Capture-Avoiding Substitution as a Nominal Algebra

Murdoch J. Gabbay[1] Aad Mathijssen[2]

[1]School of Mathematical and Computer Sciences
Heriot-Watt University, Edinburgh, Scotland

[2]Department of Mathematics and Computer Science
Eindhoven University of Technology, The Netherlands

3rd International Colloquium
on Theoretical Aspects of Computing (ICTAC 2006)
Tunis, Tunisia, 20-24 November 2006

Motivation

Capture-avoiding substitution in the λ-calculus

The λ-calculus:

$$t \quad ::= \quad x \mid tt \mid \lambda x.t$$

Motivation
Capture-avoiding substitution in the λ-calculus

The λ-calculus:

$$t \quad ::= \quad x \mid tt \mid \lambda x.t$$

Axioms:

$$\begin{array}{llll} (\alpha) & \lambda x.t & = \lambda y.(t[x \mapsto y]) & \text{if } y \notin fv(t) \\ (\beta) & (\lambda x.t)u & = t[x \mapsto u] & \\ (\eta) & \lambda x.(tx) & = t & \text{if } x \notin fv(t) \end{array}$$

Motivation

Capture-avoiding substitution in the λ-calculus

The λ-calculus:

$$t \quad ::= \quad x \mid tt \mid \lambda x.t$$

Axioms:

$$\begin{array}{lllll} (\alpha) & \lambda x.t & = \lambda y.(t[x \mapsto y]) & \text{if } y \notin fv(t) \\ (\beta) & (\lambda x.t)u & = t[x \mapsto u] & \\ (\eta) & \lambda x.(tx) & = t & \text{if } x \notin fv(t) \end{array}$$

Free variables function fv:

$$fv(x) = \{x\} \qquad fv(tu) = fv(t) \cup fv(u) \qquad fv(\lambda x.t) = fv(t) \backslash \{x\}$$

Motivation

Capture-avoiding substitution in the λ-calculus

The λ-calculus:

$$t \quad ::= \quad x \mid tt \mid \lambda x.t$$

Capture-avoiding substitution $\_[\_ \mapsto \_]$:

$$\begin{array}{lll}
x[x \mapsto t] & = t & \\
y[x \mapsto t] & = y & \\
(uv)[x \mapsto t] & = (u[x \mapsto t])(v[x \mapsto t]) & \\
(\lambda x.u)[x \mapsto t] & = \lambda x.u & \\
(\lambda y.u)[x \mapsto t] & = \lambda y.(u[x \mapsto t]) & \text{if } y \notin fv(t) \\
(\lambda y.u)[x \mapsto t] & = \lambda z.(u[y \mapsto z][x \mapsto t]) & \text{if } y \in fv(t),\ z \notin fv(t, u)
\end{array}$$

Motivation

Capture-avoiding substitution in the λ-calculus

The λ-calculus:

$$t \quad ::= \quad x \mid tt \mid \lambda x.t$$

Capture-avoiding substitution $\_[\_ \mapsto \_]$:

$$\begin{array}{lll}
x[x \mapsto t] & = t & \\
y[x \mapsto t] & = y & \\
(uv)[x \mapsto t] & = (u[x \mapsto t])(v[x \mapsto t]) & \\
(\lambda x.u)[x \mapsto t] & = \lambda x.u & \\
(\lambda y.u)[x \mapsto t] & = \lambda y.(u[x \mapsto t]) & \text{if } y \notin fv(t) \\
(\lambda y.u)[x \mapsto t] & = \lambda z.(u[y \mapsto z][x \mapsto t]) & \text{if } y \in fv(t),\ z \notin fv(t, u)
\end{array}$$

t, u and v are meta-variables ranging over lambda terms.

