Concrete models of nominal algebra substitution. . .

. . . holey functions!

Murdoch J. Gabbay

Joint work with Samuel Rota Buló and Andrea Marin

CANS meeting, King's College London, UK, 22/6/2006

Substitution

Substitution $u[a \mapsto t]$ is generally thought of as an operation on syntax.

It seems to sit between α-equivalence

$$[a]t =_\alpha [b](b\ a)t \qquad (b \text{ fresh for } t)$$

and β-equivalence

$$(\lambda a.u)t =_\beta u[a \mapsto t].$$

Here $a, b, c, \ldots \in \mathbb{A}$ are object-level variable symbols; atoms for short.

Substitution

$$[a]t =_\alpha [b](b\ a)t \qquad (b \text{ fresh for } t)$$

Here square brackets indicate some abstractor (λ, $\forall$).

$(b\ a)$ denotes 'swap b and a'.

Since b is assumed fresh for t this is identical (up to renaming bound names) to 'replace a by b'.

However, swapping has better meta-theoretic properties.

Substitution

Why does substitution sit between α- and β-equivalence?

Obviously it underlies β-equivalence: it is mentioned in its definition.

Substitution contains some measure of α-equivalence in the first two arguments, e.g.

$$a[a{\mapsto}t] = b[b{\mapsto}t].$$

In general,

$$v[a{\mapsto}t] = ((b\ a)v)[b{\mapsto}t] \qquad (b \text{ fresh for } v).$$

Does substitution exist?

Logics and λ-calculi generally use environment models for open terms.

We fix some assignment of variable symbols to denotational elements.

We then inductively extend the assignment of variable symbols, using interpretations of the term-formers of the syntax, to all terms.

Does substitution exist?

We then inductively extend the assignment of variable symbols ... to all terms.

Variables are not in the denotation (there is no element $[\![a]\!]$ of some underlying set representing 'a', independently of arbitrarily choosing one via the assignment)!

Therefore, open terms are not in the denotation (there is no element $[\![f(a)]\!]$ of some underlying set representing '$f(a)$')!

Therefore, substitution is not in the denotation (there is no function on elements such that $[\![f(a)]\!][a \mapsto [\![2]\!]] = [\![f(2)]\!]$)!

Alternatives (semantics)

Crabbé proposed a first-order logic axiomatisation of substitution.

Feldman proposed another.

In these models, variables exist in the underlying denotation (and can be substituted for).

Semantics are therefore available for open terms.

However, these axiomatisations do not handle binding. Many interesting systems, notably the λ-calculus, are excluded (or at best only partially treated).

Alternatives (λ-calculi of explicit substitution)

Designed to measure the 'cost of calculating a β-reduction'.

This is not relevant for us today: our denotations are designed to analyse validity, not complexity.

Alternatives (syntax)

Handling abstract-syntax-with-binding in a theorem-prover or programming language is a practical issue of a certain importance.

(I don't suppose I need to work hard to convince you of that.)

Various approaches:

De Bruijn indexes.

Higher-order abstract syntax.

Nominal techniques [hurrah].

Alternatives (syntax)

Nominal techniques are based on Fraenkel-Mostowski sets, which gave (for the first time) a set-theoretic semantics to α-equivalence. Thus given a set x — any x — we can build $[a]x$.

This was later characterised abstractly as nominal sets.

A nominal set is a set-with-a-finitely-supported-permutation-action.

Given a nominal set X we can build $[\mathbb{A}]X$ with elements $[a]x$ for $a \in \mathbb{A}$ and $x \in \mathbb{X}$.

