FM Techniques for Syntax-with-Binding

Murdoch J. Gabbay, October 2002

Revised October 28, 2002.

Introduction

I will talk about **names** and **binding**. We are all familiar with both of these things, but familiarity is not understanding. I will outline

- what FM is,
- what problems it was created to solve,
- some of its simpler applications.

It always takes a long time to understand a genuinely new idea. 45 minutes is not long. Whatever (positive) impression you may form of FM, in actuality it's ten times neater.

What are FM and FMG

- "FM sets" stands for "Fraenkel-Mostowski set theory". Presented in [GabbayMJ:thesis], [GabbayMJ:newaas], [GabbayMJ:newaas-jv].
- FM has become an overall label for logics and programming languages developed using FM sets (or Schanuel Topos, etc). Thus for example FreshML, Pitts' Nominal Sets, Nominal Unification.
- FMG stands for Fraenkel-Mostowski Generalised. Primitive version presented in [GabbayMJ:hotn], full version in [GabbayMJ:picfm] (pending publication) with more to follow.
- FMG is a Higher-Order Logic (HOL) and can be thought of as a HOL version of FM sets. However, it is also strictly more powerful.

Substitution as computation

The syntax of a simple λ-calculus:

(1) $$t ::= x \in \mathbb{A} \mid t\,t \mid \lambda x.t \mid 0 \mid \mathrm{Succ}(t) \mid t + t.$$

Evaluation rules:

$$\frac{}{x \Downarrow x} \qquad \frac{}{\lambda x.t \Downarrow \lambda x.t}$$

$$\frac{t_1 \Downarrow \lambda x.s \quad t_2 \Downarrow V \quad s[V/x] \Downarrow W}{t_1 t_2 \Downarrow W} \qquad \ldots$$

Thus we see that substitution models computation. Do we have a good definition of substitution?

Capture-avoiding substitution (not rigorous!)

$$
\begin{array}{lll}
[s/x]x & \stackrel{\text{def}}{=} & s \\
[s/x]y & \stackrel{\text{def}}{=} & y \\
[s/x](t_1\, t_2) & \stackrel{\text{def}}{=} & [s/x]t_1\, [s/x]t_2 \\
[s/x]\lambda x.t & \stackrel{\text{def}}{=} & \lambda x.t \\
[s/x]\lambda y.t & \stackrel{\text{def}}{=} & \lambda y'.[s/x][y'/y]t \quad \text{for } y' \text{ fresh.}
\end{array}
\tag{2}
$$

Last clause: substitution capture-avoiding. Thus for $t = x$ and $s = y$,

$$[y/x]\lambda y.x \neq \lambda y.y \quad \text{but} \quad [y/x]\lambda y.x = \lambda y'.y$$

for some/any fresh y'. What does 'fresh' mean? Not rigorous mathematical specification, and cannot be programmed.

Problem appears elsewhere

Similar phenomenon for, say, inlining optimiser:

```
let y=5 in
   let x=f(y) in
      let y=7 in <x,x,y>
```

rewrites to

```
let y=5 in
   let y'=7 in <f(y),f(y),y'>
```

Note here how we rename y to y' in the inner binding, so as not to accidentally capture it.

Example $\neq$ the λ-calculus: 1/2

Proving weakening for some type system:

(3) $\forall \Gamma, t, T, y, X.$

$$\Gamma \vdash t : T \ \wedge \ y \notin Dom(\Gamma) \implies \Gamma, y : Y \vdash t : T.$$

This says:

If we know $\Gamma \vdash t : T$

then for (fresh!) y, we know $\Gamma, y : Y \vdash t : T.$

Example $\neq$ the λ-calculus: 2/2

An inductive proof on $\vdash$ interacts badly with a rule such as

$$\frac{\Gamma, x : X \vdash t : T}{\Gamma \vdash \lambda x : X.t : X \to T} \quad x \notin Dom(\Gamma)$$

because we can weaken the conclusion with x but not the assumption. Thus in an inductive proof with simple inductive hypothesis, we assume hyp. of $\Gamma, x : X \vdash t : T$ and with rule above we obtain

If we know $\Gamma \vdash \lambda x : X.t : X \to T$

then for fresh $y \neq x$, we know $\Gamma, y : Y \vdash \lambda x : X.t : X \to T$.

