Reasoning with Names

Ian Stark

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

Overview of talk

- Examples of names and naming in computer science
- Mathematical models for names: nu-calculus, $\mathcal{S}et^{\mathcal{I}}$
- Metalogics and mechanised reasoning: HOAS, Theory of Contexts
- FM-sets, FreshML and nominal logic: $И\mathfrak{a}.\phi(\mathfrak{x},\mathfrak{a})$

What's in a name?

The idea of a *name* arises repeatedly across computer science, as an abstract piece of data that carries identity but little else. Typically, names can be compared with each other, and there is an unlimited supply of fresh names, but that is all.

Names are useful, convenient, and often very comfortable to reason about informally, but turn out to be tremendously slippery in formal reasoning.

Some uses for names in CS

- Programming: local variables; procedure parameters; $\lambda x.M$; α-conversion.
- Logic: quantifiers $\forall x.\phi$, $\exists y.P$.
- Objects: identity; references; pointers.
- Security: nonces; privacy; authentication.
- Communication: channels, TCP/IP sockets, thread IDs, π-calculus $(\nu x)P$.
- Distributed systems: locations, namespaces.

Object identity in Java

```
private static String capital (String country) {
  if (country == "Scotland") return "Edinburgh";
  else if (country == "France" ) return "Paris";
  else return "unknown"; }
...
String country = "Scotland";
System.out.println("The capital of "+country+
                   " is "+capital(country));

// Prints "The capital of Scotland is Edinburgh"
```

Object identity in Java

```
private static String capital (String country) {
  if (country == "Scotland") return "Edinburgh";
  else if (country == "France" ) return "Paris";
  else return "unknown"; }
...
String country = "Scotland";
System.out.println("The capital of "+country+
                   " is "+capital(country));
```

Object identity in Java

```
private static String capital (String country) {
  if (country == "Scotland") return "Edinburgh";
  else if (country == "France" ) return "Paris";
  else return "unknown"; }
...
String country = in.readline();
System.out.println("The capital of "+country+
                   " is "+capital(country));
```

Object identity in Java

```
private static String capital (String country) {
  if (country == "Scotland") return "Edinburgh";
  else if (country == "France" ) return "Paris";
  else return "unknown"; }
...
String country = in.readline();
System.out.println("The capital of "+country+
                   " is "+capital(country));

// Prints "The capital of Scotland is unknown"
```

Everything is an object, unfortunately

A string literal like `"Scotland"` in Java is really

```
new String( ... ).intern()
```

executed at class load time (yuk).

The temptation is just to give up and assume that all is lost; but there remain useful equivalences like:

```
String a = "Scotland";          String b = "France";
String b = "France";      ≈     String a = "Scotland";
```

What's the difficulty?

Concrete implementation of names requires care, but is generally manageable: integers, addresses, some choice of globally unique ID.

Informal reasoning is also fairly natural: be aware of aliasing, keep names distinct, and everything will be OK.

Yet to make this formal, or to mechanise reasoning about names, turns out to be surprisingly hard.

Names and functions

Often the problem is not names themselves, but capturing how they interact with other features. For example, the *nu-calculus* combines a λ-calculus of higher-order functions with names.

$\lambda x.x$	identity function
$\nu n.M$	term M using fresh name n
$\mathrm{if}\, x = n\, \mathrm{then}\, M\, \mathit{else}\, M'$	compare names

Nu-calculus examples

The nu-calculus has an operational semantics and a notion '$\approx$' of observational equivalence between terms.

$$\nu n.\nu m.(n = m) \approx false$$

$$(\lambda x.x = x)(\nu n.n) \approx true$$

$$\lambda f.\nu n.(f\,n) \not\approx \nu n.\lambda f.(f\,n) \quad : (name \to bool) \to bool$$

$$\nu n.(\lambda x.x = n) \approx \lambda x.false \quad : name \to bool$$

Models for names

We can add names to models by indexing structures. For example $\mathrm{B} \in \mathcal{S}et^{\mathcal{I}}$ has for any set of names s the set $\mathrm{B}(s)$ of values using names from s.