Nominal sets (detail)

A nominal set is a set $\mathbb{X}$ with a swapping action mapping $\mathbb{A} \times \mathbb{A} \times \mathbb{X}$ to $\mathbb{X}$, satisfying the equalities of a permutation action

$$(a\ a)x = x \qquad (a\ b)(a\ b)x = x \qquad (a\ b)x = (b\ a)x$$
$$(a\ b)(n\ m)x = ((a\ b)n\ (a\ b)m)(a\ b)x$$

Plus the freshness axiom

$$И b.\ b\#x.$$

Here a and b vary permutatively over atoms and m and n vary non-permutatively over atoms (so $a \neq b$ but perhaps $m \in \{a, b, n\}$).

NEW quantifier

$$И b.\ b\#x$$

Here if F is a function then $И b.\ F(b)$ is the unique value of F for 'most' (all but finitely many) b, if this exists.

$a\#x$ when $И b.\ (b\ a)x = x$, so the freshness axiom is

$$И b.\ И a.\ (b\ a)x = x.$$

It remains now to give axioms for substitution sets.

Substitution sets

$$
\begin{aligned}
a\#z &\Rightarrow z[a{\mapsto}x] = z \\
a[a{\mapsto}x] &= x \\
b\#z &\Rightarrow z[a{\mapsto}b] = (b\ a)z \\
a\#y &\Rightarrow z[a{\mapsto}x][b{\mapsto}y] = z[b{\mapsto}y][a{\mapsto}x[b{\mapsto}y]]
\end{aligned}
$$

A substitution action is a ternary function from $\mathbb{X} \times \mathbb{A} \times \mathbb{X}$ to $\mathbb{X}$.

Note the b in $[a{\mapsto}b]$ — we need to interpret atoms in $\mathbb{X}$!

Nominal algebra substitution (Gabbay Mathijssen 2006)

Axioms above derived from the following nominal algebra theory, by admitting as the only term-former substitution itself:

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

$$b\#T \Rightarrow ([b]X)[a{\mapsto}T] = [b](X[a{\mapsto}T])$$

$$\mathsf{var}(a)[a{\mapsto}T] = T$$

$$a\#X \Rightarrow X[a{\mapsto}T] = X$$

$$b\#X \Rightarrow X[a{\mapsto}\mathsf{var}(b)] = (b\,a)\cdot X$$

This theory is sound and complete with respect to the term-model (closed nominal terms mentioning only fs for which we have the first axiom).

Substitution sets

Call a set $\mathbb{X}$ with

- an interpretation of $\mathbb{A}$ (an injection from $\mathbb{A}$ to $\mathbb{X}$), and
- a substitution action,

a substitution set.

Examples

$\mathbb{A}$ has a substitution action. Interpret a by a and define:

$$x[a{\mapsto}y] = y \qquad (x = a \text{ or } x = y)$$
$$x[a{\mapsto}y] = x \qquad (x \neq a \text{ and } x \neq y).$$

$\mathbb{L}$ (finite lists $()$, (a), (a,b,c,a), ...) has a substitution action.

Interpret a by (a) and define ... well, here are two examples:

$$(a,b,c)(b{\mapsto}(a,b,c)) = (a,a,b,c,c)$$
$$(a,a,b)(b{\mapsto}(b,a)) = (a,a,b,a).$$

Examples (exploding models)

$\mathbb{L}$ (finite lists $()$, (a), (a,b,c,a), ...) has a different 'exploding' substitution action:

Interpret a by (a) and define ...well, here are two examples:

$$(a,b,c)(b \mapsto (c)) = ()$$
$$(a,a,b)(b \mapsto (b,c)) = (a,a,b,c).$$

If we try substituting an atom into a list which already mentions that atom, we 'explode' (return the empty list).

This is also a substitution action. So substitution can have computational content...

More examples

There is a very ‘simple’ model which we have all built, at one point or another:

Take untyped λ-terms

$$t ::= a \quad | \quad tt \quad | \quad \lambda a.t$$

up to $\alpha\beta$-equivalence; write $[\![\text{-}]\!]$ for equivalence classes.

Interpret a by $[\![a]\!]$. Define $[\![u]\!][a \mapsto [\![t]\!]] = [\![u[a \mapsto t]]\!]$.

