# A computer science application of permutation models of set theory

Andrew M. Pitts

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Joint work with Jamie Gabbay. Copy of these slides at
⟨www.cl.cam.ac.uk/users/amp12/talks/⟩.

# Abstract

The permutation model of set theory with atoms (FM-sets) devised by Fraenkel and Mostowski in the 1930s supports notions of 'name-abstraction' and 'fresh name' that provide a new way to represent, compute with, and reason about the syntax of formal systems involving variable-binding operations. Inductively defined FM-sets involving the name-abstraction set former (together with cartesian product and disjoint union) can correctly encode syntax modulo renaming of bound variables. In this way, the standard theory of algebraic data types can be extended to encompass signatures involving binding operators. In particular, there is an associated notion of structural recursion for defining syntax-manipulating functions (such as capture avoiding substitution, set of free variables, etc) and a notion of proof by structural induction, both of which remain pleasingly close to informal practice in computer science.

# Metaprogramming

The activity of creating software systems that manipulate syntactical structures (such as programming languages or formal logics).

Examples of metaprogramming work in the Cambridge Computer Laboratory:

**HOL** (Mike Gordon *et al*)**:** assists the user to construct theorems in a polymorphic version of Church's Higher Order Logic;

**Isabelle** (Paulson *et al*)**:** generic theorem proving environment in which many different logics can be implemented.

They are implemented using Standard `ML`—a *function-oriented* programming language that arose from the Metaprogramming Language of Edinburgh LCF theorem prover—progenitor of many mechanised proof assistants.

## Standard ML

- Product of Edinburgh University LFCS (Milner et al); won the BCS Award for Technical Excellence in 1987.
- Standard and non-standard MLs (OCaml, Moscow ML) widely used for metaprogramming; also for teaching; and even systems programming (Fox Project at CMU).
- Innovative type system—motivated by theorem-proving applications, but useful in general. Aspect most relevant here: user-definable concrete data types (with corresponding pattern-matching facilities).

## Getting in a bind

Standard ML's user-declared concrete data types reduce the design of data structures for the abstract syntax (parse trees) of objects languages to a mere act of declaration.

```
datatype form = Var of string
              | Plus of form * form
              | Exp of form
              | Int of form * string
                ...
```

E.g. $\int(e^x + y)dx$ represented by

```
Int(Plus(Exp(Var("x")),Var("y")),"x")
```

But abstract syntax trees are not sufficiently abstract!—they take no account of variable binding.

## Capture-avoiding substitution and renaming bound variables

E.g. to preserve meaning, the result of substituting $x + x$ for all free occurrences of $y$ in

$$\int (e^x + y)dx + \int y^2 dy$$

should not be $\displaystyle\int (e^x + (x + x))dx + \int y^2 dy$, but rather

$$\int (e^z + (x + x))dz + \int y^2 dy \qquad \text{where } z \text{ is fresh.}$$

## Getting in a bind

Hand-coding notions of

> free and bound variables, renaming of bound variables, substitution of terms for free variables, and so on

is painful. Metaprogrammers have to ensure that their term manipulating programs respect the renaming discipline—an obligation that becomes irksome and error prone for complex object languages, or large programs.

Need better automatic support for these tasks!

## A fresh approach: `FreshML` = `ML` + freshness inference

- Deduce at compile-time (i.e. decidably) not only traditional *type information*, but also information about the 'freshness' of (object-level) variables.

- Run-time behaviour of well-typed expressions guaranteed to be insensitive to renaming bound variables.

- Users notified at compile-time if their syntax-manipulating code descends below the level of abstraction which identifies object-level expressions up to renaming bound variables.

The design of the new language arises from a new mathematical model of name abstraction that uses some 1930s set theory! — Fraenkel-Mostowski permutation model of $\mathrm{ZFA} + \neg\mathrm{AC}$.

# Permutation actions

- $S_{\mathbb{A}}$ = group of permutations of countably infinite set $\mathbb{A}$ (of 'atoms').

- An action of $S_{\mathbb{A}}$ on a set $X$ is a function

$$\begin{array}{rcl} S_{\mathbb{A}} \times X & \to & X \\ (\pi, x) & \mapsto & \pi \cdot x \end{array}$$

satisfying $id \cdot x = x$ and $\pi' \cdot (\pi \cdot x) = (\pi'\pi) \cdot x$.

- $S_{\mathbb{A}}$-set = set + action.

## Examples of $S_{\mathbb{A}}$-sets

(a) $\mathbb{A}$ itself, with action $\pi \cdot a = \pi(a)$.

