Nominal Sets

Andrew Pitts

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Reading material

- AMP, *Structural Recursion with Locally Scoped Names*, preprint, 2010.
  (Full version of "Nominal System T" POPL 2010 paper.)
- AMP, *Alpha-Structural Recursion and Induction*, Journal of the ACM 53(2006)459-506.

Outline

- The end.

Outline

- The end.
- Motivation: structural recursion for data modulo α-equivalence.
- Introduction to nominal sets.

$$\sim\sim\sim\sim$$

- Nominal restriction sets.
- A simply-typed λ-calculus with name-abstraction types.

Outline

- The end.
- Motivation: structural recursion for data modulo α-equivalence.
- Introduction to nominal sets.

$\sim\sim\sim\sim$

- Nominal restriction sets.
- A simply-typed λ-calculus with name-abstraction types.
- Families of nominal sets as a model of dependent types.

In the end…

```
names Var : Set

data Term : Set where                    --(possibly open) λ-terms mod α
  V : Var -> Term                        --variable
  A : (Term × Term)-> Term               --application term
  L : (Var . Term) -> Term               --λ-abstraction

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

```
names Var : Set

data Term : Set where                        --(possibly open) λ-terms mod α
  V : Var -> Term                            --variable
  A : (Term × Term)-> Term                   --application term
  L : (Var . Term) -> Term                   --λ-abstraction

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

set of atomic names

— not an inductive datatype, but decidable

```
names Var : Set

data Term : Set where                        --(possibly open) λ-terms mod α
  V : Var -> Term                            --variable
  A : (Term × Term)-> Term                   --application term
  L : (Var . Term) -> Term                   --λ-abstraction

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

type of "functions as data" (as opposed to usual computable functions)

(cf. Poswolsky & Schürmann [ESOP 2008],
Licata, Zeilberger & Harper [LICS 2008, ICFP 2009],
Cheney [LFMTP 2008],
Westbrook et al [LFMTP 2009].)

```
names Var : Set

data Term : Set where                          --(possibly open) λ-terms mod α
  V : Var -> Term                              --variable
  A : (Term × Term)-> Term                     --application term
  L : (Var . Term) -> Term                     --λ-abstraction

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

type of "functions as data"

= NAME-ABSTRACTION

à la <u>nominal sets</u>

name-abstraction patterns

```
names Var : Set

data Term : Set where                        --(possibly open) λ-terms mod α
  V : Var -> Term                            --variable
  A : (Term × Term)-> Term                   --application term
  L : (Var . Term) -> Term                   --λ-abstraction

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

achieved by the way name-abstraction patterns match values

cf. FreshML (AMP & Shinwell)
– but matching there is <u>impure</u> – does not sit well with Curry-Howard...

In my dreams: “nominal Agda”

```
names Var : Set

data Term : Set where                          --(possibly open) λ-terms mod α
  V : Var -> Term                              --variable
  A : (Term × Term)-> Term                     --application term
  L : (Var . Term) -> Term                     --λ-abstraction

_/_ : Term -> Var -> Term -> Term              --capture-avoiding substitution
(t / x)(V x′) = if x = x′ then t else V x′
(t / x)(A(t′ , t″)) = A((t / x )t′ , (t / x )t″)
(t / x)(L(x′ . t′)) = L(x′ . (t / x)t′)

data _==_ (t : Term) : Term -> Set where       --intensional equality
  Refl : t == t
```

...want propositions as well as simple types

In my dreams: “nominal Agda”

```
names Var : Set

data Term : Set where                        --(possibly open) λ-terms mod α
  V : Var -> Term                            --variable
  A : (Term × Term)-> Term                   --application term
  L : (Var . Term) -> Term                   --λ-abstraction

_/_ : Term -> Var -> Term -> Term            --capture-avoiding substitution
(t / x)(V x′) = if x = x′ then t else V x′
(t / x)(A(t′ , t″)) = A((t / x )t′ , (t / x )t″)
(t / x)(L(x′ . t′)) = L(x′ . (t / x)t′)

data _==_ (t : Term) : Term -> Set where     --intensional equality
  Refl : t == t                              --is term equality mod α