But we need also $y = x$ to obtain hyp. Bummer. OK so it's simple, but it's in a sense characteristic and it's *not to do with* binding on the right of the sequent.

First application of FM: equivariance 1/3

We have atoms $\mathbb{A}$. For any permutation π of $\mathbb{A}$ we can apply π to a term t, predicate P, function f, ... Thus $\pi \cdot t$, $\pi \cdot P$, $\pi \cdot f$, ... This satisfies equivariance:

(Equiv) $$\pi \cdot F(x_1, \ldots, x_n) = F(\pi \cdot x_1, \ldots, \pi \cdot x_n).$$

For predicates this becomes

(Equiv) $$P(x_1, \ldots, x_n) \iff P(\pi \cdot x_1, \ldots, \pi \cdot x_n)$$

(because $\pi \cdot \top = \top$ and $\pi \cdot \bot = \bot$).

Now consider

(4) $$\Phi(\Gamma, t, T, y, Y) \stackrel{\text{def}}{=} \Gamma \vdash t : T \implies \Gamma, y : Y \vdash t : T.$$

First application of FM: equivariance 2/3

$$\Phi(\Gamma, t, T, y, Y) \stackrel{\text{def}}{=} \Gamma \vdash t : T \implies \Gamma, y : Y \vdash t : T.$$

Suppose we know

$$\forall y, Y.\, y \notin Dom(\Gamma) \wedge y \neq x \implies \Phi((\Gamma, x : X), t, T)$$

and suppose $y \notin Dom(\Gamma)$. We want to conclude

$$\Phi(\Gamma, \lambda x : X.t, T, y, Y).$$

Problem: the side condition excludes the case $y = x$. So suppose $y = x$, what do we do?

For some other $z \neq x$, $\Phi(\Gamma, \lambda x : X.t, T, z, Y)$.

First application of FM: equivariance 3/3

Now apply equivariance:

$$\Phi(\Gamma, \lambda x{:}X.t, T, z, Y) \iff \Phi(\pi{\cdot}\Gamma, \pi{\cdot}(\lambda x{:}X.t), \pi{\cdot}T, \pi{\cdot}z, \pi{\cdot}Y)$$

for $\pi = (z\ x)$ the transposition of z and x.

- $(z\ x) \cdot \Gamma = \Gamma$ since $x, z \notin Dom(\Gamma)$.
- $(z\ x) \cdot \lambda x : X.t =_\alpha \lambda x : X.t$ since $x, z \notin FV(\lambda x : X.t)$ (remarkable property of permutations).
- $(z\ x) \cdot T = T$ because types are not dependent (suppose). Similarly for Y.
- $(z\ x) \cdot z = x$.

We deduce $\Phi(\Gamma, \lambda x : X.t, T, x, Y)$ as required.

Remarkable property of permutation

For some (sorry!) λ-calclulus,

$$z, x \notin FV(s) \implies s =_\alpha (z\ x) \cdot s.$$

For example:

(5)
$$\lambda f.\lambda x.f(x) =_\alpha \lambda x.\lambda f.x(f) \quad \text{compare:}$$
$$\lambda f.\lambda x.f(x) \neq_\alpha \lambda f.\lambda f.f(f) \quad [f/x].$$

This kind of reasoning turns up in nature. Last Tuesday, before coming to France, I spoke to two researchers about using transposition to rename variable names in results to get round awkwardnesses to do with $x \notin FV(t)$. They independently—from each other and from myself—identified transposition as *the* lemma they needed.

It gets better

Weakening for some type system, revised:

(6) $$\forall \Gamma, t, T, X.\, И y.\ \Gamma \vdash t : T \implies \Gamma, y : Y \vdash t : T.$$

With rule

$$\forall \Gamma.\, И x.\, \forall t, T.\, \frac{\Gamma, x : X \vdash t : T}{\Gamma \vdash \lambda x : X.t : X \to T}.$$

This avoids using permutation at all.

