Nominal Algebra:

a NEW mathematics of variables

Murdoch J. Gabbay, Heriot-Watt University, Scotland

LMU, Munich, Germany
Wednesday 6 December 2006

Joint work with: Mathijssen, Rota Buló, Marin

Nominal algebra

It is possible to look at nominal algebra in two ways:

- Viewpoint 1. A proof-system and associated semantics which look like universal algebra (the logic and semantics of equality) but which admit quantifiers in a particularly intuitive manner.
- Viewpoint 2. A logic for a semantics in which names are first-class citizens.

Let me explain.

Motivation according to Viewpoint 1

- $\lambda a.t$ — untyped λ-calculus (LAM)
- $\forall a.\phi$ — first-order predicate logic (FOL)
- $\int \mathrm{f}\, da$ — school/kindergarten

These expressions all have in common:

- Object-level variables a.
- Meta-level variables t, u, ϕ, or f.
- Operators (or term-formers or function-symbols) λ, $\forall$, $\int$.

Nominal Terms

Nominal terms are a syntax inductively generated by

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

Here:

- $a, b, c, \ldots \in \mathbb{A}$ are atoms.
- $X, Y, Z, \ldots \in \mathbb{V}$ are unknowns.
- f, g, ... are term-formers or operators etcetera (depends on whether we're thinking in syntax or semantics).
- $[a]t$ is an abstraction.
- π is a permutation. I'll come to it later. Please ignore it for now.

Nominal Algebra representations

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

The a look like object-level variable symbols — the ones that get abstracted:

- $\lambda a.t$ — untyped λ-calculus (LAM)
- $\forall a.\phi$ — first-order predicate logic (FOL)
- $\int \mathrm{f}\, d\, a$ — school/kindergarten

Nominal Algebra representations

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

Abstraction $[a]t$ represents abstraction:

- $\lambda[a]t$ — untyped λ-calculus (LAM)
- $\forall[a]\phi$ — first-order predicate logic (FOL)
- $\int([a]\mathrm{f})d$ — school/kindergarten

Nominal Algebra representations

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

'Logical operators' such as λ, $\forall$, $\int\ d$, and so on, are represented by operators:

• $\lambda[a]t$	untyped λ-calculus (LAM)
• $\forall[a]\phi$	first-order predicate logic (FOL)
• $\int([a]\mathrm{f})d$	school/kindergarten

Nominal Algebra representations

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

X is another kind of variable, representing an unknown entity like t, u, and ϕ:

- $\lambda[a]X$ — untyped λ-calculus (LAM)
- $\forall[a]P$ — first-order predicate logic (FOL)
- $\int d([a]X)$ — school/kindergarten

$\lambda[a]X$, $\forall[a]X$, and $\int d([a]X)$ are nominal algebra terms.

(Sorry for the notational jiggery-pokery with $\int$.)

λ-calculus theory LAM

Assume term-formers λ, app, and Σ. Sugar $app(t, u)$ to tu (here t and u range over nominal terms!). Sugar $\Sigma([a]t, u)$ to $t[a{\mapsto}u]$.

Algebra is a logic of equality. Therefore assertions should take the form $t = u$. A theory is a collection of assertions we call axioms.

LAM has one axiom:

$$(\beta) \qquad\qquad (\lambda[a]Y)X = Y[a{\mapsto}X]$$

Theory of first-order logic FOL

Bit more complex:

$$P \Rightarrow Q \Rightarrow P \;=\; \top \qquad (P \Rightarrow Q) \Rightarrow (Q \Rightarrow R) \Rightarrow (P \Rightarrow R) \;=\; \top$$

$$\neg\neg P \Rightarrow P \;=\; \top \qquad \forall[a](P \wedge Q) \Leftrightarrow \forall[a]P \wedge \forall[a]Q \;=\; \top$$

$$\bot \Rightarrow P \;=\; \top \qquad a\#P \;\vdash\; \forall[a](P \Rightarrow Q) \Leftrightarrow (P \Rightarrow \forall[a]Q) \;=\; \top$$

$$T = T \;=\; \top \qquad \forall[a]P \Rightarrow P[a \mapsto T] \;=\; \top$$

$$\top \Rightarrow P \;=\; P \qquad U = T \wedge P[a \mapsto T] \Rightarrow P[a \mapsto U] \;=\; \top$$

(Assume term-formers $=, \forall, \Rightarrow, \bot, \Sigma$, and sugar.)

Freshness assertions $a\#t$

Read $a\#t$ as

- 'a does not occur unabstracted in t', or
- 'a is fresh for t'.

