Equivariant Syntax and Semantics

Andrew M. Pitts

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Can sum up the subject of this talk in three words:

Can sum up the subject of this talk in three words:

syntax,

Can sum up the subject of this talk in three words:

syntax,
syntax,

Can sum up the subject of this talk in three words:

syntax,
syntax,
syntax!

The mathematics of syntax

- Seems of no interest to mathematicians and of little interest to logicians. (?)
- Vital for computer science — because of *symbolic computation* and *automated reasoning*.
- Has yet to reach an intellectual fixpoint for syntax involving name-binding and freshness of names.

Plan

- Review initial algebra view of abstract syntax.
- Abstract syntax is not abstract enough for name-binding and freshness of names.
- Category theory to the rescue!
- Equivariant initial algebra semantics for ‘nominal’ signatures.
- Applications to metaprogramming.

How to represent syntax?

$$\int_0^x \left(\int_1^y xy \, dx \right) + y \, dy$$

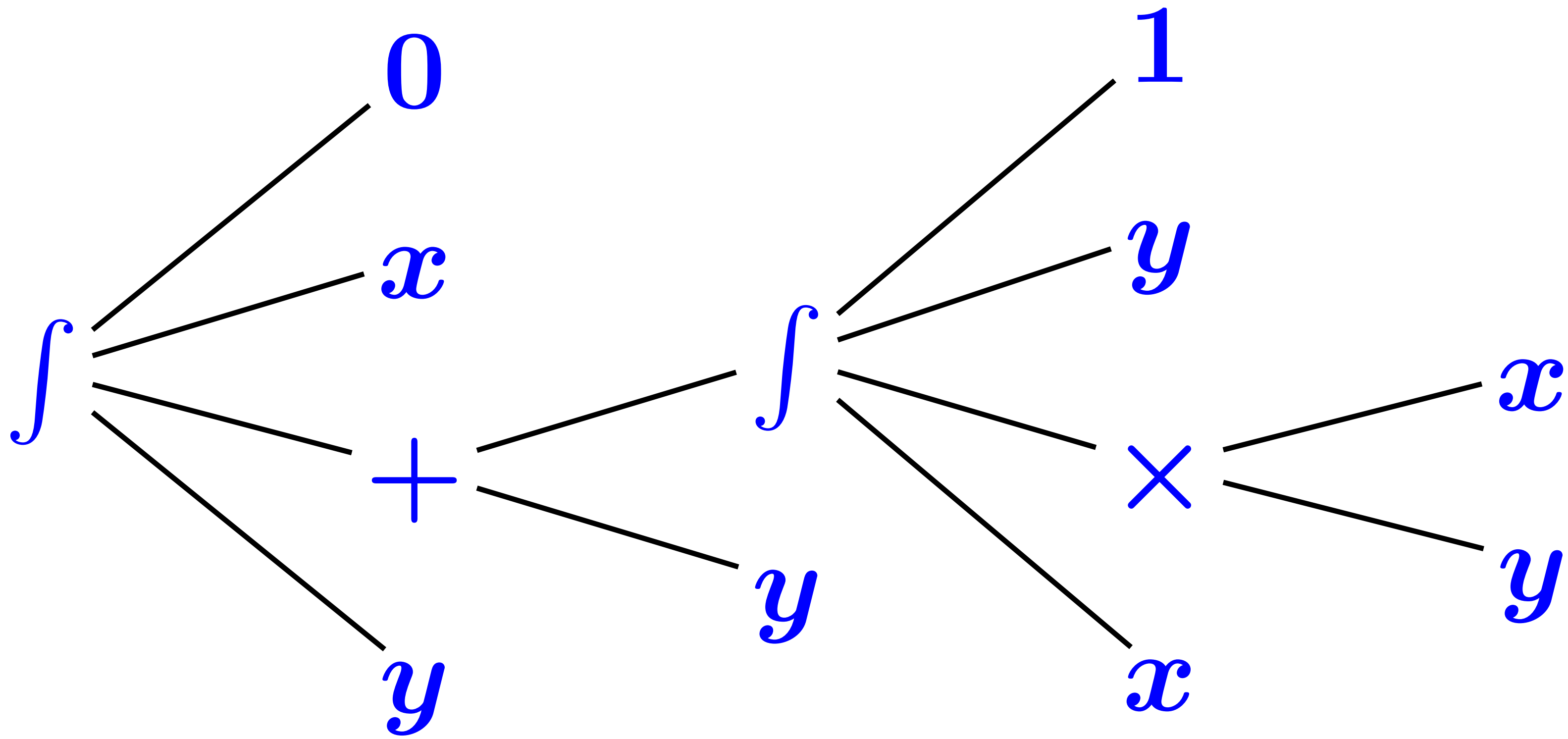
How to represent syntax?