This makes equivalence-classes of possibly open terms into a model of substitution.

More examples

We can encode

> ‘u, which is a function on t by virtue of application to t’

as

> $[\![ua]\!]$ for $a \not\in u$, which is a function on $[\![t]\!]$ by virtue of $[a \longmapsto [\![t]\!]]$.

More examples

A particular model of substitution can contain computation; z can be a big computation with a parameter named a, and $z[a \mapsto x]$ can stand in a very complex computational relation to z and x.

However, models of substitution are not a general semantics for computation; computational elements are not guaranteed across all models; there is no λ.

The environment model

Take an 'ordinary' set, such as $\mathbb{N}$. Write $\Rightarrow$ for function-spaces.

$$(\mathbb{A} \Rightarrow \mathbb{N}) \Rightarrow \mathbb{N}$$

is a model of substitution — provided we restrict to

$$\tau \in (\mathbb{A} \Rightarrow \mathbb{N}) \Rightarrow \mathbb{N}$$

(write τ as $\lambda\kappa{\in}(\mathbb{A}{\Rightarrow}\mathbb{N}).\tau(\kappa)$) such that:

- there exists finite $S \subseteq \mathbb{A}$ such that
- if $\kappa(a) = \kappa'(a)$ for all $a \in S$ then $\tau(\kappa) = \tau(\kappa')$.

('τ examines just a finite part of its argument.')

The environment model

Write this set $\hat{\mathbb{N}}$.

Interpret a by

$$\lambda\kappa.\kappa(a).$$

Interpret $\mu[a \mapsto \tau]$ by

$$\lambda\kappa.\mu(\kappa\{a \mapsto \tau(\kappa)\}).$$

$\kappa\{a \mapsto n\}$ maps a to n and b to $\kappa(b)$.

$\mu[a \mapsto \tau]$ acts on κ as μ does on κ' where $\kappa'(a) = \tau(\kappa)$!

The environment model

An interesting element of $\hat{\mathbb{N}}$ is $\lambda\kappa.\kappa(a) * \kappa(b)$.

This behaves like the syntax '$a * b$' — but it's semantic;

$$(a * b)[a \longmapsto 2][b \longmapsto 3] = \lambda\kappa.6.$$

Function models

The set of finitely-supported functions between two substitution sets $\mathbb{X}$ and $\mathbb{Y}$, is a substitution set $\mathbb{X} \Rightarrow \mathbb{Y}$.

We can write $h[a \longmapsto f]$.

Holey functions! Substitution sets form a cartesian closed category (CCC). We can program in them.

Function models

Fix $h, f \in \mathbb{X} \Rightarrow \mathbb{Y}$ and $a \in \mathbb{A}$.

Define:

$$h[a \mapsto b]x = ((b\ a)h)x \qquad \text{if } b\#h$$
$$(h[a \mapsto f])x = И a'.\ (((a'\ a)h)x)[a' \mapsto f\,x] \qquad \text{otherwise.}$$

Here b in $h[a \mapsto b]$ is the interpretation of b in $\mathbb{X} \Rightarrow \mathbb{Y}$, which is $\lambda x.b$.

Function models

This is loosely but more plainly specified by:

$$h[a \mapsto b]x = ((b\ a)h)x \qquad \text{if } b\#h$$

$$(h[a \mapsto f])x = (hx)[a \mapsto fx] \qquad \text{if } a\#x, f$$

(Some calculations)

How is the specification of the last slide related to that of the one before?

Suppose $a\#x$, f and a' is fresh ($a'\#h, f, a, x$).

Then $a\#(hx)[a\mapsto fx]$ and so

$$(a'\ a)x = x \quad (a'\ a)h = h \quad (a'\ a)f = f$$

and so

$$\begin{aligned}(a'\ a)((hx)[a\mapsto fx]) &= ((a'\ a)h(a'\ a)x)[a'\mapsto(a'\ a)f(a'\ a)x] \\ &= ((a'\ a)h)x)[a'\mapsto fx].\end{aligned}$$

A subtlety

Ad hoc change terminology!