There is a logic to freshness. It's pretty straightforward:

Freshness derivation rules

$$\frac{}{a\#b}\,(\#\mathbf{ab}) \qquad \frac{a\#t_1 \ \cdots \ a\#t_n}{a\#\mathsf{f}(t_1,\ldots,t_n)}\,(\#\mathbf{f})$$

$$\frac{}{a\#[a]t}\,(\#[]\mathbf{a}) \qquad \frac{a\#t}{a\#[b]t}\,(\#[]\mathbf{b}) \qquad \frac{\pi^{-1}(a)\#X}{a\#\pi X}\,(\#\mathbf{X})$$

Permutations

I suppose you really want to know about π now.

Here's one more theory. A theory of α-equivalence:

$$(\alpha) \qquad\qquad b\#X \Rightarrow [b](b\ a)X = [a]X.$$

Since $b\#X$ we can intuitively read $(b\ a)X$ as '$X[a\mapsto b]$'. Then the rule above is the usual α-renaming rule.

We need permutation in order to rename atoms, to avoid capture etc.

Permutations

It is possible to base a mathematical theory on renamings $[a{\mapsto}b]$ instead of permutations $(a\ b)$. I was doing that with Martin last year.

However invertibility loses no power and has better properties.

I.e. $(a\ b)[a]X \equiv [b](a\ b)X$, whereas perhaps $([a]X)[b{\mapsto}a] \equiv ([a'](X[a{\mapsto}a'][b{\mapsto}a]))$ where we assume $a'\#X$ — it's not actually impossible, but it is more complicated.

Limited brainpower $\Rightarrow$ invest it wisely.

Equality derivation rules (easy)

$$\frac{}{t = t}\,(\mathbf{refl}) \qquad \frac{t = u}{u = t}\,(\mathbf{symm}) \qquad \frac{t = u \quad u = v}{t = v}\,(\mathbf{tran})$$

$$\frac{t = u}{C[t] = C[u]}\,(\mathbf{cong}) \qquad \frac{a\#t \quad b\#t}{(a\;b) \cdot t = t}\,(\mathbf{perm})$$

For example

$$\dfrac{\dfrac{}{a\#b}\,(\mathbf{\#ab}) \qquad \dfrac{}{a\#b}\,(\mathbf{\#ab})}{[a]a = [b]b}\,(\mathbf{perm}) \qquad\qquad (b\ a)\cdot[b]b \equiv [a]a$$

$$\dfrac{\dfrac{b\#X}{b\#[a]X}\,(\mathbf{\#[]a}) \qquad \dfrac{}{a\#[a]X}\,(\mathbf{\#[]a})}{[b](b\ a)X = [a]X}\,(\mathbf{perm}) \qquad\qquad (b\ a)\cdot[a]X \equiv [b](b\ a)X$$

Here $\equiv$ is syntactic identity.

Semantics

The theories of first-order logic and of the λ-calculus permit reasoning exactly like informal practice. See the papers [oneaah], [capasn], and [nomsst].

When we would α-rename, we instead use a permutation. When we would assume $a \notin \mathsf{fn}(\phi)$, we instead write $a\#P$.

Semantics (Viewpoint 2)

Do not think that this is trivial.

Something very interesting has happened in Nominal Algebra: a, b, c, $\ldots$ are populating the semantics. Yet they can also be renamed and abstracted.

In Nominal Algebra names are first-class citizens, represented by atoms.

We can give them special properties just by imposing axioms.

For example substitution.

Theory of substitution SUB

$$\mathsf{f}(X_1, \ldots, X_n)[a\mapsto T] = \mathsf{f}(X_1[a\mapsto T], \ldots, X_n[a\mapsto T])$$

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

$$a[a\mapsto T] = T$$

$$a\#X \vdash X[a\mapsto T] = X$$

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

Picture of what we have done

- $\equiv$ is syntactic identity $\quad [a]a \not\equiv [b]b$
- $=$ (with (α)) is α-equivalence $\quad b\#X \vdash [b](b\ a)X = [a]x$
- $=_{\mathsf{SUB}}$ is substitution $\quad b\#Y \vdash Y[b{\mapsto}X] = Y$
- $=_{\mathsf{LAM}}$ is $\alpha\beta$-equivalence $\quad (\lambda[a]a)b = b$
- $=_{\mathsf{FOL}}$ is logical equivalence $\quad (\forall[a](a = a)) = \top$

See [oneaah,oneaah-jv,capasn,nomsst].

