, some people say ‘don’t bother designing better formal methods; just hire better programmers’. But this is wrong.

History shows that careful and mathematically rigorous attention to informal computational practice leads to new mathematics, which can lead (after a delay, which is nevertheless shrinking) to better tools.

Modern databases are one example. Boolean algebra is another.

Conclusions

It is therefore inevitable that somebody, somewhere, will get research added value from the new and underexploited ergonomics which nominal techniques offer in logic, automated proof, and elsewhere.

There is also research capital to be made in denotations, e.g. the nominal game theory or nominal pointers work.

I do not know who will do what, nor what the finished product will look like. But the research capital is there, waiting, for us — for anybody — to pick it up.

In case somebody asks about incomplete derivations

$$\dfrac{\dfrac{A \Rightarrow B \Rightarrow C \quad [A]^i}{B \Rightarrow C} \qquad \dfrac{?}{B}}{\dfrac{C}{A \Rightarrow C}\, i}$$

$$\dfrac{\dfrac{A \Rightarrow B \Rightarrow C \quad [A]^i}{B \Rightarrow C} \qquad \dfrac{A \Rightarrow B \quad [A]^i}{B}}{\dfrac{C}{A \Rightarrow C}\, i}$$

Capturing substitution necessary for Curry-Howard with incomplete derivations.

(Example borrowed from [Jojgov, TYPES 2002]).