Equivariant and Nominal SOS

Andrew Pitts
University of Cambridge
Computer Laboratory

This talk is about the first “S” in “SOS”

This talk is about the first “S” in “SOS”

“[Previous approaches to operational semantics] do not in general have any great claim to being syntax-directed in the sense of defining the semantics of compound phrases in terms of the semantics of their components.”

GD Plotkin, *A Structural Approach to Operational Semantics*, page 21 (Aarhus, 1981).

This talk is about the first "S" in "SOS"

> "[Previous approaches to operational semantics] do not in general have any great claim to being syntax-directed in the sense of defining the semantics of compound phrases in terms of the semantics of their components."
>
> GD Plotkin, A *Structural Approach to Operational Semantics*, page 21 (Aarhus, 1981).

Key tool for SOS: structural recursion/induction for abstract syntax trees (ASTs).

Except that in practice ASTs are not abstract enough. . .

Abstract syntax / α

Many (most?) languages involve binders.

“We identify expressions up to α-equivalence”. . .

Abstract syntax / α

Many (most?) languages involve binders.

"We identify expressions up to α-equivalence"...
(and then forget about it!)

I find common informal practices using notationless α-equivalence classes to be unsatisfactory when it comes to structurally recursive definitions and proofs by structural induction.

E.g...

Capture-avoiding substitution

for λ-terms $t ::= a \mid t\,t \mid \lambda a\,t$

$$(a := t)a' \triangleq t \text{ if } a' = a, \text{ else } \triangleq a'$$
$$(a := t)(t_1\,t_2) \triangleq ((a := t)t_1)((a := t)t_2)$$
$$(a := t)\lambda a'\,t' \triangleq \lambda a'(a := t)t' \text{ if } a' \notin fv(t, a)$$

Is this a definition of $(a := t)t'$ by recursion on the structure of the AST t'?

Capture-avoiding substitution

for λ-terms $t ::= a \mid t\,t \mid \lambda a\,t$

$$\boxed{\begin{array}{c}(a := t)a' \triangleq t \text{ if } a' = a, \text{ else } \triangleq a' \\ (a := t)(t_1\,t_2) \triangleq ((a := t)t_1)((a := t)t_2) \\ (a := t)\lambda a'\,t' \triangleq \lambda a'(a := t)t' \text{ if } a' \notin fv(t, a)\end{array}}$$

Is this a definition of $(a := t)t'$ by recursion on the structure of the AST t'?

No, because of the condition on the last clause; but it would be stupid to have to say what to do in case $a' \in fv(t, a)$.

Capture-avoiding substitution

for λ-terms $t ::= a \mid t\,t \mid \lambda a\,t$

$$(a := t)a' \triangleq t \text{ if } a' = a, \text{ else } \triangleq a'$$
$$(a := t)(t_1\,t_2) \triangleq ((a := t)t_1)((a := t)t_2)$$
$$(a := t)\lambda a'\,t' \triangleq \lambda a'(a := t)t' \text{ if } a' \notin fv(t, a)$$

Is this a definition of $(a := t)t'$ by recursion on the structure of the AST t'?

No, because of the condition on the last clause; but it would be stupid to have to say what to do in case $a' \in fv(t, a)$.

Why is it "obvious" that the above well-defines a total function on α-equivalence classes of ASTs?

Why bother?

"I'm an expert and could patch things up to full formality if pressed (but have more important and interesting things to do)."

Why bother?

"I'm an expert and could patch things up to full formality if pressed (but have more important and interesting things to do)."

Might be true for familiar old λ-calculus, but look at the mess in early versions of LTSs for π-calculus.

What about large-scale SOS definitions? (E.g. *Definition of SML* mixes ASTs with ASTs/α.)

What about non-experts? What about machines?

Why bother?

> "I'm an expert and could patch things up to full formality if pressed (but have more important and interesting things to do)."

Might be true for familiar old λ-calculus, but look at the mess in early versions of LTSs for π-calculus.

What about large-scale SOS definitions? (E.g. *Definition of SML* mixes ASTs with ASTs/α.)

What about non-experts? What about machines?

We really need a light-weight theory of structural recursion/induction for syntax/α that doesn't stray too far from common, <u>nominal</u> practices.

There is one!

Existing approaches to syntax/α

- De Bruijnery. An implementation technique that's inconvenient/error-prone for reasoning by humans.
- HOAS. Pushes the problem with structural recursion/induction up a meta-level without solving it.
- FM-sets (Gabbay-AMP). Uses Fraenkel-Mostowski permutation model of set theory with atoms.

Existing approaches to syntax/α

- De Bruijnery. An implementation technique that's inconvenient/error-prone for reasoning by humans.
- HOAS. Pushes the problem with structural recursion/induction up a meta-level without solving it.
- FM-sets (Gabbay-AMP). Uses Fraenkel-Mostowski permutation model of set theory with atoms.

