Permissive nominal terms

Murdoch J. Gabbay, www.gabbay.org.uk

Thursday 25 June 2009

Joint work with Gilles Dowek and with Dominic Mulligan
Thanks to the CILC organisers: Fabrizio and Marco

Permissive nominal terms' motivation

Let us recall first-order logic:

$\forall x.\phi$ means "ϕ holds for all values that x can take".

Let us recall the $\forall$-quantifier introduction rules:

$$\frac{\Gamma \vdash \phi \ \ (x \text{ not free in } \Gamma)}{\Gamma \vdash \forall x.\phi}\,(\forall\mathbf{R}) \qquad \frac{\Gamma,\ \phi[x := t] \vdash \psi}{\Gamma,\ \forall x.\phi \vdash \psi}\,(\forall\mathbf{L})$$

Here $\phi[x := t]$ is 'replace x by t in ϕ in a capture-avoiding manner'.

Permissive nominal terms' motivation

$\phi[x := t]$ is definable by clauses which include the following:

$$x[x := t] = t \qquad (\forall y.\psi)[x := t] = \forall y.(\psi[x := t]) \quad (y \text{ not free in } t)$$

We can write this also as follows:

$$\frac{}{x[x := t] = t} \qquad \frac{\forall y.(\psi[x := t]) = \psi' \quad (y \text{ not free in } t)}{(\forall y.\psi)[x := t] = \psi'}$$

So look at this ...

Permissive nominal terms' motivation

So look at this:

$$\frac{\Gamma \vdash \phi \;\; (x \text{ not free in } \Gamma)}{\Gamma \vdash \forall x.\phi}\,(\forall\mathbf{R}) \qquad \frac{\Gamma,\ \phi[x := t] \vdash \psi}{\Gamma,\ \forall x.\phi \vdash \psi}\,(\forall\mathbf{L})$$

$$\frac{}{x[x := t] = t} \qquad \frac{\forall y.(\psi[x := t]) = \psi' \quad (y \text{ not free in } t)}{(\forall y.\psi)[x := t] = \psi'}$$

These are all Horn clauses. The definitions above, which are typical of such definitions, are written in a logic programming language.

The rules are syntax-directed and the relations they determine are, by design, computable in linear time.

There is a pattern here which I will now briefly discuss.

Two levels of variable, α-abstraction, freshness

Two levels of variable, level 1 ('object-level') and level 2 ('meta-level'):

$$\frac{\forall y.(\psi[x := t]) = \psi' \quad (y \text{ not free in } t)}{(\forall y.\psi)[x := t] = \psi'}$$

α-abstraction and freshness side-conditions:

$$\frac{\forall y.(\psi[x := t]) = \psi' \quad (y \text{ not free in } t)}{(\forall y.\psi)[x := t] = \psi'}$$

Two levels of variable, α-abstraction, and freshness side-conditions.

Capturing substitution

There's another feature: a capturing substitution. If we write

> "let's set ψ to be $\mathsf{P}(x)$, t to be z, and ψ' to be $\forall y.(\mathsf{P}(z))$"

then we obtain the instance

$$\frac{\forall y.(\psi[x := t]) = \psi' \quad (y \text{ not free in } t)}{(\forall y.\psi)[x := t] = \psi'} \Longrightarrow \frac{\forall y.(\mathsf{P}(x)[x := z]) = \forall y.(\mathsf{P}(z))}{(\forall y.\mathsf{P}(x))[x := z] = \forall y.(\mathsf{P}(z))}$$

We must verify the side-condition 'x not free in z', which is valid, so the instance is valid.

Two levels of variable, α-abstraction, freshness side-conditions, and capturing substitution.

The informal meta-level

Two levels of variable, α-abstraction, freshness side-conditions, and capturing substitution.

This is the structure of the informal discourse of mathematics, in which simple definitions on syntax are defined, like derivability and substitution mentioned above:

In this informal discourse ‘at the meta-level’, things like

- ‘syntax and derivability of first-order logic’ or
- ‘λ-calculus syntax and its operational semantics’ or
- ‘the language of functions and integration, as in $\int f(x)\,dx$’,

and many other formal languages with names and binding, are informally but rigorously given meaning, often using Horn clauses as above.