\int	_	0	^	x	(	\int	_	1	^	y		
{	x	y	}	d	x	)	+	y	d	y		

"Concrete syntax" — sequences of tokens generated by context free grammars, etc, etc.

Not *structurally* abstract.

How to represent syntax?



"Abstract syntax" — parse trees.

Initial algebra semantics

A signature Σ determines a functorial, sum-of-products construction on sets X:

$$X \mapsto T_{\Sigma}(X) \triangleq \sum_{F \in \Sigma} X^{\mathrm{ar}(F)}$$

Initial algebra semantics

A signature Σ determines a functorial, sum-of-products construction on sets X:

$$X \mapsto T_\Sigma(X) \triangleq \sum_{F \in \Sigma} X^{\mathrm{ar}(F)}$$

single-sorted, for simplicity; so arity of each operator $F \in \Sigma$ is just the number $\mathrm{ar}(F) \in \mathbb{N}$ of its arguments

Initial algebra semantics

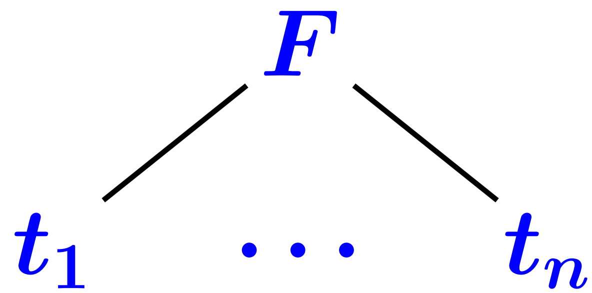
A signature Σ determines a functorial, sum-of-products construction on sets X:

$$X \mapsto T_\Sigma(X) \triangleq \sum_{F \in \Sigma} X^{\mathrm{ar}(F)}$$

typical element $(F, (x_1, \ldots, x_n))$, where operator $F \in \Sigma$ has arity $\mathrm{ar}(F) = n$ and $x_1, \ldots, x_n \in X$

Initial algebra semantics

- set $I_\Sigma \triangleq \{\text{parse trees over } \Sigma\}$
- bijection $T_\Sigma(I_\Sigma) \longrightarrow I_\Sigma$
  between $(F, (t_1, \ldots, t_n))$ in $T_\Sigma(I_\Sigma)$
  and trees in I_Σ



are determined uniquely up to bijection by their initial algebra property...

Initial algebra property

For any $T_\Sigma(X) \xrightarrow[f]{} X$

Initial algebra property

$$\begin{array}{ccc} & & I_\Sigma \\ & & \big\downarrow \overline{f} \\ T_\Sigma(X) & \xrightarrow[f]{} & X \end{array}$$

there is a unique $\overline{f}$ such that

Initial algebra property

$$
\begin{array}{ccc}
T_\Sigma(I_\Sigma) & \xrightarrow{\ \cong\ } & I_\Sigma \\
{\scriptstyle T(\overline{f})}\Big\downarrow & \text{commutes} & \Big\downarrow{\scriptstyle \overline{f}} \\
T_\Sigma(X) & \xrightarrow[\ f\ ]{} & X
\end{array}
$$

$\overline{f}(t)$ applies f iteratively, according to the structure of the tree t.