This talk: a simplified explanation of the FM-sets approach, tailored to SOS.

(Similar in spirit, but not in detail, to Gordon & Melham's *Five Axioms of Alpha-Conversion*.)

Atoms, permutations and actions

- $\mathbb{A} \triangleq$ fixed, countably infinite set, whose elements will be called atoms.

Atoms, permutations and actions

- $\mathbb{A} \triangleq$ fixed, countably infinite set, whose elements will be called atoms.
- $\mathbb{G} \triangleq$ group of all finite permutations of $\mathbb{A}$.

Atoms, permutations and actions

- $\mathbb{A} \triangleq$ fixed, countably infinite set, whose elements will be called atoms.
- $\mathbb{G} \triangleq$ group of all finite permutations of $\mathbb{A}$.
- An action of $\mathbb{G}$ on a set X is a function

$$\mathbb{G} \times X \to X \quad \text{written} \quad (\pi, x) \mapsto \pi \cdot x$$

satisfying $\boxed{\iota \cdot x = x}$ and $\boxed{\pi \cdot (\pi' \cdot x) = (\pi\pi') \cdot x}$

Atoms, permutations and actions

- $\mathbb{A} \triangleq$ fixed, countably infinite set, whose elements will be called **atoms**.
- $\mathbb{G} \triangleq$ group of all **finite permutations** of $\mathbb{A}$.
- An **action** of $\mathbb{G}$ on a set X is a function

$$\mathbb{G} \times X \longrightarrow X \quad \text{written} \quad (\pi, x) \mapsto \pi \cdot x$$

satisfying $\boxed{\iota \cdot x = x}$ and $\boxed{\pi \cdot (\pi' \cdot x) = (\pi\pi') \cdot x}$

- **$\mathbb{G}$-set** $\triangleq$ set X + action of $\mathbb{G}$ on X.

Finite support and freshness

<u>Definition</u>. A finite set A of atoms **supports** an element $x \in X$ of a $\mathbb{G}$-set X if

$$(\forall a, a' \in \mathbb{A} - A) \quad (a\, a') \cdot x = x$$

Finite support and freshness

Definition. A finite set $\boldsymbol{A}$ of atoms **supports** an element $\boldsymbol{x} \in \boldsymbol{X}$ of a $\mathbb{G}$-set $\boldsymbol{X}$ if

$$(\forall \boldsymbol{a}, \boldsymbol{a}' \in \mathbb{A} - \boldsymbol{A}) \quad (\boldsymbol{a}\,\boldsymbol{a}') \cdot \boldsymbol{x} = \boldsymbol{x}$$

the permutation that swaps a and a'

Finite support and freshness

<u>Definition</u>. A finite set A of atoms supports an element $x \in X$ of a $\mathbb{G}$-set X if

$$(\forall a, a' \in \mathbb{A} - A) \quad (a\, a') \cdot x = x$$

A nominal set is a $\mathbb{G}$-set all of whose elements have a finite support.

Finite support and freshness

Definition. A finite set A of atoms **supports** an element $x \in X$ of a $\mathbb{G}$-set X if

$$(\forall a, a' \in \mathbb{A} - A) \quad (a\,a') \cdot x = x$$

A **nominal set** is a $\mathbb{G}$-set all of whose elements have a finite support.

Lemma. If $x \in X$ has a finite support, then it has a smallest one, written $\boxed{supp(x)}$

Notation. If $a \notin supp(x)$, we write $\boxed{a \# x}$ and say "a is fresh for x."

Languages are nominal sets

For example, the set of ASTs of λ-terms $\mathbf{\Lambda} \triangleq \{t ::= a \mid t\,t \mid \boldsymbol{\lambda} a\,t\}$ with $\mathbb{G}$-action:

$$
\begin{aligned}
\pi \cdot a &\triangleq \pi(a) \\
\pi \cdot (\boldsymbol{\lambda} a\,t) &\triangleq \boldsymbol{\lambda}\pi(a)\,(\pi \cdot t) \\
\pi \cdot (t\,t') &\triangleq (\pi \cdot t)(\pi \cdot t')
\end{aligned}
$$

Languages are nominal sets

For example, the set of ASTs of λ-terms $\boldsymbol{\Lambda} \triangleq \{t ::= a \mid t\,t \mid \lambda a\,t\}$ with $\mathbb{G}$-action:

$$\begin{aligned}
\pi \cdot a &\triangleq \pi(a)\\
\pi \cdot (\lambda a\,t) &\triangleq \lambda \pi(a)\,(\pi \cdot t)\\
\pi \cdot (t\,t') &\triangleq (\pi \cdot t)(\pi \cdot t')
\end{aligned}$$

N.B. binding and non-binding constructs are treated just the same

Languages are nominal sets

For example, the set of ASTs of λ-terms $\boldsymbol{\Lambda} \triangleq \{t ::= a \mid t\,t \mid \boldsymbol{\lambda} a\,t\}$ with $\mathbb{G}$-action:

$$
\begin{aligned}
\pi \cdot a &\triangleq \pi(a) \\
\pi \cdot (\boldsymbol{\lambda} a\,t) &\triangleq \boldsymbol{\lambda}\pi(a)\,(\pi \cdot t) \\
\pi \cdot (t\,t') &\triangleq (\pi \cdot t)(\pi \cdot t')
\end{aligned}
$$

For this action, it is not hard to see that t is supported by any set of atoms containing all those occurring in t and hence

$a \mathrel{\#} t$ iff a does not occur in the tree t.