Motivation

Capture-avoiding substitution in the λ-calculus

The λ-calculus with meta-variables:

$$t \quad ::= \quad x \mid tt \mid \lambda x.t \mid X$$

Capture-avoiding substitution $\_[\_ \mapsto \_]$:

$$\begin{array}{lll}
x[x \mapsto X] & = X & \\
y[x \mapsto X] & = y & \\
(YZ)[x \mapsto X] & = (Y[x \mapsto X])(Z[x \mapsto X]) & \\
(\lambda x.Y)[x \mapsto X] & = \lambda x.Y & \\
(\lambda y.Y)[x \mapsto X] & = \lambda y.(Y[x \mapsto X]) & \text{if } y \notin fv(X) \\
(\lambda y.Y)[x \mapsto X] & = \lambda z.(Y[y \mapsto z][x \mapsto X]) & \text{if } y \in fv(X),\ z \notin fv(X, Y)
\end{array}$$

X, Y and Z represent unknown lambda terms.

Motivation

Capture-avoiding substitution in the λ-calculus

The λ-calculus with meta-variables:

$$t \quad ::= \quad x \mid tt \mid \lambda x.t \mid X$$

Capture-avoiding substitution $\_[\_ \mapsto \_]$:

$$\begin{array}{lll}
x[x \mapsto X] & = X & \\
y[x \mapsto X] & = y & \\
(Y Z)[x \mapsto X] & = (Y[x \mapsto X])(Z[x \mapsto X]) & \\
(\lambda x.Y)[x \mapsto X] & = \lambda x.Y & \\
(\lambda y.Y)[x \mapsto X] & = \lambda y.(Y[x \mapsto X]) & \text{if } y \notin fv(X) \\
(\lambda y.Y)[x \mapsto X] & = \lambda z.(Y[y \mapsto z][x \mapsto X]) & \text{if } y \in fv(X),\ z \notin fv(X, Y)
\end{array}$$

$$fv(X) = ? \qquad Y[x \mapsto X] = ?$$

Motivation

Frameworks using capture-avoiding substitution

Capture-avoiding substitution is everywhere:

- λ-calculus: $(\lambda x.t)u = t[x \mapsto u]$
- First-order logic: $\forall x.\phi = \forall x.\phi \wedge \phi[x \mapsto t]$
- Process algebra: $\sum_x p = \sum_x p + p[x \mapsto t]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $\xi x.t = \xi y.(t[x \mapsto y])$ if $y \notin fv(t)$
- $(\xi x.t)[y \mapsto u] = \xi x.(t[y \mapsto u])$ if $x \notin fv(u)$
- $v[x \mapsto t][y \mapsto u] = v[y \mapsto u][x \mapsto t[y \mapsto u]]$ if $x \notin fv(u)$

Motivation

Frameworks using capture-avoiding substitution

Capture-avoiding substitution is everywhere:

- λ-calculus: $(\lambda x.t)u = t[x \mapsto u]$
- First-order logic: $\forall x.\phi = \forall x.\phi \wedge \phi[x \mapsto t]$
- Process algebra: $\sum_x p = \sum_x p + p[x \mapsto t]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $\xi x.t = \xi y.(t[x \mapsto y])$ if $y \notin fv(t)$
- $(\xi x.t)[y \mapsto u] = \xi x.(t[y \mapsto u])$ if $x \notin fv(u)$
- $v[x \mapsto t][y \mapsto u] = v[y \mapsto u][x \mapsto t[y \mapsto u]]$ if $x \notin fv(u)$

t, u, v, ϕ, ψ, p are meta-variables ranging over terms.

Motivation

Frameworks using capture-avoiding substitution

Capture-avoiding substitution is everywhere:

- λ-calculus: $(\lambda x.X)Y = X[x \mapsto Y]$
- First-order logic: $\forall x.X = \forall x.X \wedge X[x \mapsto Y]$
- Process algebra: $\sum_x X = \sum_x X + X[x \mapsto Y]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $\xi x.X = \xi y.(X[x \mapsto y])$ if $y \notin fv(X)$
- $(\xi x.X)[y \mapsto Y] = \xi x.(X[y \mapsto Y])$ if $x \notin fv(Y)$
- $Z[x \mapsto X][y \mapsto Y] = Z[y \mapsto Y][x \mapsto X[y \mapsto Y]]$ if $x \notin fv(Y)$

X, Y and Z formally represent meta-variables.