(b) Inductively defined set of (abstract syntax trees for) lambda terms

$$\Lambda = \mu X \,.\, \mathtt{Var}(\mathbb{A}) \mid \mathtt{App}(X \times X) \mid \mathtt{Lam}(\mathbb{A} \times X)$$

with action $\begin{cases} \pi \cdot \mathtt{Var}(a) = \mathtt{Var}(\pi(a)) \\ \pi \cdot \mathtt{App}(M, M') = \mathtt{App}(\pi \cdot M, \pi \cdot M') \\ \pi \cdot \mathtt{Lam}(a, M) = \mathtt{Lam}(\pi(a), \pi \cdot M). \end{cases}$

(c) $pow(X)$ = subsets of $S_{\mathbb{A}}$-set $X$, with action $\pi \cdot S = \{\pi \cdot x \mid x \in S\}$.

Egs 1 & 2 have an important finiteness property not possessed by 3 (in general): all their elements *have finite support* . . .

## Finite support property of elements $x$ of an $S_{\mathbb{A}}$-set $X$

$S \subseteq \mathbb{A}$ supports $x \in X$ if for all $\pi \in S_{\mathbb{A}}$ fixing every element of $S$, we have $\pi \cdot x = x$.

$x$ is a finitely supported element of $X$ if there is some *finite* $S \subseteq \mathbb{A}$ supporting $x$. In that case there is always least such $S$, the support of $x$, $\boxed{\mathrm{supp}_X(x)}$.

Write $\boxed{a \mathbin{\#} x \quad (\text{“}a \text{ is fresh for } x\text{”})}$ to mean $a \notin \mathrm{supp}_X(x)$.

**Examples:**

- every $t \in \Lambda$ is finitely supported and $a \mathbin{\#} t$ iff $a$ does not occur in $t$.
- $S \in pow(\mathbb{A})$ is finitely supported iff it is either finite or cofinite.

## Fraenkel-Mostowski universe of FM-sets, $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$

is the inductively defined $S_{\mathbb{A}}$-class

$$\mathcal{V}_{\mathrm{FM}}(\mathbb{A}) = \mu\mathcal{X}\,.\,\mathtt{Atom}\,(\mathbb{A}) \mid \mathtt{Set}\,(pow_{\mathrm{fs}}\mathcal{X})$$

where $pow_{\mathrm{fs}}\mathcal{X}$ is the sub-$S_{\mathbb{A}}$-class of $\mathcal{X}$ consisting of subsets with finite support.

---

**Axiomatics:** $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ is a model of $\mathrm{ZFA}$ satisfying

$\mathbb{A} \notin pow_{\mathrm{fin}}(\mathbb{A})$ (because $\mathbb{A}$ is not finite),

$\forall x\,.\,\exists a \in \mathbb{A}\,.\,a\ \mathtt{\#}\ x$ (because every element has finite support),

and hence also

$\neg\mathrm{AC}$ (because $\forall S \in pow_{\mathrm{fin}}(\mathbb{A}).\exists a \in \mathbb{A}.a \notin S$, but no choice function has finite support).

We next describe why $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ is relevant to the representation, modulo $\alpha$-conversion, of the syntax of languages involving variable-binding operations. We'll use the terms of the untyped lambda calculus as a running example.

## Three versions of variable-renaming for lambda terms $M$

$\boxed{[a'/a]M}$ = capture-avoiding substitution of $a'$ for all free occurrences of $a$ in $M$ ($a, a' \in \mathbb{A}$)

$\boxed{\{a'/a\}M}$ = textual substitution of $a'$ for all free occurrences of $a$ in $M$

$\boxed{(a'\,a) \cdot M}$ = interchange of *all* occurrences (be they free, bound, or binding) of $a'$ and $a$ in $M$. (Special case of $\pi \cdot M$ for $\pi \in S_{\mathbb{A}}$ the transposition $(a'\,a)$.)

---

E.g.

if $M = (\lambda a' . a\, a')(\lambda a . a)$, then $\left\{\begin{array}{rcl} [a'/a]M & = & (\lambda a'' . a'\, a'')(\lambda a . a) \\ \{a'/a\}M & = & (\lambda a' . a'\, a')(\lambda a . a) \\ (a'\,a) \cdot M & = & (\lambda a . a'\, a)(\lambda a' . a') \end{array}\right.$

Third version is unfamiliar but *more basic* than the other two: don't need to know which constructors are binders to define it, yet it can be used to define $\alpha$-conversion . . .

