Generative Names and Dependent Types

Andrew Pitts

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Generative Names and Dependent Types: from FreshML to 'FreshAgda'

Andrew Pitts

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

What did FreshML give the world?

Shinwell+AMP+Gabbay
ICFP 2003

What did FreshML give the world?

HOPE

What did FreshML give the world?

HOPE

higher-order functional programming

$+$

generative names — that are permutable

$\nu a.e$ — `swap` a, b `in` e

What did FreshML give the world?

LOVE

What did FreshML give the world?

LOVE

lots of very elegant
binder-manipulating algorithms
expressed in a familiar ‘nameful’ way

Inductive types with α-abstraction

```
names Var --a type of permutable, generative names

data Term where                 --inductive type of λ-terms mod α
  V : Var -> Term               --variable
  A : (Term × Term)-> Term --application term
  L : (Var . Term) -> Term  --λ-abstraction term

_/_ : Term -> Var -> Term -> Term --capture-avoiding substitution
(t / x)(V x1) = if x = x1 then t else V x1
(t / x)(A(t1 , t2)) = A((t / x )t1 , (t / x )t2)
(t / x)(L(x1 . t1)) = L(x1 . (t / x)t1)
```

Can freely mix `_ . _` and `_ -> _` to get more subtle examples (e.g. for NbE).

Inductive types with α-abstraction

Underlying calculus:

introduction: $\alpha a.\, e$ (α-abstraction)

elimination: $e \mathbin{@} e'$ (concretion)

reduction: $\boxed{(\alpha a.\, e) \mathbin{@} e' \to \nu a.\, (\mathtt{swap}\; a\,,e'\; \mathtt{in}\; e)}$ $a \mathbin{\#} e'$

What did FreshML give the world?

HOPE

higher-order functional programming

$+$

generative names   that are permutable

$\nu a.e$ swap a, b in e

ML & Haskell already have this, but not (??) this

& no future in re-engineering general-purpose HOFLs (Shinwell's Fresh OCaml is no longer supported)—be domain-specific instead

Aim

Constructive Type Theory + generative, permutable names: combine (total) FreshML with Agda/Coq.

Application domain: formal proofs about operational semantics.

Aim

Constructive Type Theory + generative, permutable names: combine (total) FreshML with Agda/Coq.

Application domain: formal proofs about operational semantics.

Design criteria:

'do' notation is unnecessarily sequential for generative names

- [ease-of-use] no monad syntax

Aim

Constructive Type Theory + generative, permutable names: combine (total) FreshML with Agda/Coq.

Application domain: formal proofs about operational semantics.

cf. previous work on nominal type theory by Schöpp-Stark and Cheney

Design criteria:

- [ease-of-use] no monad syntax
- [ease-of-use] no bunched contexts, just ν

Aim

Constructive Type Theory + generative, permutable names: combine (total) FreshML with Agda/Coq.

Application domain: formal proofs about operational semantics.

cf. JACM 53(2006)459–506

Design criteria:

- [ease-of-use] no monad syntax
- [ease-of-use] no bunched contexts, just ν
- [technical] Curry-Howard for nominal logic's freshness quantifier: proofs of α-structural induction $=$ α-structurally recursive programs

Dependently typed α-abstraction

$$\text{И-formation: } \frac{\Gamma, a : \mathtt{Name} \vdash A : \mathtt{Set}}{\Gamma \vdash \text{И} a.\, A : \mathtt{Set}}$$

- For simplicity, assume just one type $\mathtt{Name}$ of names and write $\text{И}a.\,A$ instead of $\text{И}a : \mathtt{Name}.\,A$.
- When a does not occur in A, then $\text{И}a.\,A$ should be like FreshML's type $\mathtt{Name}\,.\,A$ of α-abstractions (cf. $(x : A) \to B$ versus $A \to B$).

Dependently typed α-abstraction

И-formation: $\dfrac{\Gamma, a : \mathtt{Name} \vdash A : \mathtt{Set}}{\Gamma \vdash \text{И} a.\, A : \mathtt{Set}}$

И-introduction: $\dfrac{\Gamma, a : \mathtt{Name} \vdash e : A}{\Gamma \vdash \alpha a.\, e : \text{И} a.\, A}$

И-elimination: $\dfrac{\Gamma \vdash e : \text{И} a.\, A \qquad \Gamma \vdash e' : \mathtt{Name}}{\Gamma \vdash e \,@\, e' : \nu a.\, (\mathtt{swap}\ a\,, e'\ \mathtt{in}\ A)}$

Dependently typed α-abstraction

$$\text{И-formation:}\quad \frac{\Gamma, a : \mathtt{Name} \vdash A : \mathtt{Set}}{\Gamma \vdash \text{И}a.\,A : \mathtt{Set}}$$

$$\text{И-introduction:}\quad \frac{\Gamma, a : \mathtt{Name} \vdash e : A}{\Gamma \vdash \alpha a.\,e : \text{И}a.\,A}$$

$$\text{И-elimination:}\quad \frac{\Gamma \vdash e : \text{И}a.\,A \qquad \Gamma \vdash e' : \mathtt{Name}}{\Gamma \vdash e \mathbin{@} e' : \nu a.\,(\mathtt{swap}\; a\,,e'\;\mathtt{in}\; A)}$$