Some really beautiful maths (soundness, completeness, sequent rules, cut-elimination, decidability, etc).

Theory of substitution SUB (the return!!)

I'd like to discuss SUB. I think that SUB is very important.

What are the properties of names $a, b, c, \ldots$?

- They are atomic: 'a' has no internal structure.
- They may be renamed and abstracted.
- They are not das Ding an sich: '$a = b$' is just false.

Nominal algebra with (α) is a logical theory of names.

Names vs. variables

What are the properties of variables $x, y, z, \ldots$?

- They are atomic: 'x' has no internal structure.
- They may be renamed and abstracted.
- They may be substituted for.
- They are not das Ding an sich: '$x = y$' may be true or false (depending on what we substitute for them).

... and they turn up in most formal languages of note, from first-order logic to JAVA.

Nominal algebra plus SUB is a theory of variables!

Names vs. variables

So: A variable is a name with a substitution action.

And now for the punchline: let's consider the class of all models of SUB. Is it cartesian closed? Because if it is, then we can design λ-calculi and thus programming-languages in which variables are (names with a subsitution action and so are) first-class citizens of the underlying domain.

Models of substitution

A model $\mathbb{X}$ is:

- An underlying (nominal) set $|\mathbb{X}|$.
- An interpretation function assigning to each atom a some element $|a| \in |\mathbb{X}|$.
- An interpretation of substitution is a map $([\mathbb{A}]|\mathbb{X}|) \times \mathbb{X} \rightarrow \mathbb{X}$.

... validating the axioms of SUB.

Models of substitution

Oops. I forgot to mention that if X is a nominal set then $[\mathbb{A}]X$ is the abstraction-set, and if $t \in X$ then $[a]t \in [\mathbb{A}]X$.

$[\mathbb{A}]X$ has underlying (normal) set $(\mathbb{A} \times X)/\sim$, where $(a, t) \sim (a', t')$ when either

- $(a, t) = (a', t')$ or
- $a' \# t$ and $t' = (a'\ a)t$.

Compare with (α).

Here $a' \# t$ is a semantic version of freshness judgements. Its construction goes way back [gabbay:thesis,newaas,newaas-jv] and is not in the scope of this talk.

Models of substitution

Models of SUB are a cartesian-closed category.

Let $\mathbb{X}, \mathbb{Y}, \mathbb{Z}$ range over such models. Suppose $t, t' \in \mathbb{X}$, $u, u' \in \mathbb{Y}$, and $f \in \mathbb{X} \Rightarrow \mathbb{Y}$.

Clearly $(t, u)[a{\mapsto}(t', u')] = (t[a{\mapsto}t'], u[a{\mapsto}u'])$.

The problem is to define a substitution action on f a function.

$$a\#t \vdash (f[a{\mapsto}g])t = (ft)[a{\mapsto}gt]$$
$$|a| = \lambda(t{\in}\mathbb{X}).|a|.$$

Very simple. Arguably, very powerful.

Lambda-abstraction, as a function

$\mathsf{abstract} \in \mathbb{X} \to \mathbb{A} \to \mathbb{X} \to \mathbb{X}$ is defined by

$$\mathsf{abstract}(t, a) = \lambda t'.t[a \mapsto t'].$$

$\mathsf{abstract}$ is a function that takes an argument and a name and λ-abstracts that name in its argument.

Contexts, as functions

Any $C \in \mathbb{X} \to \mathbb{X}$ behaves very much like C in $C[t]$, since C may bind in its argument.

It's a particularly general notion of context though: abstract could be written as λ-.-.

Access to names

We have full access to names. For example we can write the freshness test $\#$ as a function f such that:

- $ft = 0$ if $a\#t$.
- $ft = 1$ if $\neg(a\#t)$.

(Assuming two 'normal' elements $0, 1 \in |\mathbb{X}|$.

This is not possible in the λ-calculus: fx and fy may differ, but not because x is called 'x'. Likewise ft may differ from ft', but not because $\mathsf{fn}(t) \neq \mathsf{fn}(t')$.

Names in this category are like variables (they can be abstracted) but they behave a little bit like pointers too.

Conclusion

Nominal algebra is a logic in which names are first-class citizens. It permits reasoning in a very intuitive style on languages with binding, such as FOL and the λ-calculus. Freshness, permutations, a, and X correspond to fn, α-renaming, x, and t/ϕ.

This also inspired mathematics of independent interest.

One example is a (nominal) algebraic characterisation of variable as name+substitution.

I am excited about the implications for designing programming languages.

Thank you very much for listening.