Typical domains of discourse

When first-order logic was designed, the typical domain of discourse was numbers. Put simply: first-order logic was designed for physics. Higher-order logic extends this to functions from numbers to numbers (i.e. real numbers), and further.

In computer science we're not reasoning (just) about numbers! We're reasoning about logics and computation. A typical domain of discourse is a syntactic structure which displays name-binding structure. From this, the two-level structure of nominal terms, reflecting object- and meta-level.

(We can Gödel encode everything into numbers, but we don't like to because that introduces a level of emulation. Distinction between computation and computability; proof-theory and derivability, etc.)

Nominal terms introduction

No syntax with these features existed until nominal terms were introduced (Theoretical Computer Science, 2004, co-authored by Christian Urban, Andrew Pitts, and myself).

Nominal terms are sound with respect to a semantics in Fraenkel-Mostowski set theory developed in my thesis.

I find the semantics beautiful. I will not tell you about it today! What is relevant here, is that nominal terms syntax is also backed up by a rigorous semantics.

Syntax shapes thought. It's amazing what a difference a change of syntax can make.

Nominal terms quick outline

Nominal terms have two levels of variable: level 1 variables atoms $a, b, c, \ldots$ and level 2 variables unknowns $X, Y, Z, \ldots$.

α-abstraction is handled by a dedicated abstraction construct $[a]X$.

Intuitively, $[a]X$ expresses what '$\lambda x.t$', '$\forall x.\phi$' or '$\int t\,dx$' all have in common.

But there is a catch. I will give some intuition about it; for more details, see the paper.

Nominal terms: a catch

There is a catch: X represents 'any term' (just like t represents 'any term' and ϕ represents 'any predicate' in the examples above).

X can be instantiated to anything. Because it can be instantiated to anything, it potentially has all atoms free. Thus, we cannot α-convert a in $[a]X$ because there is no fresh b for X.

If we tried to α-convert a to some level 1 variable b, then somebody else might choose to instantiate X to something mentioning b.

$$\overset{\text{'}\lambda x.t\text{'}}{[a]X} \quad \overset{\text{fresh } b \text{ for } X}{\Longrightarrow} \quad \overset{\text{'}\lambda y.t[x:=y]\text{'}}{[b](b\ a)\cdot X} \quad \overset{X:=b}{\Longrightarrow} \quad \overset{\text{'}\lambda y.x\text{'}}{[b]a}$$

$(b\ a)$ is the 'swapping', mapping b to a and a to b. More in a moment.

Summary of motivation (again)

I want to run over the motivations for this once more.

The informal meta-level of computer science is not like that of physics. We're not manipulating solely numbers, functions on numbers, and relations on numbers; we're often manipulating trees with names and binding, and relations and functions on them.

We're developing a syntax for expressing the informal meta-level.

We can build on top of this theories of validity, derivability, operational semantics, denotational semantics, abstract machines, soundness, completeness, complexity, implementation, and so on.

The informal meta-level of computer science seems to exhibit a two-level structure as discussed above.

Atoms and term-formers

We now come to some definitions.

Fix a countably infinite set $\mathbb{A}$ of atoms (level 1 variables).

We use a permutative convention that $a, b, c, \ldots$ will range over distinct atoms.

Fix a set of term-formers. $\mathsf{f}, \mathsf{g}, \mathsf{h}$ will range over distinct term-formers.

Examples of term-formers are lam, app, and forall. We need theories of logic or rewriting to make these 'do' something; for now they are just ways of building terms with suggestive names.

The comb, and permissions sorts

Call $S \subseteq \mathbb{A}$ co-infinite when $\mathbb{A} \setminus S$ is infinite.

Fix an infinite, co-infinite set $\mathsf{comb} \subseteq \mathbb{A}$.

We can build comb by fixing a bijection ι between natural numbers and $\mathbb{A}$ and setting

$$\mathsf{comb} = \{\iota(2i) \mid \text{all } i\}.$$

A permission sort has the form

$$(\mathsf{comb} \cup A_1) \setminus A_2$$

for finite sets $A_1 \subseteq \mathbb{A}$ and $A_2 \subseteq \mathbb{A}$.

S, S', T will range over permission sorts.

Unknowns and permutations

For each S fix a disjoint countably infinite set of unknowns of sort S (level 2 variables). X^S, Y^S, Z^S, will range over distinct unknowns of sort S.