Nominal powersets

If X is a $\mathbb{G}$-set, we get a $\mathbb{G}$-action on its subsets by defining for each $\pi \in \mathbb{G}$ and $S \subseteq X$:

$$\boxed{\pi \cdot S \triangleq \{\pi \cdot x \mid x \in S\}}$$

Even if X is nominal, not every subset of it is necessarily finitely supported; e.g. $S \subseteq \mathbb{A}$ is finitely supported iff either S or $\mathbb{A} - S$ is finite.

Nominal powersets

If X is a $\mathbb{G}$-set, we get a $\mathbb{G}$-action on its subsets by defining for each $\pi \in \mathbb{G}$ and $S \subseteq X$:

$$\boxed{\pi \cdot S \triangleq \{\pi \cdot x \mid x \in S\}}$$

The set $\boxed{P_{\mathrm{fs}}(X)}$ of finitely supported subsets of a nominal set X is, by construction, a nominal set.

Nominal powersets

If X is a $\mathbb{G}$-set, we get a $\mathbb{G}$-action on its subsets by defining for each $\pi \in \mathbb{G}$ and $S \subseteq X$:

$$\boxed{\pi \cdot S \triangleq \{\pi \cdot x \mid x \in S\}}$$

The set $\boxed{P_{\mathrm{fs}}(X)}$ of finitely supported subsets of a nominal set X is, by construction, a nominal set.

N.B. $supp(S) = \emptyset$ iff S is an equivariant subset, i.e.

$$(\forall \pi \in \mathbb{G})(\forall x \in X)\ x \in S \ \Rightarrow\ \pi \cdot x \in S$$

Nominal function sets

We get a $\mathbb{G}$-action on the functions from a $\mathbb{G}$-set X to a $\mathbb{G}$-set Y by defining for each $\pi \in \mathbb{G}$, $f : X \to Y$ and $x \in X$:

$$\boxed{(\pi \cdot f)(x) \triangleq \pi \cdot (f(\pi^{-1} \cdot x))}$$

As for subsets, even if X and Y are nominal, *not* every function from X to Y is necessarily finitely supported.

Nominal function sets

We get a $\mathbb{G}$-action on the functions from a $\mathbb{G}$-set X to a $\mathbb{G}$-set Y by defining for each $\pi \in \mathbb{G}$, $f : X \to Y$ and $x \in X$:

$$\boxed{(\pi \cdot f)(x) \triangleq \pi \cdot (f(\pi^{-1} \cdot x))}$$

The set $\boxed{X \to_{\mathrm{fs}} Y}$ of finitely supported functions from a nominal set X to a nominal set Y is, by construction, a nominal set.

Nominal function sets

We get a $\mathbb{G}$-action on the functions from a $\mathbb{G}$-set X to a $\mathbb{G}$-set Y by defining for each $\pi \in \mathbb{G}$, $f : X \to Y$ and $x \in X$:

$$\boxed{(\pi \cdot f)(x) \triangleq \pi \cdot (f(\pi^{-1} \cdot x))}$$

The set $\boxed{X \to_{\mathrm{fs}} Y}$ of finitely supported functions from a nominal set X to a nominal set Y is, by construction, a nominal set.

N.B. $supp(f) = \emptyset$ iff f is an equivariant function, i.e.

$$(\forall \pi \in \mathbb{G})(\forall x \in X)\ \pi \cdot (f\ x) = f(\pi \cdot x)$$

Swapping and freshness are equivariant

For any nominal set X, the ternary function $\mathbb{A} \times \mathbb{A} \times X \to X$ given by $(a, a', x) \mapsto (a\, a') \cdot x$ is equivariant:

$$\boxed{\pi \cdot ((a\, a') \cdot x) = (\pi(a)\, \pi(a')) \cdot (\pi \cdot x)}$$

(because $\pi(a\, a')\pi^{-1} = (\pi(a)\, \pi(a'))$ in $\mathbb{G}$).

Also, the freshness relation is equivariant:

$$\boxed{a \mathrel{\#} x \;\Rightarrow\; \pi(a) \mathrel{\#} \pi \cdot x}$$

The second fact follows from the first because of the general logical properties of finitely supported sets and functions. . .