A substitution set is confluent when it satisfies

$$a\#y \Rightarrow z[a{\mapsto}x][b{\mapsto}y] = z[b{\mapsto}y][a{\mapsto}x[b{\mapsto}y]]$$

and non-confluent otherwise.

It is the non-confluent substitution sets which form a CCC. Even if $\mathbb{X}$ and $\mathbb{Y}$ are confluent, $\mathbb{X} \Rightarrow \mathbb{Y}$ need not be.

So f can have a and b in its support, and make two quite irreconcilable decisions depending on which gets instantiated (by a substitution) first.

Examples

$t \mapsto [a]t$ is a function in $\mathbb{X} \Rightarrow [\mathbb{A}]\mathbb{X}$ mapping x to $[a]x$.

If $\mathbb{X}$ is the ground term model for the syntax of the λ-calculus (call it Λ) then we can compose with term-former λ in $[\mathbb{A}]\mathbb{X} \Rightarrow \mathbb{X}$ to obtain $t \mapsto \lambda a.t \in \Lambda \Rightarrow \Lambda$.

Examples

If $x \in \mathbb{X}$ then

$$\text{‘the substitution } \big[a \longmapsto x\big]\text{’}$$

can be represented as

$$\lambda z.z\big[a \longmapsto x\big] \in \mathbb{X} \Rightarrow \mathbb{X}.$$

So we can represent substitution explicitly in the model.

Examples

Elements can be considered as functions over their atoms, yet also as data.

Thus we can write $x[a \longmapsto y]$ and $y[a \longmapsto x]$; both are correct. In a typed λ-calculus if yx is correct then xy is not; by virtue of the types once is 'master' and the other is 'slave'.

With substitution instead of application there is no sense in which x is a priori the 'master' (like a function in the simply-typed λ-calculus) and y is a priori the 'slave' (its argument, of lower type).

Examples

$\lambda\kappa.\kappa(a) * \kappa(b) \in \hat{\mathbb{N}}$ represents $(\lambda n.\lambda m.n * m)$ if we use substitutions instead of function application to instantiate.

Substitutions $[a{\mapsto}5]$ and $[b{\mapsto}6]$ can arrive in any order.

Examples

Let $\mathbb{B}$ be a two-element set $\{\top, \bot\}$.

$\#$ in $(\mathbb{A} \times \mathbb{X}) \Rightarrow \hat{\mathbb{B}}$ is such that $\#(a, x) = \lambda\kappa.\top$ when $a\#x$ and otherwise $\#(a, x) = \lambda\kappa.\bot$.

Using $\#$ we can case-split on whether a value has been substituted for an atom (in some programming language based on this semantics). Perhaps we can take appropriate action to fetch a value.

Examples

Much more flexible treatment of unknowns than usual:

- Usually, names of variable symbols 'x', 'y' cannot be passed as data (pointers can be passed as data, for comparison).
- Now, a kind of variable symbol 'a', 'b', can passed as data, compared for equality (reminiscent of pointers), and substituted for — even inside functions (reminiscent of symbolic computation).

Final example

An ‘exception handler’

$$\lambda x.\texttt{if}\ a\#x\ \texttt{then}\ x\ \texttt{else}\ x' \in \mathbb{X} \Rightarrow \mathbb{X}$$

(using natural notation) treats a as an exception.

If it detects a it defaults to x'.

Important: the semantics propagates a for us.

We do not need to explicitly propagate the exception, i.e. change code to ‘chaperone’ a through intervening steps of the calculation.

Conclusions

There is interest in reflecting properties of variables and freshness into logics and programming languages, to obtain good models of and reasoning principles on syntax-with-binding.