If $S \neq S'$ then there is no particular connection between X^S and $X^{S'}$.

A permutation is a bijection on atoms such that $\{a \mid \pi(a) \neq a\}$ is finite. π and π' will range over permutations (not necessarily distinct).

Write id for the identity permutation such that $\mathsf{id}(a) = a$ always.
Write $(a\ b)$ for the swapping which maps a to b, b to a, and all other c to themselves.

Permissive nominal terms

Define (permissive nominal) terms by:

$$r, s, t, \ldots ::= a \mid \mathsf{f}(r_1, \ldots, r_n) \mid [a]r \mid \pi \cdot X^S$$

We write $\equiv$ for syntactic identity; $r \equiv s$ when r and s denote identical terms.

Write $\mathsf{id} \cdot X^S$ just as X^S, for short. For example, write $[a]X^S$ instead of $[a]\mathsf{id} \cdot X^S$.

Free atoms

If $S \subseteq \mathbb{A}$, define the pointwise action by:

$$\pi \cdot S = \{\pi(a) \mid a \in S\}$$

Define free atoms $\mathsf{fa}(r)$ by:

$$\mathsf{fa}(a) = \{a\} \qquad \mathsf{fa}(\mathsf{f}(r_1, \ldots, r_n)) = \mathsf{fa}(r_1) \cup \ldots \cup \mathsf{fa}(r_n)$$

$$\mathsf{fa}([a]r) = \mathsf{fa}(r) \backslash \{a\} \qquad \mathsf{fa}(\pi \cdot X^S) = \pi \cdot S$$

An intuition for $\mathsf{fa}(r)$ is 'possible free atoms after instantiation'.

Define a permutation action by:

$$\pi \cdot a \equiv \pi(a) \qquad \pi \cdot (\mathsf{f}(r_1, \ldots, r_n)) \equiv \mathsf{f}(\pi \cdot r_1, \ldots, \pi \cdot r_n)$$

$$\pi \cdot [a]r \equiv [\pi(a)](\pi \cdot r) \qquad \pi \cdot (\pi' \cdot X^S) \equiv (\pi \circ \pi') \cdot X^S$$

Capturing substitution (for level 2 variables)

A substitution θ is a function from unknowns to terms such that $\mathsf{fa}(\theta(X^S)) \subseteq S$ always.

So the permissions set S in X^S describes the 'permission' we have to instantiate X^S, namely to terms with free atoms in S.

θ will range over substitutions.

Define a substitution action by:

$$a\theta \equiv a \qquad \mathsf{f}(r_1,\ldots,r_n)\theta \equiv \mathsf{f}(r_1\theta,\ldots,r_n\theta)$$
$$([a]r)\theta \equiv [a](r\theta) \qquad (\pi{\cdot}X^S)\theta \equiv \pi{\cdot}\theta(X^S)$$

For example: $(\pi \cdot X^S)[X^S := a] \equiv \pi(a)$
$([a]X^S)[X^S := a] \equiv [a]a$

α-equivalence

Let $\pi\big|_S$ denote the partial function 'π restricted to S'. Define α-equivalence $=_\alpha$ inductively by:

$$\frac{}{a =_\alpha a}\,(=_\alpha \mathbf{aa}) \qquad \frac{r_1 =_\alpha s_1 \quad \cdots \quad r_n =_\alpha s_n}{\mathsf{f}(r_1,\ldots,r_n) =_\alpha \mathsf{f}(s_1,\ldots,s_n)}\,(=_\alpha \mathsf{f})$$

$$\frac{r =_\alpha s}{[a]r =_\alpha [a]s}\,(=_\alpha [\mathbf{a}]) \qquad \frac{(b\ a)\cdot r =_\alpha s \qquad (b \notin \mathsf{fa}(r))}{[a]r =_\alpha [b]s}\,(=_\alpha [\mathbf{b}])$$

$$\frac{(\pi\big|_S = \pi'\big|_S)}{\pi \cdot X^S =_\alpha \pi' \cdot X^S}\,(=_\alpha \mathbf{X})$$

Examples of α-equivalence

We can α-convert a in $[a]X^S$. For example,

$$[b](b\ a)\cdot X^S =_\alpha [a]X^S \quad \text{for any} \quad b \notin S.$$

Let's prove that. Suppose $b \notin S$.

$$\dfrac{\dfrac{}{((b\ a)\circ(b\ a))\cdot X^S =_\alpha X^S}\,(=_\alpha \mathbf{X})}{[b](b\ a)\cdot X^S =_\alpha [a]X^S}\,(=_\alpha [\mathbf{b}])$$

The instance of $(=_\alpha [\mathbf{b}])$ is valid because $b \notin \mathsf{fa}(X^S) = S$. The instance of $(=_\alpha \mathbf{X})$ is valid because $(b\ a)\circ(b\ a) = \mathsf{id}$.