First-order logic

First-order logic for nominal sets is just like for ordinary sets. For example:

- Negation: if $[\![\phi(x)]\!] = S \in P_{\mathrm{fs}}(X)$, then

$$[\![\neg\phi(x)]\!] = X - S$$

  (RHS is in $P_{\mathrm{fs}}(X)$ because it is supported by any finite set of atoms supporting S.)

First-order logic

First-order logic for nominal sets is just like for ordinary sets. For example:

- Negation: if $[\![\phi(x)]\!] = S \in P_{\mathrm{fs}}(X)$, then
$$[\![\neg\phi(x)]\!] = X - S$$

- For all: if $[\![\phi(x,y)]\!] = S \in P_{\mathrm{fs}}(X \times Y)$, then
$$[\![\forall x.\, \phi(x,y)]\!] = \{y \in Y \mid \forall x \in X.\, (x,y) \in S\}$$
(RHS is in $P_{\mathrm{fs}}(Y)$, because it is supported by any finite set of atoms supporting S.)

Higher-order logic

Higher-order logic for nominal sets is as for ordinary sets, except that we have to restrict to *finitely supported* sets and functions when forming powersets and exponentials.

Higher-order logic

Higher-order logic for nominal sets is as for ordinary sets, except that we have to restrict to *finitely supported* sets and functions when forming powersets and exponentials.

For example
Tarski Fixpoint Theorem. For any monotone and finitely supported function Φ from $P_{\mathrm{fs}}(X)$ to itself, the usual least (pre)fixed point

$$\mu(\Phi) \triangleq \bigcap \{S \in P_{\mathrm{fs}}(X) \mid \Phi(S) \subseteq S\}$$

is again finitely supported, hence in $P_{\mathrm{fs}}(X)$.

Rule-based inductive definitions

Theorem. If X is a nominal set and $R \subseteq X$ is inductively defined by a set of rules, then $R \in P_{\text{fs}}(X)$ if the rule-set is finitely supported, i.e. if there is a finite set of atoms A such that for any $a, a' \in \mathbb{A} - A$

if $\dfrac{h_1 \in R \quad \cdots \quad h_n \in R}{c \in R}$ is in the rule-set,

then so is $\dfrac{(a\,a') \cdot h_1 \in R \quad \cdots \quad (a\,a') \cdot h_n \in R}{(a\,a') \cdot c \in R}$.

(In which case $supp(R) \subseteq A$. In particular R is equivariant if the rule-set is.)

α-Equivalence, structurally

$$\frac{}{a =_\alpha a} \qquad \frac{t_1 =_\alpha t_1' \qquad t_2 =_\alpha t_2'}{t_1\, t_2 =_\alpha t_1'\, t_2'}$$

$$\frac{(a\, a'') \cdot t =_\alpha (a'\, a'') \cdot t'}{\lambda a\, t =_\alpha \lambda a'\, t'}\ a'' \# (a, t, a', t')$$

α-Equivalence, structurally

$$\frac{}{a =_\alpha a} \qquad \frac{t_1 =_\alpha t_1' \qquad t_2 =_\alpha t_2'}{t_1\, t_2 =_\alpha t_1'\, t_2'}$$

$$\frac{(a\, a'') \cdot t =_\alpha (a'\, a'') \cdot t'}{\lambda a\, t =_\alpha \lambda a'\, t'}\; a'' \# (a, t, a', t')$$

This set of rules is equivariant, because swapping and freshness are. So by the theorem $=_\alpha$ is equivariant:

$$t =_\alpha t' \;\Rightarrow\; \pi \cdot t =_\alpha \pi \cdot t'$$

Can use this for an easy proof that $=_\alpha$ is an equivalence relation...

$t =_\alpha t' \;\&\; t' =_\alpha t'' \Rightarrow t =_\alpha t''$

Use rule induction—show that

$$H \triangleq \{(t, t') \mid (\forall t'')\; t' =_\alpha t'' \Rightarrow t =_\alpha t''\}$$

is closed under the rules defining $=_\alpha$. Only closure under the rule for λ-abstractions is non-trivial.

Have to prove (1) implies $(\lambda a\, t, \lambda a'\, t') \in H$, where

(1) $((a\, a'') \cdot t, (a'\, a'') \cdot t') \in H \;\;\&\;\; a'' \# (a, t, a', t')$

$$t =_\alpha t' \ \& \ t' =_\alpha t'' \Rightarrow t =_\alpha t''$$

Use rule induction—show that

$$H \triangleq \{(t, t') \mid (\forall t'')\ t' =_\alpha t'' \Rightarrow t =_\alpha t''\}$$

is closed under the rules defining $=_\alpha$. Only closure under the rule for λ-abstractions is non-trivial.