Motivation

Frameworks using capture-avoiding substitution

Capture-avoiding substitution is everywhere:

- λ-calculus: $(\lambda x.X)Y = X[x \mapsto Y]$
- First-order logic: $\forall x.X = \forall x.X \wedge X[x \mapsto Y]$
- Process algebra: $\sum_x X = \sum_x X + X[x \mapsto Y]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $\xi x.X = \xi y.(X[x \mapsto y])$ if $y \notin fv(X)$
- $(\xi x.X)[y \mapsto Y] = \xi x.(X[y \mapsto Y])$ if $x \notin fv(Y)$
- $Z[x \mapsto X][y \mapsto Y] = Z[y \mapsto Y][x \mapsto X[y \mapsto Y]]$ if $x \notin fv(Y)$

$$fv(X) = \ ? \qquad Y[x \mapsto X] = \ ?$$

Motivation

Axiomatisation of capture-avoiding with meta-variables?

Question

Can we axiomatise capture-avoiding substitution with meta-variables with the following properties:

- *generic: parametric over the choice of term-formers*
- *close to informal practice: direct support for binding*

Motivation

Axiomatisation of capture-avoiding with meta-variables?

Question

Can we axiomatise capture-avoiding substitution with meta-variables with the following properties:

- *generic: parametric over the choice of term-formers*
- *close to informal practice: direct support for binding*

Answer

Yes, using the new framework of Nominal Algebra:

- *Nominal Algebra directly supports binding and meta-variables.*
- *Axiomatise capture-avoiding substitution as a theory that allows for arbitary term-formers.*

Nominal Algebra

Nominal Algebra:

- an equational logic on Nominal Terms (Urban, Gabbay, Pitts)
- designed to closely mirror informal reasoning about binding and meta-variables
- has built-in α-equivalence
- is sorted to keep terms well-formed

Nominal Algebra

Nominal Algebra:

- an equational logic on Nominal Terms (Urban, Gabbay, Pitts)
- designed to closely mirror informal reasoning about binding and meta-variables
- has built-in α-equivalence
- is sorted to keep terms well-formed

Properties of Nominal Algebra:

- semantics in nominal sets
- semantics based on α-equivalence classes, not functions
- sound and complete proof system
- unification up to α-equivalence is decidable

Nominal Algebra

Example properties/axioms

Meta-level properties expressed in nominal algebra:

- λ-calculus: $(\lambda[a]X)Y = X[a \mapsto Y]$
- First-order logic: $a\#Y \vdash \forall[a]X = \forall[a]X \wedge X[a \mapsto Y]$
- Process algebra: $a\#X \vdash \sum[a]X = \sum[a]X + X[a \mapsto Y]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $b\#X \vdash \xi[a]X = \xi[b](X[a \mapsto b])$
- $a\#Y \vdash (\xi[a]X)[b \mapsto Y] = \xi[a](X[b \mapsto Y])$
- $a\#Y \vdash Z[a \mapsto X][b \mapsto Y] = Z[b \mapsto Y][a \mapsto X[b \mapsto Y]]$

Nominal Algebra

Example properties/axioms

Meta-level properties expressed in nominal algebra:

- λ-calculus: $(\lambda[a]X)Y = X[a \mapsto Y]$
- First-order logic: $a\#Y \vdash \forall[a]X = \forall[a]X \wedge X[a \mapsto Y]$
- Process algebra: $a\#X \vdash \sum[a]X = \sum[a]X + X[a \mapsto Y]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $b\#X \vdash \xi[a]X = \xi[b](X[a \mapsto b])$
- $a\#Y \vdash (\xi[a]X)[b \mapsto Y] = \xi[a](X[b \mapsto Y])$
- $a\#Y \vdash Z[a \mapsto X][b \mapsto Y] = Z[b \mapsto Y][a \mapsto X[b \mapsto Y]]$

Atoms a, b represent object-variables x, y.