Initial algebra property

- Encompasses useful principles of

  structural recursion
  and
  structural induction

  for parse trees over Σ.

  (Generalises *primitive recursion* and *mathematical induction* for the natural numbers.)

Initial algebra property

- Encompasses useful principles of

  structural recursion
  and
  structural induction

  for parse trees over Σ.

- 'Arrow-theoretic' rather than 'element-theoretic' characterisation of parse trees—important later.

Abstract syntax is not sufficiently abstract

Parse trees take no account of variable binding.

$$\int_0^x \left(\int_1^y xy\,dx\right) + y\,dy$$

Abstract syntax is not sufficiently abstract

Parse trees take no account of variable binding.

$$\int_0^x \left(\int_1^y xy\,dx\right) + y\,dy$$

$$\int_0^x \left(\int_1^y uy\,du\right) + y\,dy$$

semantically equal expressions, represented by different, but α-convertible parse trees.

Abstract syntax is not sufficiently abstract

Parse trees take no account of variable binding.

$$\int_0^x \left(\int_1^y xy\,dx\right) + y\,dy$$

$$\int_0^x \left(\int_1^x ux\,du\right) + x\,dx$$

semantically equal expressions, represented by different, but α-convertible parse trees.

Abstract syntax is not sufficiently abstract

Parse trees take no account of variable binding.

$$\int_0^x \left(\int_1^y xy\,dx\right) + y\,dy$$

$$\int_0^x \left(\int_1^x ux\,du\right) + x\,dx$$

Isn't this a matter of *semantics* rather than syntax? No, because...

Substitution

E.g. in order to respect meaning, the result of the syntactic operation

> *substitute* $2x$ *for the free occurrence of* y *in*
> $\int_0^1 (x + y) dx$

Substitution

E.g. in order to respect meaning, the result of the syntactic operation

> *substitute* $2x$ *for the free occurrence of* y *in*
> $\int_0^1 (x + y)dx$ = y + 0.5

Substitution

E.g. in order to respect meaning, the result of the syntactic operation

> *substitute $2x$ for the free occurrence of y in*
> $\int_0^1 (x + y) dx$

is not $\int_0^1 (x + 2x) dx$, but rather, is $\int_0^1 (u + 2x) du$, where u is fresh.

The problem

Hand-coding notions of

- free and bound variables,
- renaming of bound variables,
- freshness of variables
- substitution for free variables, etc

is painful and error-prone for complex languages, or large programs.

Need better mathematical foundations leading to better automatic support for these tasks.

Wanted

Generalisation of initial algebra semantics yielding useful principles of structural recursion/induction for parse trees modulo α-conversion over a nominal signature.

Wanted

Generalisation of initial algebra semantics yielding useful principles of structural recursion/induction for parse trees modulo α-conversion over a nominal signature.

- close to informal practice (cf. "Barendregt Variable Convention")
- lead to improved languages for metaprogramming

Wanted

Generalisation of initial algebra semantics yielding useful principles of structural recursion/induction for parse trees modulo α-conversion over a nominal signature.

extension of usual notion of many-sorted algebraic signature to treat parse trees with lexically scoped binders modulo α-equivalence

Nominal signatures

- Sorts partitioned in two: sorts of bindable names (ν) and sorts of data (δ).
- Operators (F) have arities $\tau \longrightarrow \delta$, where

$$\tau ::= \nu \mid \delta \mid 1 \mid \tau,\tau \mid \nu.\tau$$

Nominal signatures

- Sorts partitioned in two: sorts of bindable names (ν) and sorts of data (δ).
- Operators (F) have arities $\tau \longrightarrow \delta$, where

$$\tau ::= \nu \mid \delta \mid 1 \mid \tau , \tau \mid \nu . \tau$$

type of pairs

type of name-abstractions

Nominal signatures

- Sorts partitioned in two: sorts of bindable names (ν) and sorts of data (δ).
- Operators (F) have arities $\tau \longrightarrow \delta$, where

$$\tau ::= \nu \mid \delta \mid 1 \mid \tau,\tau \mid \nu.\tau$$

here, for simplicity, we will assume there's just one

Nominal signatures

Closely related notions:

- *binding signatures* of Fiore, Plotkin & Turi (LICS 1999)
- *nominal algebras* of Honsell, Miculan & Scagnetto (ICALP 2001)

Nominal signatures

Closely related notions:

- *binding signatures* of Fiore, Plotkin & Turi (LICS 1999)
- *nominal algebras* of Honsell, Miculan & Scagnetto (ICALP 2001)

N.B. all these notions of signature restrict attention to iterated, but *unary* name-binding—there are other kinds of lexically scoped binder.

Example: π-calculus

sort of bindable names: ν (channels)

sort of data: π (processes)

operators:

$$\begin{aligned}
0 &: 1 \longrightarrow \pi \\
Par &: \pi, \pi \longrightarrow \pi \\
Sum &: \pi, \pi \longrightarrow \pi \\
In &: \nu, (\nu.\pi) \longrightarrow \pi \\
Out &: \nu, \nu, \pi \longrightarrow \pi \\
Tau &: \pi \longrightarrow \pi \\
Nu &: \nu.\pi \longrightarrow \pi \\
Guard &: \nu, \nu, \pi \longrightarrow \pi
\end{aligned}$$

Example: an untyped FPL

sort of bindable names: var (variables)

sort of data: exp (expressions)

operators:

$$\begin{aligned} Var &: var \longrightarrow exp \\ App &: exp, exp \longrightarrow exp \\ Fun &: var.exp \longrightarrow exp \\ Let &: exp, (var.exp) \longrightarrow exp \\ Letrec &: var.(exp, exp) \longrightarrow exp \end{aligned}$$

Example: an untyped FPL

sort of bindable names: var (variables)

sort of data: exp (expressions)

$$Var : var \longrightarrow exp$$
$$App : exp, exp \longrightarrow exp$$
$$Fun : var.exp \longrightarrow exp$$
$$Let : exp, (var.exp) \longrightarrow exp$$
$$Letrec : var.(exp, exp) \longrightarrow exp$$

$Let(t, (x.t'))$
stands for
$let\ x = t\ in\ t'$

$Letrec(x.(t, t'))$
stands for
$letrec\ x = t\ in\ t'$

Parse trees and their types over a nominal signature:

- infinitely many atoms $a : \nu$ for each sort ν of bindable names
- $() : 1$ and $\dfrac{t : \tau \qquad t' : \tau'}{(t, t') : \tau, \tau'}$
- $\dfrac{t : \tau}{a.t : \nu.\tau}$ for each atom $a : \nu$
- $\dfrac{t : \tau}{F\,t : \delta}$ if F has arity $\tau \longrightarrow \delta$

α-Equivalence, $=_\alpha$

least congruence identifying $a.t$ with $b.[a{\mapsto}b]t$ if b does not occur (at all) in t

where

$[a{\mapsto}b]t$ = rename all free occurrences of a to be b in t.

Wanted

Generalisation of initial algebra semantics yielding useful principles of structural recursion/induction for parse trees modulo α-conversion over a nominal signature.

Category theory to the rescue!

The notions underlying initial algebra semantics have purely arrow-theoretic definitions, so . . .

Category theory to the rescue!

The notions underlying initial algebra semantics have purely arrow-theoretic definitions, so change category from $\mathcal{S}et$ to one with a suitable functorial construction for modelling name-abstraction $\nu.\tau$

Category theory to the rescue!

The notions underlying initial algebra semantics have purely arrow-theoretic definitions, so change category from *Set* to one with a suitable functorial construction for modelling name-abstraction $\nu.\tau$

> not a new idea—cf. initial algebra semantics in categories of domains, in order to treat fixpoint recursion

Two candidates to replace the category of sets and functions (both in Proc. LICS'99):

- Fiore-Plotkin-Turi: category of presheaves on finite sets & functions

  —nice categorical analysis of de Bruijn indices/levels; not so nice (?) for applications

Two candidates to replace the category of sets and functions (both in Proc. LICS'99):

- Fiore-Plotkin-Turi: category of presheaves on finite sets & injections
- Gabbay-AMP: category of FM-sets ($\simeq$ 'Schanuel topos')
  —a semantics for *name-abstraction* and *freshness of names* via use of permutation actions

Why use name-permutation/swapping?