Have to prove (1) & (2) implies $\boldsymbol{\lambda} a\, t =_\alpha t''$, where

(1) $((a\, a'') \cdot t, (a'\, a'') \cdot t') \in H \ \ \& \ \ a'' \# (a, t, a', t')$

(2) $\boldsymbol{\lambda} a'\, t' =_\alpha t''$

$$t =_\alpha t' \mathbin{\&} t' =_\alpha t'' \Rightarrow t =_\alpha t''$$

Use rule induction—show that

$$H \triangleq \{(t, t') \mid (\forall t'')\; t' =_\alpha t'' \Rightarrow t =_\alpha t''\}$$

is closed under the rules defining $=_\alpha$. Only closure under the rule for λ-abstractions is non-trivial.

Have to prove (1) & (2) implies $\lambda a\, t =_\alpha t''$, where

(1) $((a\, a'') \cdot t, (a'\, a'') \cdot t') \in H \;\;\&\;\; a'' \# (a, t, a', t')$

(2) $\lambda a'\, t' =_\alpha t''$

The rules for $=_\alpha$ are syntax-directed, so $t'' = \lambda a_1\, t_1$ where. . .

$$t =_\alpha t' \mathbin{\&} t' =_\alpha t'' \Rightarrow t =_\alpha t''$$

Use rule induction—show that

$$H \triangleq \{(t, t') \mid (\forall t'')\ t' =_\alpha t'' \Rightarrow t =_\alpha t''\}$$

is closed under the rules defining $=_\alpha$. Only closure under the rule for λ-abstractions is non-trivial.

Have to prove (1) & (2) implies $\lambda a\, t =_\alpha \lambda a_1\, t_1$, where

(1) $((a\, a'') \cdot t, (a'\, a'') \cdot t') \in H \ \ \& \ \ a'' \# (a, t, a', t')$

(2) $(a'\, a_2) \cdot t' =_\alpha (a_1\, a_2) \cdot t_1 \ \ \& \ \ a_2 \# (a', t', a_1, t_1)$

$t =_\alpha t' \ \&\ t' =_\alpha t'' \Rightarrow t =_\alpha t''$

Use rule induction—show that

$$H \triangleq \{(t, t') \mid (\forall t'')\ t' =_\alpha t'' \Rightarrow t =_\alpha t''\}$$

is closed under the rules defining $=_\alpha$. Only closure under the rule for λ-abstractions is non-trivial.

Have to prove (1) & (2) implies $\lambda a\, t =_\alpha \lambda a_1\, t_1$, where

(1) $((a\, a'') \cdot t, (a'\, a'') \cdot t') \in H \ \ \&\ \ a'' \# (a, t, a', t')$

(2) $(a'\, a_2) \cdot t' =_\alpha (a_1\, a_2) \cdot t_1 \ \ \&\ \ a_2 \# (a', t', a_1, t_1)$

$=_\alpha$ and (hence) H are equivariant, so choose a fresh a_3 and consider $(a''\, a_3)\cdot$(1) and $(a_2\, a_3)\cdot$(2).

$$t =_\alpha t' \;\&\; t' =_\alpha t'' \Rightarrow t =_\alpha t''$$

Use rule induction—show that

$$H \triangleq \{(t, t') \mid (\forall t'')\; t' =_\alpha t'' \Rightarrow t =_\alpha t''\}$$

is closed under the rules defining $=_\alpha$. Only closure under the rule for λ-abstractions is non-trivial.

Have to prove (1) & (2) implies $\lambda a\, t =_\alpha \lambda a_1\, t_1$, where

(1) $((a\, a_3) \cdot t, (a'\, a_3) \cdot t') \in H \;\;\&\;\; a_3 \# (a, t, a', t')$

(2) $(a'\, a_3) \cdot t' =_\alpha (a_1\, a_3) \cdot t_1 \;\;\&\;\; a_3 \# (a', t', a_1, t_1)$

$$t =_\alpha t' \;\&\; t' =_\alpha t'' \Rightarrow t =_\alpha t''$$

Use rule induction—show that

$$H \triangleq \{(t, t') \mid (\forall t'')\, t' =_\alpha t'' \Rightarrow t =_\alpha t''\}$$

is closed under the rules defining $=_\alpha$. Only closure under the rule for λ-abstractions is non-trivial.

Have to prove (1) & (2) implies $\lambda a\, t =_\alpha \lambda a_1\, t_1$, where

(1) $((a\, a_3) \cdot t, (a'\, a_3) \cdot t') \in H \;\;\&\;\; a_3 \,\#\, (a, t, a', t')$

(2) $(a'\, a_3) \cdot t' =_\alpha (a_1\, a_3) \cdot t_1 \;\;\&\;\; a_3 \,\#\, (a', t', a_1, t_1)$

(1) & (2) & definition of H give. . .

$$t =_\alpha t' \;\&\; t' =_\alpha t'' \Rightarrow t =_\alpha t''$$

Use rule induction—show that

$$H \triangleq \{(t, t') \mid (\forall t'')\; t' =_\alpha t'' \Rightarrow t =_\alpha t''\}$$

is closed under the rules defining $=_\alpha$. Only closure under the rule for λ-abstractions is non-trivial.