Nominal Algebra

Example properties/axioms

Meta-level properties expressed in nominal algebra:

- λ-calculus: $(\lambda[a]X)Y = X[a \mapsto Y]$
- First-order logic: $a\#Y \vdash \forall[a]X = \forall[a]X \wedge X[a \mapsto Y]$
- Process algebra: $a\#X \vdash \sum[a]X = \sum[a]X + X[a \mapsto Y]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $b\#X \vdash \xi[a]X = \xi[b](X[a \mapsto b])$
- $a\#Y \vdash (\xi[a]X)[b \mapsto Y] = \xi[a](X[b \mapsto Y])$
- $a\#Y \vdash Z[a \mapsto X][b \mapsto Y] = Z[b \mapsto Y][a \mapsto X[b \mapsto Y]]$

Unknowns X, Y, Z represent meta-variables t, u, v, ϕ, p.

Nominal Algebra
Example properties/axioms

Meta-level properties expressed in nominal algebra:

- λ-calculus: $(\lambda[a]X)Y = X[a \mapsto Y]$
- First-order logic: $a\#Y \vdash \forall[a]X = \forall[a]X \wedge X[a \mapsto Y]$
- Process algebra: $a\#X \vdash \sum[a]X = \sum[a]X + X[a \mapsto Y]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $b\#X \vdash \xi[a]X = \xi[b](X[a \mapsto b])$
- $a\#Y \vdash (\xi[a]X)[b \mapsto Y] = \xi[a](X[b \mapsto Y])$
- $a\#Y \vdash Z[a \mapsto X][b \mapsto Y] = Z[b \mapsto Y][a \mapsto X[b \mapsto Y]]$

Freshnesses $a\#Y$ and $b\#X$ represent $x \notin fv(u), y \notin fv(t)$

Nominal Algebra
Example properties/axioms

Meta-level properties expressed in nominal algebra:

- λ-calculus: $(\lambda[a]X)Y = X[a \mapsto Y]$
- First-order logic: $a\#Y \vdash \forall[a]X = \forall[a]X \wedge X[a \mapsto Y]$
- Process algebra: $a\#X \vdash \sum[a]X = \sum[a]X + X[a \mapsto Y]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $b\#X \vdash \xi[a]X = \xi[b](X[a \mapsto b])$
- $a\#Y \vdash (\xi[a]X)[b \mapsto Y] = \xi[a](X[b \mapsto Y])$
- $a\#Y \vdash Z[a \mapsto X][b \mapsto Y] = Z[b \mapsto Y][a \mapsto X[b \mapsto Y]]$

Abstractions $[a]X$ and $[b]Y$ represent binding fragments $x.t, y.u$

Nominal Algebra
Example properties/axioms

Meta-level properties expressed in nominal algebra:

- λ-calculus: $(\lambda[a]X)Y = X[a \mapsto Y]$
- First-order logic: $a\#Y \vdash \forall[a]X = \forall[a]X \wedge X[a \mapsto Y]$
- Process algebra: $a\#X \vdash \sum[a]X = \sum[a]X + X[a \mapsto Y]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $b\#X \vdash \xi[a]X = \xi[b](X[a \mapsto b])$
- $a\#Y \vdash (\xi[a]X)[b \mapsto Y] = \xi[a](X[b \mapsto Y])$
- $a\#Y \vdash Z[a \mapsto X][b \mapsto Y] = Z[b \mapsto Y][a \mapsto X[b \mapsto Y]]$

Term-formers for $\lambda, \_\_, \forall, \wedge, \sum, +$.