eg : (x x′ : Var) ->
  ((V x) / x′)(L(x . V x′)) == L(x′ . V x)  --(λx.x′)[x/x′] = λx′.x
eg x x′ = {! !}
```

Structural recursion mod alpha

Structural Recursion: recursive definitions of (total) functions whose values at a *structure* are given functions of their values at *immediate substructures*.

- Gödel (Tate) System T — structure = numbers, structural recursion = primitive recursion for $\mathbb{N}$.
- Burstall, Martin-Löf *et al* generalized this to abstract syntax trees.

Structural recursion

E.g. for λ-trees

$$Tr \triangleq \{t ::= \mathtt{V}\, a \mid \mathtt{A}\, t\, t \mid \mathtt{L}\, a\, t\}$$

($a \in \mathbb{A}$, infinite set of atom[ic name]s)

Structural recursion

E.g. for λ-trees

$$Tr \triangleq \{t ::= \mathtt{V}\, a \mid \mathtt{A}\, t\, t \mid \mathtt{L}\, a\, t\}$$

$$\begin{array}{llll}
\text{Given} & f_1 & \in & \mathbb{A} \to X \\
 & f_2 & \in & X \times X \to X \\
 & f_3 & \in & \mathbb{A} \times X \to X \\
\hline
\text{exists unique} & f & \in & Tr \to X \quad \text{s.t.}
\end{array}$$

$$\begin{aligned}
f\,(\mathtt{V}\, a) &= f_1\, a \\
f\,(\mathtt{A}\, t\, t') &= f_2(f\, t, f\, t') \\
f(\mathtt{L}\, a\, t) &= f_3(a, f\, t)
\end{aligned}$$

Structural recursion

E.g. for λ-trees

$$Tr \triangleq \{t ::= \mathtt{V}\, a \mid \mathtt{A}\, t\, t \mid \mathtt{L}\, a\, t\}$$

$$\begin{array}{lrcl}
\text{Given} & f_1 & \in & \mathbb{A} \to X \\
 & f_2 & \in & X \times X \to X \\
 & f_3 & \in & \mathbb{A} \times X \to X \\
\hline
\text{exists unique} & f & \in & Tr \to X \quad \text{s.t.}
\end{array}$$

$$\begin{aligned}
f\,(\mathtt{V}\, a) &= f_1\, a \\
f\,(\mathtt{A}\, t\, t') &= f_2(f\, t, f\, t') \\
f\,(\mathtt{L}\, a\, t) &= f_3(a, f\, t)
\end{aligned}$$

Not very useful! (because $\mathtt{L}\, a$ is a binder)

λ-terms $t \in \Lambda \;=\; Tr/{\equiv_\alpha}$, λ-trees mod α-equivalence

$a \mapsto \mathtt{V}\,a$	variables
$t, t' \mapsto \mathtt{A}\,t\,t'$	application terms
$a, t \mapsto \mathtt{L}\,a.\,t$	λ-abstraction terms

λ-terms $t \in \Lambda = Tr/{\equiv_\alpha}$, λ-trees mod α-equivalence

$a \mapsto \mathtt{V}\, a$	variables
$t, t' \mapsto \mathtt{A}\, t\, t'$	application terms
$a, t \mapsto \mathtt{L}\, a.\, t$	λ-abstraction terms

$$\mathtt{L}\, a.\, t = \mathtt{L}\, a'.\, t[a'/a] \qquad \text{if } a' \notin fv(a, t)$$

Informal structural recursion

E.g. $f = (-)[t_1/a_1] \in \mathbf{\Lambda} \to \mathbf{\Lambda}$
(capture-avoiding substitution)
is well-(and totally-)defined by:

$$
\begin{array}{rcl}
f\,(\mathtt{V}\,a) & = & \textbf{if } a = a_1 \textbf{ then } t_1 \textbf{ else } \mathtt{V}\,a \\
f\,(\mathtt{A}\,t\,t') & = & \mathtt{A}\,(f\,t)\,(f\,t') \\
f(\mathtt{L}\,a.\,t) & = & \mathtt{L}\,a.\,(f\,t) \quad \text{if } a \notin fv(a_1, t_1)
\end{array}
$$

Informal structural recursion

E.g. $[\![-]\!] \in \Lambda \to (Env \to D)$
(denotation in a suitable domain D)
is well-defined by:

$$
\begin{aligned}
[\![\mathtt{V}\ a]\!] &= \lambda(\rho \in Env) \to \rho\ a \\
[\![\mathtt{A}\ t\ t']\!] &= \lambda(\rho \in Env) \to app([\![t]\!]\rho\,,[\![t']\!]\rho) \\
[\![\mathtt{L}\ a.\,t]\!] &= \lambda(\rho \in Env) \to fun(\lambda(d \in D) \to [\![t]\!](\rho[a \to d]))
\end{aligned}
$$

(where $app \in D \times D \to_{cts} D$ and $fun \in (D \to_{cts} D) \to_{cts} D$ are continuous functions satisfying…)

Informal structural recursion

E.g. $[\![-]\!] \in \Lambda \to (\mathit{Env} \to D)$
(denotation in a suitable domain D)
is well-defined by:

$$
\begin{aligned}
