Concrete models of nominal algebra substitution. . .

. . . holey functions!

Murdoch J. Gabbay

Edinburgh University, Scotland, 13/6/2006

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,

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

What does substitution do?

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 model using interpretations of the term-formers of the syntax, to all terms.

So variable symbols do not exist in the denotation, and so neither does substitution as a function on denotational elements.

Alternatives (semantics)

Crabbé proposed a first-order logic axiomatisation of substiution.

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 a β-reduction’.

This is not relevant for this talk; we are doing algebra today so $x = y$ in the underlying denotation is just a fact.

Alternatives (syntax)

Handling abstract-syntax-with-binding in a theorem-prover or programming language is a practical issue of a certain importance. For example it was the object of my PhD (so it was very important to me).

De Bruijn indexes.

Higher-order abstract syntax.

Nominal techniques.

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.

Nominal sets

A nominal set is a set $\mathbb{X}$ with a swapping action mapping $\mathbb{A} \times \mathbb{A} \times \mathbb{X}$ to 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

$$\mathsf{N}b.\; b\#x.$$

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

Alternatives (syntax)

$$\text{И}b.\ b\#x$$

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

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

$$\text{И}b.\ \text{И}a.\ (b\ a)x = x.$$

It remains now to give axioms for substitution sets.

Axioms for substitution

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

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 $\mathbb{X}$!

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:

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

More examples

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.

A corollary of confluence is that it does not matter which representatives we choose, so this is well-defined.

More examples

But in all these examples the atoms are ‘already in there’; they are just syntax.

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

$$\tau = \lambda\kappa.\tau(\kappa) \in (\mathbb{A} \Rightarrow \mathbb{N}) \Rightarrow \mathbb{N}$$

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

$$\lambda\kappa \in (\mathbb{A} \Rightarrow \mathbb{N}).\ \mu(\kappa\{a{\mapsto}\tau(\kappa)\}).$$

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

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;

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

Function models

(A subset of) the set of functions between two substitution sets $\mathbb{X}$ and $\mathbb{Y}$, is a substitution set.

We can write $h[a{\mapsto}f]$.

Holey functions.

Function models

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

Define:

$$\begin{aligned} h[a{\mapsto}b]x &= ((b\ a)h)x && \text{if } b\#h \\ (h[a{\mapsto}f])x &= \text{И}a'.\,(((a'a)h)x)[a'{\mapsto}fx] && \text{otherwise.} \end{aligned}$$

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

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

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

'the substitution $[a\mapsto x]$'

can be represented as

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

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 $x[a{\mapsto}y]$ and $y[a{\mapsto}x]$; 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.

Much more flexible treatment of unknowns than usual; variable symbols can be detected at run-time, also at function-types!

Final example

An ‘exception handler’

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

(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 of the exception, i.e. change code to ‘chaperone’ a through intervening steps of the calculation.

Conclusions

There is already 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.

. . . HOAS, ∇, И (nominal techniques), de Bruijn, theory of contexts, FreshOCaml, . . .

A side-effect of nominal techniques was to reflect these into the semantics itself.

Conclusions

I want to reflect another property of variables into the semantics; that of ‘being substituted for’. I want to do it to mirror informal practice:

If I have x and I have y, then I can write $y[a\mapsto x]$; the parameters have names which are themselves data!

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.

What else is hiding out there?