И Quantifier

The И ('new') quantifier in FM is defined by:

$$И a.\, P(a) \stackrel{\mathrm{def}}{=} \exists L \in \mathcal{P}_{cofin}(\mathbb{A}).\, \forall a \in L.\, P(a).$$

Here $\mathcal{P}_{cofin}(\mathbb{A})$ is the set of cofinite sets of atoms. L is cofinite when $\mathbb{A} \setminus L$ is finite.

From properties of finite and cofinite sets it follows

(7) $$И a.\, P(a) \wedge И a.\, Q(a) \iff И a.\, (P(a) \wedge Q(a)).$$

Consider for example $P(a) = a \in FV(s)$ and $Q(a) = a \in FV(t)$ for s, t in some datatype of terms. Note how semantics gives rise to a new entity И in a new logic.

И and Negation

Theorem 1: Every $X \subseteq \mathbb{A}$ is either finite or cofinite:

$$\mathcal{P}(\mathbb{A}) = \mathcal{P}_{fin}(\mathbb{A}) + \mathcal{P}_{cofin}(\mathbb{A}).$$

Counter-intuitive: “what happened to the rest of the powerset then?”. It does not exist in the FM universe, though of course it exists in a model of FM inside another, traditional, universe.

This ensures that

(8) $$И a.\, P(a) \Rightarrow И a.\, Q(a) \iff И a.\, (P(a) \Rightarrow Q(a))$$

(9) $$И a.\, \neg P(a) \iff \neg И a.\, P(a).$$

Another example: $=_\alpha$ 1/2

We now apply this to give an improved definition of α-equivalence $=_\alpha$ for a datatype of λ-terms

$$\Lambda \stackrel{\text{def}}{=} \mathbb{A} + \Lambda \times \Lambda + \mathbb{A} \times \Lambda.$$

$=_\alpha$ is given by

$$\begin{array}{lll} x =_\alpha y & \Leftrightarrow & x = y \\ s_1\, s_2 =_\alpha t_1\, t_2 & \Leftrightarrow & s_1 =_\alpha t_1 \wedge s_2 =_\alpha t_2 \\ \lambda x.s =_\alpha \lambda y.t & \Leftrightarrow & И z.\, (z\ x) \cdot s =_\alpha (z\ y) \cdot t. \end{array} \quad (10)$$

Another example: $=_\alpha$ 2/2

The proof of even a simple property, e.g. transitivity of $=_\alpha$, is now (inductive and the interesting clause is)

$$\begin{array}{rl} & \big(\mathit{И} n.\, (n\ a) \cdot s =_\alpha (n\ b) \cdot t\big) \wedge \\ & \big(\mathit{И} n.\, (n\ b) \cdot t =_\alpha (n\ c) \cdot u\big) \\ \Longleftrightarrow & \big(\mathit{И} n.\, (n\ a) \cdot s =_\alpha (n\ b) \cdot t =_\alpha (n\ c) \cdot u\big), \end{array}$$

whereas without $\mathit{И}$ we must use something like "$n_1 \notin FV(s) \cup FV(t)$" and "$n_2 \notin FV(t) \cup FV(u)$", and then use equivariance to rename n_1 and n_2 to some common n fresh for s, t, and u.

Abstraction types

Finally, we can get rid of $=_\alpha$ entirely (if we want to).

$$\Lambda_\alpha \stackrel{\text{def}}{=} \mathbb{A} + \Lambda_\alpha \times \Lambda_\alpha + [\mathbb{A}]\Lambda_\alpha\,.$$

$[\mathbb{A}]\Lambda$ is an FM abstraction type. Elements of $[\mathbb{A}]T$ are like elements of T with a distinguished bound atom. Thus Λ_α is isomorphic to $\Lambda/=_\alpha$ but is also inductive. The $\mathrm{FreshML}$ type system accommodates abstraction types.

Capture-avoiding substiution (rigorous)

Capture-avoiding substitution (in for example FreshML) becomes:

```
sub s a var(a) => s
sub s a var(b) => b
sub s a app(t1,t2)
                => app(sub s a t1,sub s a t2)
sub s a lam(<n>t)
                => lam(<n>(sub s a t))
```

But what is the type system of such a language; we must ensure the fresh choice of n does not escape its scope. What about the interaction with state? This is the FreshML project.