Nominal techniques reflected these properties into the semantics itself.

Conclusions

This gives us a powerful handle with which to reflect even more.

For example, the property of 'being substituted for'.

The importance of the cartesian closed property is that we get a programming language. Its power has not yet been established, but it's very strong as the examples given, and some others, demonstrate.

Names are data, they can be compared for equality in nominal style. And yet, given x and y, we can write $y[a \mapsto x]$ and get meaningful results!

What else is hiding out there?

Post-conclusions (the programming language)

Types are substitution sets. Terms are:

$$t ::= a \quad | \quad \pi \cdot x_{\mathbb{X}} \quad | \quad tt \quad | \quad \lambda x_{\mathbb{X}}.t \quad | \quad \lambda a.t \quad | \quad t[a \mapsto t]$$

up to α-equivalence by λx-bound variables, but not by λa-bound variables.

Typing

$$\frac{u:\mathbb{Y}}{\lambda x_{\mathbb{X}}.u:\mathbb{X}\Rightarrow\mathbb{Y}} \qquad \frac{u:\mathbb{X}}{\lambda a.u:\mathbb{X}\Rightarrow\mathbb{X}} \qquad \frac{}{a:\mathbb{X}} \qquad \frac{}{\pi\cdot x_{\mathbb{X}}:\mathbb{X}}$$

$$\frac{u':\mathbb{X}\quad u:\mathbb{X}}{u'[a\mapsto u]:\mathbb{X}} \qquad \frac{u':\mathbb{X}\Rightarrow\mathbb{Y}\quad u:\mathbb{X}}{u'u:\mathbb{Y}}$$

We do not type $\lambda a.x$ as $[\mathbb{A}]\mathbb{X}$, we inject abstractions directly into the function space.

Permutation and substitution actions

$$a\{x{\mapsto}t\} \equiv a \qquad (\pi \cdot x)\{x{\mapsto}t\} \equiv \pi \cdot t$$

$$(uv)\{x{\mapsto}t\} \equiv u\{x{\mapsto}t\}v\{x{\mapsto}t\}$$

$$(\lambda a.u)\{x{\mapsto}t\} \equiv \lambda a.(u\{x{\mapsto}t\})$$

$$(\lambda y.u)\{x{\mapsto}t\} \equiv \lambda y.(u\{x{\mapsto}t\}) \qquad (y \text{ not free in } t)$$

$$(u'[a{\mapsto}u])\{x{\mapsto}t\} \equiv u'\{x{\mapsto}t\}[a{\mapsto}u\{x{\mapsto}t\}]$$

$$\pi \cdot a \equiv \pi(a) \qquad \pi \cdot \pi' \cdot x \equiv (\pi \circ \pi') \cdot x \qquad \pi \cdot (tt') \equiv (\pi \cdot t)(\pi \cdot t')$$

$$\pi \cdot (\lambda x.t) \equiv \lambda x.\pi \cdot (t\{x{\mapsto}\pi^{-1} \cdot x\}) \qquad \pi \cdot \lambda a.t \equiv \lambda \pi(a).\pi \cdot t$$

$$\pi \cdot (t'[a{\mapsto}t]) \equiv (\pi \cdot t')[\pi(a){\mapsto}\pi \cdot t]$$

Some sound equalities

$$(\lambda x.t')t = t'\{x{\mapsto}t\} \qquad (\lambda a.z)x = z[a{\mapsto}x]$$

$$a[a{\mapsto}x] = x \qquad b[a{\mapsto}x] = b$$

$$a\#x \Rightarrow (h[a{\mapsto}f])x = (hx)[a{\mapsto}fx]$$

$$b\#x \Rightarrow \lambda a.x = \lambda b.(b\ a) \cdot x$$

$$(z'z)[a{\mapsto}x] = z'[a{\mapsto}x](z[a{\mapsto}x]) \qquad \text{and more...}$$