Have to prove (1) & (2) implies $\lambda a\, t =_\alpha \lambda a_1\, t_1$, where

(1) $((a\, a_3) \cdot t, (a'\, a_3) \cdot t') \in H \;\;\&\;\; a_3 \# (a, t, a', t')$

(2) $(a\, a_3) \cdot t =_\alpha (a_1\, a_3) \cdot t_1 \;\;\&\;\; a_3 \# (a, t, a_1, t_1)$

Now apply $=_\alpha$-rule for λ-abstractions to (2).

$t =_\alpha t' \;\&\; t' =_\alpha t'' \Rightarrow t =_\alpha t''$

Use rule induction—show that

$$H \triangleq \{(t, t') \mid (\forall t'')\; t' =_\alpha t'' \Rightarrow t =_\alpha t''\}$$

is closed under the rules defining $=_\alpha$. Only closure under the rule for λ-abstractions is non-trivial.

Have to prove (1) & (2) implies $\lambda a\, t =_\alpha \lambda a_1\, t_1$, where

(1) $((a\, a_3) \cdot t, (a'\, a_3) \cdot t') \in H \;\;\&\;\; a_3 \# (a, t, a', t')$

(2) $(a\, a_3) \cdot t =_\alpha (a_1\, a_3) \cdot t_1 \;\;\&\;\; a_3 \# (a, t, a_1, t_1)$

Done!

"Some/any" proof pattern

$$\boxed{\dfrac{(a\,a'')\cdot t =_\alpha (a'\,a'')\cdot t'}{\lambda a\,t =_\alpha \lambda a'\,t'}\ a'' \# (a, t, a', t')}$$

proof check	proof search
$(\exists a'' \in \mathbb{A})$	$(\forall a'' \in \mathbb{A})$
$a'' \# (a, t, a', t')$ &	$a'' \# (a, t, a', t') \Rightarrow$
$(a\,a'')\cdot t =_\alpha (a'\,a'')\cdot t'$	$(a\,a'')\cdot t =_\alpha (a'\,a'')\cdot t'$
$\Downarrow$	$\Uparrow$
$\lambda a\,t =_\alpha \lambda a'\,t'$	$\lambda a\,t =_\alpha \lambda a'\,t'$

“Some/any” theorem

<u>Theorem</u>. For any $S \in P_{\mathrm{fs}}(\mathbb{A})$, if $A \in P_{\mathrm{fin}}(\mathbb{A})$ supports S then the following are equivalent:

1. $(\forall a \in \mathbb{A})\ a \notin A \ \Rightarrow\ a \in S$
2. $\mathbb{A} - S$ is finite
3. $(\exists a \in \mathbb{A})\ a \notin A \ \&\ a \in S$

“Some/any” theorem

<u>Theorem</u>. For any $S \in P_{\mathrm{fs}}(\mathbb{A})$, if $A \in P_{\mathrm{fin}}(\mathbb{A})$ supports S then the following are equivalent:

1. $(\forall a \in \mathbb{A})\ a \notin A \ \Rightarrow\ a \in S$
2. $\mathbb{A} - S$ is finite
3. $(\exists a \in \mathbb{A})\ a \notin A \ \&\ a \in S$

<u>Proof</u> of 1 $\Rightarrow$ 2:

1 says $\mathbb{A} - A \subseteq S$, so $\mathbb{A} - S \subseteq A$ is finite.

“Some/any” theorem

Theorem. For any $S \in P_{\text{fs}}(\mathbb{A})$, if $A \in P_{\text{fin}}(\mathbb{A})$ supports S then the following are equivalent:

1. $(\forall a \in \mathbb{A})\ a \notin A \ \Rightarrow\ a \in S$
2. $\mathbb{A} - S$ is finite
3. $(\exists a \in \mathbb{A})\ a \notin A\ \&\ a \in S$

Proof of 2 $\Rightarrow$ 3:

If 2, then $A \cup (\mathbb{A} - S)$ is a finite subset of the infinite set $\mathbb{A}$, so there is some a in its complement, i.e. in $(\mathbb{A} - A) \cap S$.

“Some/any” theorem

Theorem. For any $S \in P_{\mathrm{fs}}(\mathbb{A})$, if $A \in P_{\mathrm{fin}}(\mathbb{A})$ supports S then the following are equivalent:

1. $(\forall a \in \mathbb{A})\ a \notin A \ \Rightarrow \ a \in S$
2. $\mathbb{A} - S$ is finite
3. $(\exists a \in \mathbb{A})\ a \notin A \ \ \& \ \ a \in S$

Proof of $3 \Rightarrow 1$:

Suppose $a \in \mathbb{A} - A$ and $a \in S$. For any other $a' \in \mathbb{A} - A$, we have $(a\, a') \cdot S = S$ (since A supports S), so $a' = (a\, a')(a) \in (a\, a') \cdot S = S$.