FM offers a semantics for binding which illuminates the phenomenon it models. It guides us to type systems, logics, and other formal environments, with facilities such as the three listed in Slide 20.

We need:

We require support for:

- “Choose a fresh atom” ($\mathcal{N}$).
- “Call the bound atom n” (‘in $\lambda n.t$’).
- “Bind n in t” (‘form $\lambda n.t$ from t’).

This allows us to, for example, give semantics to a program such as

```
case x' of <n>x => <n>f(x)
```

This is the capture-avoiding application of `f` under the binder: n is chosen fresh, f applied to the body, then n is rebound. The FM semantics allows us to develop a type system to prevent n inappropriately escaping the scope of such a pattern-matching, for example in

```
case x' of <n>x => n
```

Not just λ-calculus

We have used the λ-calculus as a running example. Do not leave with the impression that this talk is about the λ-calculus. Far from it. In my most recent paper (with the referees) I apply sophisticated versions of these techniques to analyse the syntax and semantics of the π-calculus, a process calculus for modelling distributed, communicating systems which generate fresh tokens during computation.

We can analyse syntax *and* semantics because the semantics of the π-calculus is syntax-based, and can suffer very badly from problems caused by binding. The semantics in question are designed for model-checking and verification (e.g. of bisimulations).

In my paper I claim to have brought order to the confusion, and to have opened the door to using FM languages to code novel and effective algorithms on the models I build.

Names (and binding) are everywhere

- Syntax and operational semantics (e.g. λ-calculus, PCF, $\lambda\sigma$-calculus, π-calculus, graphs with hidden names (e.g. XML), ...).
  Difficulties with defs: capture-avoiding substitution thus evaluation, type weakening, logics & langs for manipulating graphs with local names, even model theory ...

- Implementations of logics and calculi (Isabelle, HOL, abstract machines for calculi above ...).
  Difficulties with defs & algorithms: de Bruijn datatypes with shift functions make papers and programs unreadable and possibly bugged, name-carrying terms force hand-coding of α-equivalence, abstract machines rendered complex, ...

- Denotational semantics (models of π-calculus processes). Model theory?

The six axioms of FMG

Hypothesise **type of atoms** $\mathbb{A}$. Definitionally extend with type of **permutations** $P_{\mathbb{A}}$, bijective $f : \mathbb{A} \to \mathbb{A}$. Axioms:

(Act-$\mathbb{A}$)	$\pi \cdot a$	$= \pi(a : \mathbb{A})$
(Act-$\circ$)	$\pi \cdot \pi' \cdot x$	$= (\pi \circ \pi') \cdot x$
(Act-**Id**)	$\mathbf{Id} \cdot x$	$= x$
(Eqv1)	$\pi \cdot f(x)$	$= (\pi \cdot f)(\pi \cdot x)$
(Eqv2)	$\pi \cdot c$	$= c$ $\quad c$ a closed term

(Small) $\quad \exists A \in \mathbb{A}^{\mathcal{S}}.\ A\ supports\ x$

Here $\circ$ denotes function composition. $\pi(a)$ denotes value of π as a function at a. $\mathbb{A}^{\mathcal{S}} \subseteq \mathcal{P}(\mathbb{A}) \stackrel{\text{def}}{=} \mathbb{A} \to \mathbb{B}$ can be finite sets $\mathcal{P}_{fin}(\mathbb{A})$.

The Plan

The axioms above define a Higher-Order Logic (set-theoretic version exists too, if we prefer). These foundational systems can be used to interpret functions, state spaces, models, graphs, algorithms, automata, programming languages, logics, and so on. We interpret (say) a programming language in an FM universe and use the extra FM structure to encode binding. We use this encoding of binding to guide us in the design of, say, a pattern-matching discipline and type system which provides, in a safe and useful way, the facilities of Slide 20. From the theory emerges useful practice.

It seems to work. For example, FreshML. Other work being developed too, for example Pitts' Nominal Logic, my recent work on the π-calculus, and work by Urban Pitts and myself on unification of logics-with-binding.