[\![\mathtt{V}\, a]\!] &= \lambda(\rho \in \mathit{Env}) \to \rho\, a \\
[\![\mathtt{A}\, t\, t']\!] &= \lambda(\rho \in \mathit{Env}) \to \mathit{app}([\![t]\!]\rho\,,[\![t']\!]\rho) \\
[\![\mathtt{L}\, a.\, t]\!] &= \lambda(\rho \in \mathit{Env}) \to \mathit{fun}(\lambda(d \in D) \to [\![t]\!](\rho[a \to d]))
\end{aligned}
$$

why is this (very standard) definition
independent of the choice of bound variable a ?

Informal structural recursion

$$\begin{array}{llll}
\text{Given} & f_1 & \in & \mathbb{A} \to X \\
 & f_2 & \in & X \times X \to X \\
 & f_3 & \in & \mathbb{A} \times X \to X \\
\text{and finite} & \overline{a} & \subseteq & \mathbb{A} \\
\text{satisfying...} & & & \\
\hline
\text{exists unique} & f & \in & \Lambda \to X \quad \text{s.t.}
\end{array}$$

$$\begin{array}{rcll}
f\,(\mathtt{V}\,a) & = & f_1\,a & \\
f\,(\mathtt{A}\,t\,t') & = & f_2(f\,t, f\,t') & \\
f(\mathtt{L}\,a.\,t) & = & f_3(a, f\,t) & \text{if } a \notin \overline{a}
\end{array}$$

Informal structural recursion

$$
\begin{array}{llll}
\text{Given} & f_1 & \in & \mathbb{A} \to X \\
 & f_2 & \in & X \times X \to X \\
 & f_3 & \in & \mathbb{A} \times X \to X \\
\text{and finite} & \overline{a} & \subseteq & \mathbb{A} \\
\text{satisfying...} & & & \\
\hline
\text{exists unique} & f & \in & \Lambda \to X \quad \text{s.t.}
\end{array}
$$

$$
\begin{aligned}
f\,(\mathtt{V}\,a) &= f_1\,a \\
f\,(\mathtt{A}\,t\,t') &= f_2(f\,t, f\,t') \\
f(\mathtt{L}\,a.\,t) &= f_3(a, f\,t) \qquad \text{if } a \notin \overline{a}
\end{aligned}
$$

What conditions ensure that f respects α-equivalence and is total?

Informal structural recursion

$$
\begin{array}{llcl}
\text{Given} & f_1 & \in & \mathbb{A} \to X \\
 & f_2 & \in & X \times X \to X \\
 & f_3 & \in & \mathbb{A} \times X \to X \\
\text{and finite} & \bar{a} & \subseteq & \mathbb{A} \\
\text{satisfying...} & & & \\
\hline
\text{exists unique} & f & \in & \Lambda \to X \quad \text{s.t.}
\end{array}
$$

$$
\begin{array}{rcl}
f\,(\mathtt{V}\,a) & = & f_1\,a \\
f\,(\mathtt{A}\,t\,t') & = & f_2(f\,t, f\,t') \\
f(\mathtt{L}\,a.\,t) & = & f_3(a, f\,t) \qquad \text{if } a \notin \bar{a}
\end{array}
$$

$\mathtt{L}\,a'.\,t' =$

$= f_3(a', f\,t')$

Informal structural recursion

$$
\begin{array}{llll}
\text{Given} & f_1 & \in & \mathbb{A} \to \boldsymbol{X} \\
 & f_2 & \in & \boldsymbol{X} \times \boldsymbol{X} \to \boldsymbol{X} \\
 & f_3 & \in & \mathbb{A} \times \boldsymbol{X} \to \boldsymbol{X} \\
\text{and finite} & \overline{a} & \subseteq & \mathbb{A} \\
\text{satisfying\ldots} & & & \\
\hline
\text{exists unique} & f & \in & \boldsymbol{\Lambda} \to \boldsymbol{X} \quad \text{s.t.}
\end{array}
$$

$$
\begin{array}{rcll}
f\,(\mathtt{V}\,a) & = & f_1\,a & \\
f\,(\mathtt{A}\,t\,t') & = & f_2(f\,t, f\,t') & \\
f(\mathtt{L}\,a.\,t) & = & f_3(a, f\,t) & \text{if } a \notin \overline{a}
\end{array}
$$

$a \,\#\, \mathtt{L}\,a.\,t$

$a \,\#\, f_3(a, f\,t)$ (???)

(In)dependence

An important question: what does it mean (abstractly) for a name to occur in a mathematical object?

(In)dependence

An important question: what does it mean (abstractly) for a name to occur in a mathematical object?

Type Theory's answer is functional dependency: make the object a function of names and instantiate the function at a particular name.

(In)dependence

An important question: what does it mean (abstractly) for a name to occur in a mathematical object?

Type Theory's answer is functional dependency: make the object a function of names and instantiate the function at a particular name.