Nominal Algebra
Example properties/axioms

Meta-level properties expressed in nominal algebra:

- λ-calculus: $(\lambda[a]X)Y = X[a \mapsto Y]$
- First-order logic: $a\#Y \vdash \forall[a]X = \forall[a]X \wedge X[a \mapsto Y]$
- Process algebra: $a\#X \vdash \sum[a]X = \sum[a]X + X[a \mapsto Y]$

And for any binder $\xi \in \{\lambda, \forall, \sum\}$:

- α-equivalence: $b\#X \vdash \xi[a]X = \xi[b](X[a \mapsto b])$
- $a\#Y \vdash (\xi[a]X)[b \mapsto Y] = \xi[a](X[b \mapsto Y])$
- $a\#Y \vdash Z[a \mapsto X][b \mapsto Y] = Z[b \mapsto Y][a \mapsto X[b \mapsto Y]]$

Substitution is a term-former: we write $\mathsf{sub}([a]t, u)$ as $t[a \mapsto u]$.

An axiomatisation of capture-avoiding substitution

An axiomatisation of capture-avoiding substitution:

$$\begin{array}{rrcl}
(\mathbf{var}\mapsto) & a[a\mapsto X] & = & X \\
(\#\mapsto) & a\#Y \vdash Y[a\mapsto X] & = & Y \\
(\mathbf{f}\mapsto) & \mathsf{f}(Y_1,\ldots,Y_n)[a\mapsto X] & = & \mathsf{f}(Y_1[a\mapsto X],\ldots,Y_n[a\mapsto X]) \\
(\mathbf{abs}\mapsto) & b\#X \vdash ([b]Y)[a\mapsto X] & = & [b](Y[a\mapsto X]) \\
(\mathbf{ren}\mapsto) & b\#X \vdash X[a\mapsto b] & = & (b\ a)\cdot X
\end{array}$$

Here:

- f ranges over term-formers... including sub
- cases $b[a\mapsto X]$ and $([a]Y)[a\mapsto X]$ are covered by $(\#\mapsto)$
- $(b\ a)\cdot X$ swaps b and a when X is instantiated
- $(\mathbf{ren}\mapsto)$ links to the underlying theory of α-equivalence
- we call this axiomatisation SUB

Instantiation of axioms

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

Instantiation	Resulting property
	$a\#Y \vdash Y[a \mapsto X] = Y$
$Y := b$	$b[a \mapsto X] = b$, since $\vdash a\#b$
$Y := a$	none, since $\not\vdash a\#a$
$Y := [a]Z$	$([a]Z)[a \mapsto X] = [a]Z$, since $\vdash a\#[a]Z$
$Y := [b]Z$	$a\#Z \vdash ([b]Z)[a \mapsto X] = [b]Z$
$Y := \mathsf{f}(Y_1, \ldots, Y_n)$	$a\#Y_1, \ldots, a\#Y_n \vdash$ $\mathsf{f}(Y_1, \ldots, Y_n)[a \mapsto X] = \mathsf{f}(Y_1, \ldots, Y_n)$
$Y := Z, X := Y, a := b$	$b\#Z \vdash Z[b \mapsto Y] = Z$

Equational proofs

Lemma

$c\#X, c\#Y \vdash ([b]Y)[a \mapsto X] = [c](Y[b \mapsto c][a \mapsto X])$ *is derivable.*

Equational proofs

Lemma

$c\#X, c\#Y \vdash ([b]Y)[a \mapsto X] = [c](Y[b \mapsto c][a \mapsto X])$ *is derivable.*

Proof.