α-equivalence

Why use permutations in α-equivalence?

Why set $[a]X^S =_\alpha [b](b\ a) \cdot X^S$ (where $b \notin S$) instead of

$$[a]X^S =_\alpha [b](a \mapsto b)X^S$$

where $(b \mapsto a)$ is a renaming which maps a to b, b to b, and all other c to c?

Permutation is naturally capture-avoiding. For example,

$$(a\ b) \cdot [a][b]\mathsf{f}(a,b) \equiv [b][a]\mathsf{f}(b,a) =_\alpha [a][b]\mathsf{f}(a,b).$$

Lemma (natural capture-avoidance): if $\pi(a) = a$ for every $a \in \mathsf{fa}(r)$, then $\pi \cdot r =_\alpha r$.

This plays a major part in soundness of $(=_\alpha [\mathbf{b}])$ and $(=_\alpha \mathbf{X})$.

Permutations vs. renaming

Renaming is not naturally capture-avoiding:

$$(a \mapsto b) \cdot [a][b]\mathsf{f}(a,b) \equiv [b][b]\mathsf{f}(b,b) \neq_\alpha [b][b]\mathsf{f}(b,b)$$

To define α-equivalence using $(a \mapsto b)$ we would need to introduce an element of capture-avoidance, e.g. so that

$$(a \mapsto b) \cdot [a][b]\mathsf{f}(a,b) \equiv [b][c]\mathsf{f}(b,c)$$

(or similar).

But we use α-equivalence in the definition of capture-avoidance, to avoid capture. A circularity is introduced.

It's not obvious, but this turns out to be a major issue.

There's maths here and I'm not telling you about it now.

Natural capture-avoidance

This is what nominal terms don't have, and permissive nominal terms do:

Lemma: For any r, there exist infinitely many b such that $b \notin \mathsf{fa}(r)$. As a corollary, for any r and a there exists infinitely many fresh b (so $b \notin \mathsf{fa}(r)$) such that for some s, $[a]r =_\alpha [b]s$.

Proof: The first part is by an easy induction on r. We consider only the case $r \equiv \pi \cdot X^S$:

- $\mathsf{fa}(\pi \cdot X^S) = \pi \cdot S$, and since S is co-infinite, so is $\pi \cdot S$. That is, infinitely many b are such that $b \notin \pi \cdot S$.

The corollary follows using $(=_\alpha [\mathbf{b}])$.

In summary

We discussed the informal meta-level. It has two levels of variable, α-abstraction, freshness side-conditions, and capturing substitution.

We looked at $[a]X$, which expresses the shared kernel of $\lambda x.t$, $\forall x.\phi$, and $\int t\,dx$. Because X can 'become any term', α-equivalence for the abstracted atom a fails, because there's nothing fresh to rename it to.

We constructed permissive nominal terms syntax. Level 2 variables have a permissions set, which is an infinite co-infinite set of atoms differing finitely from comb.

$[b](b\ a)\cdot X^S =_\alpha [a]X^S$ for any $b \notin S$.

Again, there's mathematics in the background. This syntax is the tip of an iceberg.

In summary

To what uses can permissive nominal terms be put?

Unification (developed in the paper in full detail).

Rewriting. Logic-programming (the Horn clauses mentioned earlier; see ‘nominal logic programming’ by Cheney and Urban). Algebra (equational reasoning of the informal meta-level).

Logics with level 2 quantifiers. $\forall X^S.\Phi(X^S)$. This is current work; it would not be possible without permissive nominal terms. The ‘permissive’ bit reconciles level 2 quantification with level 1 α-conversion.

This is part of a broader programme of nominal research.

Thanks for listening.