Recall usual definition of $\alpha$-conversion, $=_\alpha$, as smallest congruence relation on $\Lambda$ containing $\mathtt{Lam}(a, M) =_\alpha \mathtt{Lam}(a', [a'/a]M)$.

**Theorem.** $=_\alpha$ coincides with the relation $\sim \subseteq \Lambda \times \Lambda$ inductively generated by the axioms and rules

$$\mathtt{Var}(a) \sim \mathtt{Var}(a)$$

$$\frac{M_1 \sim M_1' \qquad M_2 \sim M_2'}{\mathtt{App}(M_1, M_2) \sim \mathtt{App}(M_1', M_2')}$$

$$\frac{(a''\ a) \cdot M \sim (a''\ a') \cdot M'}{\mathtt{Lam}(a, M) \sim \mathtt{Lam}(a', M')}$$

for some/any $a''$ with $a''\ \#\ (a, M, a', M')$

**Corollary.** In $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$, the quotient $\Lambda/{=_\alpha}$ is in bijection with the inductively defined FM-set

$$\Lambda_\alpha = \mu X \,.\, \mathtt{Var}(\mathbb{A}) \mid \mathtt{App}(X \times X) \mid \mathtt{Lam}(\mathtt{[}\mathbb{A}\mathtt{]}\, X).$$

where $\mathtt{[}\mathbb{A}\mathtt{]}\, X$ is the FM-set of 'atom-abstractions' of elements of the FM-set $X$...

---

Results hold for any 'binding signature'.

**Moral:** By moving from the usual von Neumann universe of sets to the universe of FM-sets we can get a useful, purely inductive characterisation of sets of abstract syntax modulo $\alpha$-conversion.

## FM-set $[\mathbb{A}]\,X$ of atom-abstractions of an FM-set $X$

(Simplifying assumption: $\mathrm{supp}(X) = \emptyset$.)

**Introduction:** $\boxed{a\,.\,x} \in [\mathbb{A}]\,X$ iff $a \in \mathbb{A}$ and $x \in X$.

**Equality:** $a\,.\,x = b\,.\,y \in [\mathbb{A}]\,X$ iff $(c\,a)\cdot x = (c\,b)\cdot y \in X$,
for some/any $c \in \mathbb{A}$ with $c\,\#\,(a, x, b, y)$.

**Swapping:** $(a_1\,a_2)\cdot(a\,.\,x) = ((a_1\,a_2)\cdot a)\,.\,((a_1\,a_2)\cdot x)$.

Hence $\quad \mathrm{supp}(a\,.\,x) = \mathrm{supp}(x) \smallsetminus \{a\}$.

# Atom-abstraction, cont.

(Simplifying assumption: $\mathrm{supp}(X) = \mathrm{supp}(Y) = \emptyset$.)

**Elimination:** given $f \in (\mathbb{A} \times X \to Y)$ in $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ satisfying

$$\forall a \in \mathbb{A}, x \in X \ (a \mathbin{\#} f \Rightarrow a \mathbin{\#} f(a, x))$$

then for any $e \in [\mathbb{A}]X$ there is a unique element

$$\boxed{\texttt{case}\ e\ \texttt{of}\ \{a\,.\,x \texttt{ => } f(a, x)\}} \in Y$$

satisfying

**Reduction:** for all $a' \in \mathbb{A}$, $x' \in X$,

$\texttt{case}\ a'\,.\,x'\ \texttt{of}\ \{a\,.\,x \texttt{ => } f(a, x)\} = f(a'', (a''\, a') \cdot x')$,

for some/any $a'' \mathbin{\#} (f, (a'\,.\,x'))$.

## Structural recursion for the inductively defined FM-set $\Lambda_\alpha = \mu X \,.\, \mathtt{Var}(\mathbb{A}) \mid \mathtt{App}(X \times X) \mid \mathtt{Lam}(\mathtt{[}\mathbb{A}\mathtt{]}\, X)$

(For simplicity, just give 'iterative' version.)

In $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$, given $var : \mathbb{A} \to X$, $app : X \times X \to X$ and $lam : \mathbb{A} \times X \to X$ such that

for all $a \mathbin{\#} (X, lam)$ and $x \in X$, $a \mathbin{\#} lam(a, x)$

then there is a unique function $f : \Lambda_\alpha \to X$ satisfying:

- for all $a \in \mathbb{A}$, $f(\mathtt{Var}(a)) = var(a)$
- for all $t, t' \in \Lambda_\alpha$, $f(\mathtt{App}(t, t')) = app(f(t), f(t'))$
- for some/any $a \mathbin{\#} (X, var, app, lam)$ and all $t \in \Lambda_\alpha$, $f(\mathtt{Lam}(a \,.\, t)) = lam(a, f(t))$.