Often a more important question is: what does it mean (abstractly) for a name to not occur in a mathematical object?

Type Theory's answer is. . . ??

(In)dependence

An important question: what does it mean (abstractly) for a name to occur in a mathematical object?

Type Theory's answer is functional dependency: make the object a function of names and instantiate the function at a particular name.

Often a more important question is: what does it mean (abstractly) for a name to not occur in a mathematical object?

Type Theory's answer is. . . ??

Theory of nominal sets provides a nice mathematical notion of "name non-occurrence", called freshness.

Introduction to nominal sets

Preliminaries on name-permutations

- $\mathbb{A}$ = fixed infinite set of (atomic) names (a, b,...)

Preliminaries on name-permutations

- $\mathbb{A}$ = fixed infinite set of (atomic) names (a, b,...)
- $\mathfrak{S}(\mathbb{A})$ = group of finite permutations of $\mathbb{A}$ (π, π',...)
  - π finite means: $\{a \in \mathbb{A} \mid \pi(a) \neq a\}$ is finite.
  - group: multiplication is composition of functions $\pi' \circ \pi$; identity is identity function ι.

Preliminaries on name-permutations

- $\mathbb{A}$ = fixed infinite set of (atomic) names (a, b,...)
- $\mathfrak{S}(\mathbb{A})$ = group of finite permutations of $\mathbb{A}$ (π, π',...)
- action of $\mathfrak{S}(\mathbb{A})$ on a set X is a function $(-)\cdot(-) \in \mathfrak{S}(\mathbb{A}) \times X \to X$ satisfying for all $x \in X$
  - $\pi' \cdot (\pi \cdot x) = (\pi' \circ \pi) \cdot x$
  - $\iota \cdot x = x$

Preliminaries on name-permutations

- $\mathbb{A}$ = fixed infinite set of (atomic) names (a, b,...)
- $\mathfrak{S}(\mathbb{A})$ = group of finite permutations of $\mathbb{A}$ (π, π',...)
- action of $\mathfrak{S}(\mathbb{A})$ on a set X is a function $(-)\cdot(-) \in \mathfrak{S}(\mathbb{A}) \times X \to X$ satisfying for all $x \in X$
  - $\pi' \cdot (\pi \cdot x) = (\pi' \circ \pi) \cdot x$
  - $\iota \cdot x = x$
- swapping: $(a\ b) \in \mathfrak{S}(\mathbb{A})$ is the function mapping a to b, b to a and fixing all other names.

Nominal sets

are sets X with with a $\mathfrak{S}(\mathbb{A})$-action satisfying

Finite support property: for each $x \in X$, there is a finite subset $\overline{a} \subseteq \mathbb{A}$ that supports x:

$$a, a' \notin \overline{a} \;\Rightarrow\; (a\ a') \cdot x = x$$

Fact: in a nominal set every $x \in X$ possesses a *smallest* finite support, written $supp(x)$.

Nominal sets

are sets X with with a $\mathfrak{S}(\mathbb{A})$-action satisfying

Finite support property: for each $x \in X$, there is a finite subset $\overline{a} \subseteq \mathbb{A}$ that supports x:

$$a, a' \notin \overline{a} \;\Rightarrow\; (a\ a') \cdot x = x$$

Fact: in a nominal set every $x \in X$ possesses a *smallest* finite support, written $supp(x)$.

E.g. action $\begin{cases} \pi \cdot (\mathtt{V}\,a) & = \mathtt{V}(\pi(a)) \\ \pi \cdot (\mathtt{A}\,t\,t') & = \mathtt{A}\,(\pi \cdot t)(\pi \cdot t') \\ \pi \cdot (\mathtt{L}\,a.\,t) & = \mathtt{L}\,(\pi(a)).(\pi \cdot t) \end{cases}$

respects α-equivalence of λ-terms and has finite support property: $supp(t) = fv(t)$, free variables of t (exercise!).

Category of nominal sets, $\mathcal{Nom}$

- objects are nominal sets
- morphisms are functions $f \in X \to Y$ that are equivariant:

$$\pi \cdot (f\,x) = f(\pi \cdot x)$$

for all $\pi \in \mathfrak{S}(\mathbb{A})$, $x \in X$.