$$
\begin{array}{ll}
 & ([b]Y)[a \mapsto X] \\
= & \quad \{\ [b]Y = [c](c\ b) \cdot Y, \text{ since } c\#X, c\#Y \vdash c\#[b]Y, b\#[b]Y\ \} \\
 & ([c](c\ b) \cdot Y)[a \mapsto X] \\
= & \quad \{\ \text{axiom } (\mathbf{abs}{\mapsto}), \text{ since } c\#X, c\#Y \vdash c\#X\ \} \\
 & [c]((c\ b) \cdot Y)[a \mapsto X] \\
= & \quad \{\ \text{axiom } (\mathbf{ren}{\mapsto}), \text{ since } c\#X, c\#Y \vdash c\#X\ \} \\
 & [c]Y[b \mapsto c][a \mapsto X]
\end{array}
$$

□

Equational proofs

Lemma

$c\#X, c\#Y \vdash ([b]Y)[a \mapsto X] = [c](Y[b \mapsto c][a \mapsto X])$ *is derivable.*

Corollary

The axiomatisation of substitution can mimic the usual definition of capture-avoiding substitution (without unknowns):

$$
\begin{array}{lll}
x[x \mapsto t] & = t & \\
y[x \mapsto t] & = y & \\
\mathsf{f}(u_1, \ldots, u_n)[x \mapsto t] & = \mathsf{f}(u_1[x \mapsto t], \ldots, u_n[x \mapsto t]) & \\
(\xi x.u)[x \mapsto t] & = \xi x.u & \\
(\xi y.u)[x \mapsto t] & = \xi y.u[x \mapsto t] & \text{if } y \notin fv(u) \\
(\xi y.u)[x \mapsto t] & = \xi z.(u[y \mapsto z][x \mapsto t] & \text{if } y \in fv(t),\ z \notin fv(t, u)
\end{array}
$$

Equational proofs

Lemma

$X[a \mapsto a] = X$ is derivable.

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

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

$$\frac{[b](b\ a)\cdot X = [a]X}{[a]X = [b](b\ a)\cdot X}\ (\mathbf{symm})$$

$$\frac{[a]X = [b](b\ a)\cdot X}{X[a \mapsto a] = ((b\ a)\cdot X)[b \mapsto a]}\ (\mathbf{congf})$$

$$\frac{[b\#X]^1}{a\#(b\ a)\cdot X}\ (\#\mathbf{X})$$

$$\frac{a\#(b\ a)\cdot X}{((b\ a)\cdot X)[b \mapsto a] = X}\ (\mathbf{ax}_{\mathbf{ren}\mapsto})$$

$$\frac{X[a \mapsto a] = ((b\ a)\cdot X)[b \mapsto a] \qquad ((b\ a)\cdot X)[b \mapsto a] = X}{X[a \mapsto a] = X}\ (\mathbf{tran})$$

$$\frac{X[a \mapsto a] = X}{X[a \mapsto a] = X}\ (\mathbf{fr})^1$$

α-conversion

α-conversion in nominal algebra is expressed by the proof rule:

$$\frac{a\#t \quad b\#t}{(b\ a)\cdot t = t}\ \textbf{(perm)}$$

α-conversion

α-conversion in nominal algebra is expressed by the proof rule:

$$\frac{a\#t \quad b\#t}{(b\ a)\cdot t = t}\ (\mathbf{perm})$$

Why not replace this rule by the following axiom instead?

$$a\#X, b\#X \vdash X[a \mapsto b] = X$$

α-conversion

α-conversion in nominal algebra is expressed by the proof rule:

$$\frac{a\#t \quad b\#t}{(b\ a)\cdot t = t}\ (\mathbf{perm})$$

Why not replace this rule by the following axiom instead?

$$a\#X, b\#X \vdash X[a \mapsto b] = X$$

This destroys the proof theory:

- When proving properties by induction on the size of terms, you often want to freshen up a term using α-conversion.
- Freshening using the axiom increases term size, destroying the inductive hypothesis

α-conversion

α-conversion in nominal algebra is expressed by the proof rule:

$$\frac{a\#t \quad b\#t}{(b\ a)\cdot t = t}\ \textbf{(perm)}$$

Why not replace this rule by the following axiom instead?

$$a\#X, b\#X \vdash X[a \mapsto b] = X$$

Not all theories with binding use substitution of terms for atoms.
For example, the π-calculus has substitution of atoms for atoms.