## Structural induction for $\Lambda_\alpha$

Given $S \subseteq \Lambda_\alpha$ in $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$, to prove $S = \Lambda_\alpha$, it suffices to show

- for all $a \in \mathbb{A}$, $\mathtt{Var}(a) \in S$
- for all $t, t' \in \Lambda_\alpha$, $(t \in S \mathbin{\&} t' \in S \Rightarrow \mathtt{App}(t, t') \in S)$
- for some/any $a \mathbin{\texttt{#}} S$ and all $t \in \Lambda_\alpha$,
  $(t \in S \Rightarrow \mathtt{Lam}(a\,.\,t) \in S)$.

This 'for some/any fresh $a \ldots$' keeps on cropping up! It is the $\mathsf{N}a\,(\ldots)$ quantifier of (Gabbay-Pitts, LICS'99) and has very pleasant logical properties.

## Example: set of free variables

Can use structural recursion for $\Lambda_\alpha$ to deduce existence of function $fv : \Lambda_\alpha \to pow_{\mathrm{fin}}(\mathbb{A})$ in $\mathcal{V}_{\mathrm{FM}}(\mathbb{A})$ satisfying

- for all $a \in \mathbb{A}$, $fv(\mathtt{Var}(a)) = \{a\}$
- for all $t, t' \in \Lambda_\alpha$, $fv(\mathtt{App}(t, t')) = fv(t) \cup fv(t')$
- for all $a \in \mathbb{A}$, $t \in \Lambda_\alpha$, $fv(\mathtt{Lam}(a\,.\,t)) = fv(t) \setminus \{a\}$.

Can use structural induction for $\Lambda_\alpha$ to prove

$fv(t) = \mathrm{supp}(t)$, for all $t \in \Lambda_\alpha$.

## Example: capture-avoiding substitution

Given $a \in \mathbb{A}$ and $t \in \Lambda_\alpha$, can use structural recursion for $\Lambda_\alpha$ to deduce existence of function $[t/a](-) : \Lambda_\alpha \to \Lambda_\alpha$ satisfying

- for all $a' \in \mathbb{A}$,
  $[t/a]\mathtt{Var}(a') = (\mathtt{if}\ a' = a\ \mathtt{then}\ t\ \mathtt{else}\ \mathtt{Var}(a'))$
- for all $t, t' \in \Lambda_\alpha$,
  $[t/a]\mathtt{App}(t', t'') = \mathtt{App}([t/a]t', [t/a]t'')$
- for some/any $a'\ \#\ (a, t)$ and all $t' \in \Lambda_\alpha$,
  $[t/a]\mathtt{Lam}(a' . t') = \mathtt{Lam}(a' . [t/a]t')$.

---

(Can use structural induction for $\Lambda_\alpha$ to prove that this agrees with usual definition under the bijection $\Lambda_\alpha \cong (\Lambda/{=_\alpha})$, but why bother with $\Lambda/{=_\alpha}$ at all now we have $\Lambda_\alpha$ with good properties?)

## Conclusion

It is possible to give an abstract account of

> *names, freshness, name-binding,*
> *renaming, substitution,* . . .
> for syntactical structures

founded upon the simple notion of name permutation (indeed, just name transposition will do).

Moreover, such a theory is useful for metaprogramming applications.

# Work in progress

- Design and implementation of `FreshML`.
- `Isabelle/FM`—machine-assisted proof for FM-set theory within Isabelle object logic (Gabbay).
- `FM-HOL`—a version of higher order logic with atoms, permutation action, finite support property, etc (Norrish, Pitts).
- Applications to structural operational semantics.

  (E.g. recent work of Luca Cardelli and Andrew Gordon on a modal logic for $\pi$-calculus and ambient calculus makes use of the И quantifier.)

**Further reading:**

- Gabbay-Pitts, Proc. LICS'99 ⟨`ftp.cl.cam.ac.uk/papers/amp12/newaas.pdf`⟩
- Pitts-Gabbay, Proc. MPC2000 ⟨`ftp.cl.cam.ac.uk/papers/amp12/metlpb.ps`⟩
- Gabbay ⟨`M.J.Gabbay@dpmms.cam.ac.uk`⟩ PhD thesis (status: submitted)