Category of nominal sets, $\mathcal{N}om$

Fact. $\mathcal{N}om$ is equivalent to the Schanuel topos, a well-known Grothendieck topos classifying the geometric theory of an infinite decidable object.

Category of nominal sets, $\mathcal{N}om$

Fact. $\mathcal{N}om$ is equivalent to the Schanuel topos, a well-known Grothendieck topos classifying the geometric theory of an infinite decidable object.

Finite products: $X_1 \times \cdots \times X_n$ is cartesian product of sets with $\mathfrak{S}(\mathbb{A})$-action

$$\pi \cdot (x_1, \ldots, x_n) = (\pi \cdot x_1, \ldots, \pi \cdot x_n)$$

which satisfies

$$supp((x, \ldots, x_n)) = supp(x_1) \cup \cdots \cup \mathbf{sup}(x_n)$$

Category of nominal sets, $\mathcal{Nom}$

Fact. $\mathcal{Nom}$ is equivalent to the Schanuel topos, a well-known Grothendieck topos classifying the geometric theory of an infinite decidable object.

Exponentials: Y^X is the set of functions $f \in X \to Y$ that are finitely supported w.r.t. the $\mathfrak{S}(\mathbb{A})$-action

$$\pi \cdot f = \lambda(x \in X) \to \pi \cdot (f(\pi^{-1} \cdot x))$$

(Can be tricky to see when $f \in X \to Y$ is in Y^X.)

Category of nominal sets, $\mathcal{N}om$

Fact. $\mathcal{N}om$ is equivalent to the Schanuel topos, a well-known Grothendieck topos classifying the geometric theory of an infinite decidable object.

Subobject classifier: $\Omega = \{0,1\}$ with trivial $\mathfrak{S}(\mathbb{A})$-action: $\pi \cdot b = b$ (so $supp(b) = \emptyset$).

($\mathcal{N}om$ is a Boolean topos: $\Omega = 1 + 1$.)

Natural number object: $\mathbb{N} = \{0,1,2,\ldots\}$ with trivial $\mathfrak{S}(\mathbb{A})$-action: $\pi \cdot n = n$ (so $supp(n) = \emptyset$).

Coproducts are given by disjoint union.

The nominal set of names

$\mathbb{A}$ is a nominal set once equipped with the action

$$\pi \cdot a = \pi(a)$$

which satisfies $supp(a) = \{a\}$.

The nominal set of names

$\mathbb{A}$ is a nominal set once equipped with the action

$$\pi \cdot a = \pi(a)$$

which satisfies $supp(a) = \{a\}$.

$\mathbb{A}$ is not $\mathbb{N}$! Although $\mathbb{A} \in \mathcal{S}et$ is countable, $\mathbb{A} \in \mathcal{N}om$ satisfies

$$\mathcal{N}om \models (\forall f \in \mathbb{A}^{\mathbb{N}})(\exists a \in \mathbb{A})\ a \notin f\,\mathbb{N}$$

The nominal set of names

$\mathbb{A}$ is a nominal set once equipped with the action

$$\pi \cdot a = \pi(a)$$

which satisfies $supp(a) = \{a\}$.

$\mathbb{A}$ is not $\mathbb{N}$! Although $\mathbb{A} \in \mathcal{S}et$ is countable, $\mathbb{A} \in \mathcal{N}om$ satisfies

$$\mathcal{N}om \models (\forall f \in \mathbb{A}^{\mathbb{N}})(\exists a \in \mathbb{A})\ a \notin f\,\mathbb{N}$$

However

$$\mathcal{N}om \not\models (\exists c \in \mathbb{A}^{\mathbb{A}^{\mathbb{N}}})(\forall f \in \mathbb{A}^{\mathbb{N}})\ c(f) \notin f\,\mathbb{N}$$

$\mathcal{N}om$ does not satisfy the Axiom of Choice

Freshness

For each nominal set X, we can define a relation $\# \subseteq \mathbb{A} \times X$ of freshness:

$$a \# x \triangleq a \notin supp(x)$$

Equivalently, $a \# x$ iff $(a\ b) \cdot x = x$ holds for cofinitely many b.

Freshness

For each nominal set X, we can define a relation $\# \subseteq \mathbb{A} \times X$ of freshness:

$$a \# x \triangleq a \notin supp(x)$$

- In $\mathbb{N}$, $a \# n$ always.
- In $\mathbb{A}$, $a \# b$ iff $a \neq b$.
- In Λ, $a \# t$ iff $a \notin fv(t)$.
- In $X \times Y$, $a \# (x, y)$ iff $a \# x$ and $a \# y$.
- In Y^X, $a \# f$ can be subtle!