Substitution as a rewrite system

Directing the equalities of our axiomatisation SUB we obtain a nominal rewrite system SUBr.

Lemma (Equivalence of equality and rewriting)

SUB *is equivalent to the transitive reflexive symmetric closure of* SUBr *(assuming sufficient freshnesses).*

So we can use nice properties from the world of rewriting such as confluence and termination.

Substitution as a rewrite system
Simultaneous substitutions

Problem: SUBr is not terminating because SUB has a simultaneous character:

$$X[a \mapsto a'][b \mapsto b'][c \mapsto c'] \to^* X[c \mapsto c'][b \mapsto b'][a \mapsto a']$$
$$X[c \mapsto c'][b \mapsto b'][a \mapsto a'] \to^* X[a \mapsto a'][b \mapsto b'][c \mapsto c']$$

Substitution as a rewrite system

Simultaneous substitutions

Solution: introduce an equational theory SUBe of simultaneous substitutions:

$$
\begin{array}{rll}
a\#Y, b\#X & \vdash Z[a \mapsto X][b \mapsto Y] & = \quad Z[b \mapsto Y][a \mapsto X] \\
a\#Y & \vdash Y[a \mapsto X] & = \quad Y
\end{array}
$$

Substitution as a rewrite system

Simultaneous substitutions

Solution: introduce an equational theory SUBe of simultaneous substitutions:

$$
\begin{array}{rlcl}
a\#Y, b\#X & \vdash Z[a \mapsto X][b \mapsto Y] & = & Z[b \mapsto Y][a \mapsto X] \\
a\#Y & \vdash Y[a \mapsto X] & = & Y
\end{array}
$$

Lemma

SUBr *is terminating and confluent up to* SUBe.

Substitution as a rewrite system
Simultaneous substitutions

Solution: introduce an equational theory SUBe of simultaneous substitutions:

$$\begin{array}{rlcl} a\#Y, b\#X & \vdash Z[a \mapsto X][b \mapsto Y] & = & Z[b \mapsto Y][a \mapsto X] \\ a\#Y & \vdash Y[a \mapsto X] & = & Y \end{array}$$

Lemma
SUBr *is terminating and confluent up to* SUBe.

Lemma
Each SUBe *equivalence class has a representative to which each term in that class rewrites.*

Substitution as a rewrite system
Confluence

Theorem (Confluence)

SUBr *is confluent.*

Proof (sketch).

Suppose $t \longrightarrow^* t_1$ and $t \longrightarrow^* t_2$.
By confluence up to SUBe, t_1 and t_2 rewrite to terms u_1 and u_2, such that $u_1 = u_2$ in SUBe. Then u_1 and u_2 have the same representative u to which they rewrite. □

Corollaries of confluence

Some corollaries of confluence:

- SUB is a conservative extension over the empty theory:

  $$\Delta \vdash_{\mathsf{SUB}} t = u \quad \text{iff} \quad \Delta \vdash_{\emptyset} t = u$$

  for all t and u not mentioning substitution.
- SUB is equivalent to the usual definition of capture-avoiding substitution, on terms not mentioning unknowns X, Y, Z.

Decidability

Lemma
$\Delta \vdash_{\mathsf{SUBe}} t = u$ *is decidable.*

Theorem
$\Delta \vdash_{\mathsf{SUB}} t = u$ *is decidable.*

Proof (sketch).

1. Rewrite t and u to normal forms up to SUBe t' and u'.
2. Check whether $\Delta \vdash_{\mathsf{SUBe}} t' = u'$ is decidable.

□

ω-completeness
Definition

Some terminology:

- Call a term t closed if it does not mention unknowns.
- Write σ for an instantiation of unknowns to closed terms.

SUB is sound and complete with respect to the closed term model.
This is also called ω-completeness.