Problem of 'capture': as a total operation on parse trees, $[a \mapsto b](-)$ doesn't respect $=_\alpha$, so can't be part of a theory of terms modulo $=_\alpha$.

E.g. $b.a =_\alpha c.a$, but applying $[a \mapsto b]$

$[a \mapsto b](b.a) = b.b \neq_\alpha c.b = [a \mapsto b](c.a)$.

Why use name-permutation/swapping?

Problem of 'capture': as a total operation on parse trees, $[a \mapsto b](-)$ doesn't respect $=_\alpha$, so can't be part of a theory of terms modulo $=_\alpha$.

Traditional solution: replace $[a \mapsto b]t$ by a *more complicated*, capture-avoiding form of renaming (and substitution).

Why use name-permutation/swapping?

Problem of 'capture': as a total operation on parse trees, $[a \mapsto b](-)$ doesn't respect $=_\alpha$, so can't be part of a theory of terms modulo $=_\alpha$.

A nice alternative: use a *less complicated* form of renaming

$(a\,b)\cdot t$ = swap *all* occurrences of a and b in t

Inductive definition of $=_\alpha$

$$a =_\alpha a$$

$$() =_\alpha ()$$

$$\frac{t_1 =_\alpha t'_1 \quad t_2 =_\alpha t'_2}{(t_1, t_2) =_\alpha (t'_1, t'_2)}$$

$$\frac{t =_\alpha t'}{a.t =_\alpha a.t'}$$

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

$$\frac{t =_\alpha t'}{F\,t =_\alpha F\,t'}$$

Inductive definition of $=_\alpha$

$$a =_\alpha a$$

$$() =_\alpha ()$$

$$\frac{t_1 =_\alpha t'_1 \quad t_2 =_\alpha t'_2}{(t_1, t_2) =_\alpha (t'_1, t'_2)}$$

$$\frac{t =_\alpha t'}{a.t =_\alpha a.t'}$$

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

$$\frac{t =_\alpha t'}{F\, t =_\alpha F\, t'}$$

Freshness: "a' does not occur in t"

Category of FM-sets

Fix an infinite set $\mathbb{A}$ of 'atoms' a, b, c ...

- Objects: sets X equipped with an $\mathbb{A}$-permutation action, all of whose elements have the finite support property

  each $x \in X$ satisfies $(a\,b)\cdot x = x$ *for all but finitely many* $a, b \in \mathbb{A}$.

Category of FM-sets

Fix an infinite set $\mathbb{A}$ of 'atoms' a, b, c ...

- Objects: sets X equipped with an $\mathbb{A}$-permutation action, all of whose elements have the finite support property
- Morphisms: equivariant functions

  $$f((a\,b)\cdot x) = (a\,b)\cdot(f\,x)$$

Category of FM-sets

Fix an infinite set $\mathbb{A}$ of 'atoms' a, b, c . . .

- Objects: sets X equipped with an $\mathbb{A}$-permutation action, all of whose elements have the finite support property
- Morphisms: equivariant functions
- Freshness $a \# x$ ("a is fresh for x") is a derived notion:

  $a \# x$ iff $(a\, b)\cdot x = x$ for *all but finitely many* $b \in \mathbb{A}$.

Atom-abstractions, $\mathbb{A}.X$

quotient of $\mathbb{A} \times X$ by equivalence relation identifying (a, x) and (a', x')

iff either $a = a'$ and $x = x'$,
or $a' \# x$ and $(a\,a')\cdot x = x'$.

Atom-abstractions, $\mathbb{A}.X$

quotient of $\mathbb{A} \times X$ by equivalence relation identifying (a, x) and (a', x')

iff either $a = a'$ and $x = x'$,
or $a' \# x$ and $(a\, a') \cdot x = x'$.