- $\mathcal{S}et^{\mathcal{I}}$ – nu-calculus
- $\mathcal{S}et^{\mathcal{I}}$, $\mathrm{Cpo}^{\mathcal{I}}$, $\mathrm{Prof}^{\mathcal{I}}$ – π-calculus
- $\mathcal{S}et^{\mathcal{S}}$ – Idealized Algol
- $\mathcal{S}et^{\mathcal{V}}$ – Abstract syntax with binders

The *Schanuel topos* is a subcategory of $\mathcal{S}et^{\mathcal{I}}$ equivalent to sets with a permutation action.

Reasoning about names

A sound and adequate model gives a valid reasoning method, but it can be hard work. Other methods include:

- *Logical relations* between name sets or state sets
  e.g. proving correctness of a memoisation operator.
- *Separation logic* for heaps and pointers; $\phi * \psi$, $\phi \mathrel{-\!\!*} \psi$
  e.g. in-place list reversal, graph marking.
- *Bunched implications* for all kinds of resources
  e.g. $\phi * \psi$, $\phi \wedge \psi$, $\forall_{new} x.\phi(x)$.

This leads us to look for *metalogics* that provide support for reasoning about names and binding.

Working with binders

$$\begin{array}{lll} \mathtt{datatype}\ \mathrm{Term} = \mathit{var}\ \ \mathtt{of}\ \mathrm{Name} & & x \\ \quad | \ \ \mathit{app}\ \mathtt{of}\ \mathrm{Term} * \mathrm{Term} & & (MN) \\ \quad | \ \ \mathit{lam}\ \mathtt{of}\ ? & & \lambda x.M \end{array}$$

We seek to fill in the '?' so as to give:

- uniform behaviour under α-conversion;
- recursively defined functions on Term;
- proof by induction over the structure of Term.

"*In this situation the common practice of human provers is to say one thing and do another*"

Approaches to formalising binding

- de Bruijn indices.
- $? = \mathrm{Name} * \mathrm{Term}$.

  Reprove α-conversion for each object logic.
- $? = \mathrm{Term} \to \mathrm{Term}$.
- $? = \mathrm{Name} \to \mathrm{Term}$.

  Issues with recursion, induction and AC!
- Fraenkel-Mostowski set theory.

  Requires reworking everything, but once only.

FM set theory

Originally devised to prove the independence of the Axiom of Choice from other axioms of *ZF with atoms* (ZFA).

Given an infinite set of atoms $\mathbb{A}$, we take sets X with an action of $\mathrm{perm}(\mathbb{A})$ such that all $x \in X$ have *finite support*:

$$\mathrm{supp}(x) = \bigcap \{ w \mid \forall \pi \in \mathrm{perm}(\mathbb{A}) \,.\, \pi|_w = \mathrm{id}_w \Rightarrow \pi \cdot x = x \}$$

All constructions on FM sets are *equivariant*.

A new *abstraction* set former $[\mathbb{A}]X$ provides an inductive type to fill our gap: $? = [\mathrm{Name}]\mathrm{Term}$.

Fresh ML

www.freshml.org

"... a new language derived from Standard ML which provides superior facilities for writing software systems which manipulate syntax involving binding operations."

```
val identity = let fresh x:Var in Fn(<x>(Var x)) end

fun subst (x, e, Var y) = if x#y then Var y else e
  | subst (x, e, Fn(<y>e1)) = Fn(<y>(subst(x, e, e1)))
  | subst (x, e, App(e1,e2)) =
                  App(subst(x, e, e1), subst(x, e, e2))
```

Nominal logic

A first-order theory of FM sets. Axioms cover the action of swaps $(a\ b)$ and properties of *freshness* $a \# x$ like:

$$a \# x \wedge a' \# x \implies (a\ a') \cdot x = x$$

leading to the *freshness quantifier*:

$$И a.\phi \overset{\Delta}{\iff} \exists a.(a \# \vec{x}) \wedge \phi \iff \forall a.(a \# \vec{x}) \Rightarrow \phi$$

We can then, for example, state η-conversion as

$$\forall t{:}Term\,.\,И a{:}Var\,.\,t = lam(a, app(t, var(a)))\,.$$

Some open areas

- Higher-order nominal logic; FM type theory.
- Induction, recursion and choice axioms in HOAS.
- Bringing more powerful techniques like logical relations into the metalogic.
- Practical experience in applying these metalogics to all of the name uses given right back at the beginning...