“Some/any” theorem

Theorem. For any $S \in P_{\mathrm{fs}}(\mathbb{A})$, if $A \in P_{\mathrm{fin}}(\mathbb{A})$ supports S then the following are equivalent:

1. $(\forall a \in \mathbb{A})\ a \notin A \;\Rightarrow\; a \in S$
2. $\mathbb{A} - S$ is finite
3. $(\exists a \in \mathbb{A})\ a \notin A \;\&\; a \in S$

Freshness quantifier: if $\phi(a)$ is a property of atoms s.t. $\{a \in \mathbb{A} \mid \phi(a)\}$ is finitely supported, write

$\boxed{\text{И}a\, \phi(a)}$ (and say “for some/any fresh a, $\phi(a)$”)

if $\begin{cases} S & \triangleq \{a \in \mathbb{A} \mid \phi(a)\} \\ A & \triangleq supp(S) \end{cases}$ satisfy 1/2/3.

Languages/α are nominal sets

For example for λ-terms, $\Lambda \triangleq \{t ::= a \mid t\,t \mid \lambda a\,t\}$:

Set of equivalence classes $\Lambda/{=_\alpha}$ with $\mathbb{G}$-action

$$\boxed{\pi \cdot [t]_\alpha \triangleq [\pi \cdot t]_\alpha}$$

is a nominal set: $[t]_\alpha$ is supported by $supp(t)$, and in fact one can prove

$$supp([t]_\alpha) = \{\text{free variables of } t\}$$

so that

$$a \# [t]_\alpha \;\Leftrightarrow\; a \text{ not free in } t$$

α-Structural recursion

<u>Theorem</u>. Given a nominal set X and

$f \in \mathbb{A} \to_{\mathrm{fs}} X$

$g \in X \times X \to_{\mathrm{fs}} X$

$h \in \mathbb{A} \times X \to_{\mathrm{fs}} X$ s.t. $(\mathsf{N}a)(\forall x \in X)\ a \mathrel{\#} h(a, x)$,

there is a unique $k \in (\Lambda/{=_\alpha}) \to_{\mathrm{fs}} X$ s.t.

$$(\forall a \in \mathbb{A})\ k[a]_\alpha = f(a)$$

$$(\forall t_1, t_2 \in \Lambda)\ k[t_1\, t_2]_\alpha = g(k[t_1]_\alpha, k[t_2]_\alpha)$$

$$(\mathsf{N}a)(\forall t \in \Lambda)\ k[\lambda a\, t] = h(a, k[t]_\alpha)$$

(and $supp(k) = supp(f, g, h)$).

α-Structural recursion

Theorem. Given a nominal set X and

$f \in \mathbb{A} \longrightarrow_{\mathrm{fs}} X$

$g \in X \times X \longrightarrow_{\mathrm{fs}} X$

$h \in \mathbb{A} \times X \longrightarrow_{\mathrm{fs}} X$ s.t. $(\text{И}a)(\forall x \in X)\ a \# h(a, x)$,

there is a unique $k \in (\Lambda/{=_\alpha}) \longrightarrow_{\mathrm{fs}} X$ s.t.

$$(\forall a \in \mathbb{A})\ k[a]_\alpha = f(a)$$

$$(\forall t_1, t_2 \in \Lambda)\ k[t_1\, t_2]_\alpha = g(k[t_1]_\alpha, k[t_2]_\alpha)$$

$$(\text{И}a)(\forall t \in \Lambda)\ k[\lambda a\, t] = h(a, k[t]_\alpha)$$

(and $supp(k) = supp(f, g, h)$).

Capture-avoiding substitution

$$(a := t)a' \triangleq \text{if } a' = a \text{ then } t \text{ else } a'$$

$$(a := t)(t_1\, t_2) \triangleq ((a := t)t_1)((a := t)t_2)$$

$$(a := t)\lambda a'\, t' \triangleq \lambda a'(a := t)t' \qquad \text{if } a' \notin fv(t, a)$$

Capture-avoiding substitution

$$\boxed{\begin{aligned}(a := t)[a']_\alpha &\triangleq f(a')\\ (a := t)[t_1 t_2]_\alpha &\triangleq g((a := t)[t_1]_\alpha, (a := t)[t_2]_\alpha)\\ (a := t)[\lambda a'\, t']_\alpha &\triangleq h(a', (a := t)[t']_\alpha) \quad \text{if } a' \# ([t]_\alpha, a)\end{aligned}}$$

is a definition by α-structural recursion of a total function $(a := t)(-) : (\Lambda/{=_\alpha}) \to (\Lambda/{=_\alpha})$ using:

$$f(a') \triangleq \text{if } a' = a \text{ then } [t]_\alpha \text{ else } [a']_\alpha$$

$$g([t_1]_\alpha, [t_2]_\alpha) \triangleq [t_1\, t_2]_\alpha$$

$$h(a', [t']_\alpha) \triangleq [\lambda a'\, t']_\alpha$$

Check: $(\text{И} a')(\forall t')\ a' \# h(a', [t']_\alpha)$?