Functor $\mathbb{A}.(-) :$ FM-Set $\longrightarrow$ FM-Set has excellent properties—in particular it can be used with sums and products in inductive definitions of FM-sets.

Theorem

For nominal signature Σ,

$\{$parse trees over $\Sigma\}/{=_\alpha}$
with its natural FM-sets structure

is initial algebra for associated functor
T_Σ : FM-Set $\longrightarrow$ FM-Set.

Theorem

For nominal signature Σ,

$\{$parse trees over $\Sigma\}/{=_\alpha}$
with its natural FM-sets structure

is initial algebra for associated functor
T_Σ : FM-Set $\longrightarrow$ FM-Set.

for simplicity, assume Σ has a single data sort δ and a single sort of bindable names ν

Theorem

For nominal signature Σ,

$\{$parse trees over $\Sigma\}/{=_\alpha}$
with its natural FM-sets structure

is initial algebra for associated functor
T_Σ : FM-Set $\longrightarrow$ FM-Set.

Lemma: support of α-equivalence class of a parse tree coincides with the set of free names of (any representative) parse tree.

Theorem

For nominal signature Σ,

$\{$parse trees over $\Sigma\}/{=_\alpha}$
with its natural FM-sets structure

is initial algebra for associated functor T_Σ : FM-Set $\longrightarrow$ FM-Set.

generalises usual 'sum-of-products' functor by interpreting name-abstraction arities $\nu.(-)$ as atom-abstraction functors $\mathbb{A}.(-)$

Wanted

Generalisation of initial algebra semantics yielding useful principles of structural recursion/induction for parse trees modulo α-conversion over a nominal signature.

- close to informal practice ("Barendregt Variable Convention")
- lead to improved languages for metaprogramming

Close to informal practice

- FM-Set models classical logic $+ \mathrm{ZFA} + \neg \mathrm{AC}$.
- Equivariance becomes part of the implicit mathematical infra-structure—no need to prove it case-by-case.
- Initial algebra property $\Rightarrow$ structural induction involving freshness quantifier—formalises a common informal logical pattern.

Close to informal practice

- FM-Set models classical logic $+ \mathrm{ZFA} + \neg\mathrm{AC}$.
- Equivariance becomes part of the implicit mathematical infra-structure—no need to prove it case-by-case.

"for some/any fresh name. . . "

- $\Rightarrow$ structural induction involving freshness quantifier—formalises a common informal logical pattern.

Close to informal practice

- FM-Set models classical logic $+ \mathbf{ZFA} + \neg\mathbf{AC}$.
- Equivariance becomes part of the implicit mathematical infra-structure—no need to prove it

See it at work in the Cardelli-Caires spatial process logic (TACS 2001 & CONCUR 2002)

- freshness quantifier—formalises a common informal logical pattern.

Applications to metaprogramming

Shinwell, Gabbay, AMP: FreshML = ML +

- bindable names and name-abstraction types
- name-abstraction patterns
- static freshness checking, guarantees run-time behaviour respects $=_\alpha$

Applications to metaprogramming

Shinwell, Gabbay, AMP: FreshML

See
⟨www.cl.cam.ac.uk/users/amp12/freshml/⟩

Applications to metaprogramming

Urban, Gabbay, AMP:

extension of first-order unification to parse trees mod $=_\alpha$ over a nominal signature

with applications to term-rewriting & logic programming (work in progress).

‘Syntax modulo’

Here: initial algebra semantics for syntax modulo $=_\alpha$.

Use of name-permutation (rather than renaming) leads to a rich theory with good structural recursion/induction principles for syntax modulo $=_\alpha$.

‘Syntax modulo’

Other important ways of making syntax more abstract:

- quotient by ‘structural congruence’ in process calculus (cf. the ‘Chemical abstract machine’)
- graph structures (e.g. semistructured data with references)

Are there useful notions of *structural* recursion/induction for these?

Final

¡Gracias por su atención!