$\boxed{\text{permutative, not substitutive, dependency types on names}}$

Dependently typed α-abstraction

Ꙥ-formation: $\dfrac{\Gamma, a : \mathtt{Name} \vdash A : \mathtt{Set}}{\Gamma \vdash Ꙥ a.\, A : \mathtt{Set}}$

Ꙥ-introduction: $\dfrac{\Gamma, a : \mathtt{Name} \vdash e : A}{\Gamma \vdash \alpha a.\, e : Ꙥ a.\, A}$

Ꙥ-elimination: $\dfrac{\Gamma \vdash e : Ꙥ a.\, A \qquad \Gamma \vdash e' : \mathtt{Name}}{\Gamma \vdash e \mathbin{@} e' : \nu a.\, (\mathtt{swap}\; a\,,e'\; \mathtt{in}\; A)}$

generative names in types

Dependently typed α-abstraction

И-formation: $\dfrac{\Gamma, a : \mathtt{Name} \vdash A : \mathtt{Set}}{\Gamma \vdash \text{И}a.\,A : \mathtt{Set}}$

И-introduction: $\dfrac{\Gamma, a : \mathtt{Name} \vdash e : A}{\Gamma \vdash \alpha a.\,e : \text{И}a.\,A}$

И-elimination: $\dfrac{\Gamma \vdash e : \text{И}a.\,A \qquad \Gamma \vdash e' : \mathtt{Name}}{\Gamma \vdash e \mathbin{@} e' : \nu a.\,(\mathtt{swap}\; a\,, e'\; \mathtt{in}\; A)}$

И-equality: $\dfrac{\Gamma, a : \mathtt{Name} \vdash e : A \qquad \Gamma \vdash e' : \mathtt{Name}}{\begin{array}{l}\Gamma \vdash \ (\alpha a.\,e) \mathbin{@} e' = \nu a.\,(\mathtt{swap}\; a\,, e'\; \mathtt{in}\; e) \\ \qquad\qquad\qquad : \ \nu a.\,(\mathtt{swap}\; a\,, e'\; \mathtt{in}\; A)\end{array}}$

what does '=' mean for expressions with generative names?

Decidable equality for generative expressions

$$
\begin{array}{rcll}
\nu a.\, e & = & e & (a \# e) \\
\nu a.\, \nu b.\, e & = & \nu b.\, \nu a.\, e & \\
E[\nu a.\, e] & = & \nu a.\, E[e] & (a \# E) \\
(\lambda x \to e)\, v & = & e[v/x] & \text{[Plotkin's } \beta v\text{]} \\
& \vdots & &
\end{array}
$$

evaluation contexts: $E ::= \bullet \mid E\, e \mid v\, E \mid \nu a.\, E \mid \cdots$
expressions: $e ::= x \mid a \mid \lambda x \to e \mid e\, e \mid \nu a.\, e \mid \cdots$
canonical forms: $v ::= a \mid \lambda x \to e \mid u \mid \cdots$
neutral forms: $u ::= x \mid u\, v \mid \cdots$

Decidable equality for generative expressions

$$
\begin{array}{rcll}
\nu a.\, e & = & e & (a \,\#\, e) \\
\nu a.\, \nu b.\, e & = & \nu b.\, \nu a.\, e & \\
E[\nu a.\, e] & = & \nu a.\, E[e] & (a \,\#\, E) \\
(\lambda x \to e)\; v & = & e[v/x] & \text{[Plotkin's } \beta v\text{]} \\
 & \vdots & &
\end{array}
$$

evaluation contexts: $E ::= \bullet \mid E\,e \mid v\,E \mid \nu a.\, E \mid \cdots$
expressions: $e ::= x \mid a \mid \lambda x \to e \mid e\,e \mid \nu a.\, e \mid \cdots$
canonical forms: $v ::= a \mid \lambda x \to e \mid u \mid \cdots$
neutral forms: $u ::= x \mid u\,v \mid \cdots$

References? (N.B. open expressions; and definition of $E/v/u$ in presence of $\mathbf{\Pi}$-, $\mathbf{\Sigma}$- & И-types is subtle.)

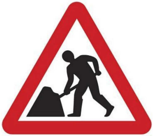
Generative names creep into the pure CTT fragment

Conventional $\mathbf{\Pi}$-elimination:

$$\frac{\Gamma \vdash e_1 : (x : A) \to B \qquad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1\, e_2 : B[e_1/x]}$$

Nu $\mathbf{\Pi}$-elimination:

$$\frac{\Gamma \vdash e_1 : (x : A) \to B \qquad \Gamma \vdash e_2 = \nu \vec{a}.\, v : A}{\Gamma \vdash e_1\, e_2 : \nu \vec{a}.\, B[v/x]}$$



Done:

- Declarative type system with $\boldsymbol{\Sigma}/\boldsymbol{\Pi}/\mathtt{Set} + \boldsymbol{\nu}/\mathtt{swap}/$**И**.

Semi-done:

- Model using nominal sets (specifically, a version of Moggi's dynamic allocation monad on the universe of 'FM-sets' of Gabbay+AMP).

Not done:

- Decidability of type-checking (via algorithmic type system equivalent to the declarative one).
- Inductive types + dependently typed pattern-matching.
- Implementation.