Capture-avoiding substitution

$$\boxed{\begin{aligned} (a := t)[a']_\alpha &\triangleq f(a') \\ (a := t)[t_1 t_2]_\alpha &\triangleq g((a := t)[t_1]_\alpha, (a := t)[t_2]_\alpha) \\ (a := t)[\lambda a'\, t']_\alpha &\triangleq h(a', (a := t)[t']_\alpha) \quad \text{if } a' \mathrel{\#} ([t]_\alpha, a) \end{aligned}}$$

is a definition by α-structural recursion of a <u>total</u> function $(a := t)(-) : (\Lambda/{=_\alpha}) \to (\Lambda/{=_\alpha})$ using:

$$\begin{aligned} f(a') &\triangleq \text{if } a' = a \text{ then } [t]_\alpha \text{ else } [a']_\alpha \\ g([t_1]_\alpha, [t_2]_\alpha) &\triangleq [t_1\, t_2]_\alpha \\ h(a', [t']_\alpha) &\triangleq [\lambda a'\, t']_\alpha \end{aligned}$$

Check: $(\forall a')(\forall t')\ a' \mathrel{\#} [\lambda a'\, t']_\alpha$ ✓

α-Structural induction

<u>Theorem</u>. For any $S \in P_{\mathrm{fs}}(\Lambda/{=_\alpha})$

$$(\forall t \in \Lambda)[t]_\alpha \in S$$

holds iff

$$(\forall a \in \mathbb{A})[a]_\alpha \in S$$
$$(\forall t_1, t_2 \in \Lambda)[t_1]_\alpha \in S \;\&\; [t_2]_\alpha \in S \;\Rightarrow\; [t_1\, t_2]_\alpha \in S$$
$$(\text{И}a)(\forall t \in \Lambda)[t]_\alpha \in S \;\Rightarrow\; [\lambda a\, t]_\alpha \in S$$

Why bother?

"I'm an expert and could patch things up to full formality if pressed (but have more important & interesting things to do)."

Might be true for familiar old λ-calculus, but look at the mess in early versions of LTSs for π-calculus.

What about large-scale SOS definitions? (E.g. *Definition of SML* mixes ASTs with ASTs/α.)

What about non-experts? What about machines?

We really need **a light-weight theory of structural recursion/induction for syntax/α that doesn't stray too far from common, nominal practices.**

Have we provided one?

Assessment of α-structural recursion/induction

Assessment of α-structural recursion/induction

- Only standard foundations.

Assessment of α-structural recursion/induction

- Only standard foundations.
- Usual notion of α-equivalence on ASTs is made <u>easier</u> through use of permutations rather than non-bijective renaming.

Assessment of α-structural recursion/induction

- Only standard foundations.
- Usual notion of α-equivalence on ASTs is made easier through use of permutations rather than non-bijective renaming.
- Crucial finite support property is automatically carried along by our constructions in SOS (if we avoid choice principles)—no real work needed.

Assessment of α-structural recursion/induction

- Only standard foundations.
- Usual notion of α-equivalence on ASTs is made easier through use of permutations rather than non-bijective renaming.
- Crucial finite support property is automatically carried along by our constructions in SOS (if we avoid choice principles)—no real work needed.
- The "some/any" property formalises common practice around the use of fresh names.

Assessment of α-structural recursion/induction

- Only standard foundations.
- Usual notion of α-equivalence on ASTs is made easier through use of permutations rather than non-bijective renaming.
- Crucial finite support property is automatically carried along by our constructions in SOS (if we avoid choice principles)—no real work needed.
- The "some/any" property formalises common practice around the use of fresh names.
- Used λ-calculus as an example here, but can treat a wide class of languages with statically scoped binders over multiple flavours of name.

Assessment of α-structural recursion/induction

For non-experts?

Is the use of permutations simple enough to become part of standard practice?

(It's now part of mine!)

Assessment of α-structural recursion/induction

For machines?

Computational consequences of the nominal sets model of syntax:

- for functional programming: FreshML & Fresh O'Caml (MR Shinwell, AMP, MJ Gabbay)
- for logic programming: nominal unification (C Urban, AMP, MJ Gabbay); α-prolog (J Cheney, C Urban)
- for proof assistants: so far there is no "Fresh-HOL" or "Fresh-Coq" because...

  (legacy code, use of Hilbert ε-operator, nominal dependent type theory)

Thanks

James Cheney (Cornell), Jamie Gabbay (INRIA), Mark Shinwell & Christian Urban (Cambridge).

Further info

`www.cl.cam.ac.uk/users/amp12/freshml/`

SOS!

We need better tools for SOS.
I'd like to hear what you want from a tool for machine-assisted proof specific to the domain of SOS.