Theorem (ω-completeness)

$\Delta \vdash_{\mathsf{SUB}} t = u \quad \textit{iff} \quad \vdash_{\mathsf{SUB}} t\sigma = u\sigma \textit{ for all } \sigma \textit{ such that} \vdash \Delta\sigma$

ω-completeness
Proof

Theorem (ω-completeness)

$\Delta \vdash_{\textsf{SUB}} t = u$ *iff* $\vdash_{\textsf{SUB}} t\sigma = u\sigma$ *for all* σ *such that* $\vdash \Delta\sigma$

ω-completeness
Proof

Theorem (ω-completeness)

$\Delta \vdash_{\text{SUB}} t = u$ *iff* $\vdash_{\text{SUB}} t\sigma = u\sigma$ *for all* σ *such that* $\vdash \Delta\sigma$

Proof (sketch).

Left-to-right: property of any theory in nominal algebra. □

ω-completeness
Proof

Theorem (ω-completeness)

$\Delta \vdash_{\mathsf{SUB}} t = u$ *iff* $\vdash_{\mathsf{SUB}} t\sigma = u\sigma$ *for all* σ *such that* $\vdash \Delta\sigma$

Proof (sketch).

Right-to-left: by contraposition.

1. Suppose $\nvdash_{\mathsf{SUB}} t = u$.
2. By confluence, then also $\nvdash_{\mathsf{SUBe}} t = u$.
3. Then we can suffice by showing that there exists a σ such that $\vdash \Delta\sigma$ and $\nvdash_{\mathsf{SUB}} t\sigma = u\sigma$.
4. Work by induction on the size of t and u.

□

Conclusions

Nominal algebra allows us to:

- axiomatise capture-avoiding substitution with meta-variables
- parametric over the choice of term-formers
- supporting binding and freshness directly

The axiomatisation has strong properties:

- equivalent to ‘ordinary’ capture-avoiding substitution on terms without unknowns
- conservative extension of the empty theory
- decidability of equality
- ω-completeness

Related work
axiomatisations of substitution

Related axiomatisations of substitutions:

- Logos (Crabbé):
  - also uses atoms and freshness conditions
  - does not treat binding
  - works in first-order logic
- Polynomial substitution algebras (Feldman):
  - closer to Cylindric Algebras and Lambda Abstraction Algebras
  - $\forall a, \forall b, \ldots$ are encoded as an infinite family of unary operators
  - less expressive on open terms
- Explicit substitutions:
  - implementation vs axiomatisation
  - variables are often encoded as de Bruijn indices

Related work

Applications of our work

Applications of our axiomatisation of substitution:

- the basic notion of equality in one-and-a-halfth-order logic: a theory of first-order logic with meta-variables (Gabbay, Mathijssen)
- abstract (non-term-based) models: substitution sets (Gabbay, Marin, Bulò)
- basic notion of capture-avoiding substitution in nominal equational logic (Pitts, Clouston)

Future work

Future work on capture-avoiding substitution:

- unification up to SUB
- take SUB over itself:
  - express $X[a \mapsto Y][t/X]$ as $X[a \mapsto Y][X \mapsto \mathcal{T}]$ in a stronger axiom system where $\mathcal{T}$ is a 'stronger' meta-variable
  - related to the NEW calculus of contexts and hierarchical nominal rewriting (Gabbay)
- develop logics and λ-calculi with a new way of treating meta-variables, binding and substitution

Further reading

- Murdoch J. Gabbay, Aad Mathijssen:
  Nominal Algebra.
  Submitted STACS'07.
- Murdoch J. Gabbay, Andrea Marin, Samuel Rota Bulò:
  A nominal semantics for simple types.
  Submitted STACS'07.
- Murdoch J. Gabbay, Aad Mathijssen:
  One-and-a-halfth-order Logic.
  PPDP'06.

Papers and slides of talks can be found on my web page:
http://www.win.tue.nl/~amathijs