Nominal Sets and Functional Programming

Andrew Pitts

UNIVERSITY OF CAMBRIDGE

Computer Science & Technology

FoPPS 2019

Nominal sets provide a mathematical theory of *names* based on some simple math to do with properties invariant under permuting names.

Nominal sets provide a mathematical theory of *names* based on some simple math to do with properties invariant under permuting names.

Application area:
computing with / proving properties of
data involving name-binding & scoped local names
in functional programming languages and
theorem-proving systems.

Theory of nominal sets yields principles of structural recursion and induction for syntax modulo renaming of bound names which is close to informal practice and yet fully formal.

Outline

L1 Structural recursion and induction in the presence of name-binding operations. Introducing the category of nominal sets.

L2 Nominal algebraic data types and α-structural recursion.

L3 Dependently typed λ-calculus with locally fresh names and name-abstraction.

References:

AMP, *Alpha-Structural Recursion and Induction*, JACM 53(2006)459-506.

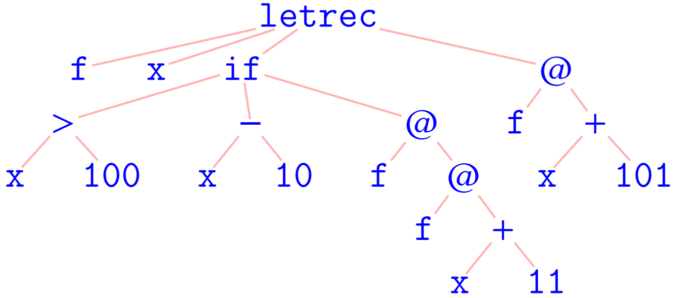
AMP, J. Matthiesen and J. Derikx,
A Dependent Type Theory with Abstractable Names, ENTCS 312(2015)19-50.

Lecture 1

For semantics, concrete syntax

```
letrec f x = if x > 100 then x − 10
else f ( f ( x + 11 ) ) in f ( x + 100 )
```

is unimportant compared to abstract syntax (ASTs):



We should aim for compositional semantics of program constructions, rather than of whole programs.

ASTs enable two fundamental (and inter-linked) tools in programming language semantics:

- Definition of functions on syntax
  by recursion on its structure.
- Proof of properties of syntax
  by induction on its structure.

Structural recursion

Recursive definitions of functions whose values at a *structure* are given functions of their values at *immediate substructures.*

- Gödel System T (1958):

$$\begin{array}{rcl} \text{structure} & = & \text{numbers} \\ \text{structural recursion} & = & \text{primitive recursion for } \mathbb{N}. \end{array}$$

- Burstall, Martin-Löf *et al* (1970s) generalized this to ASTs.

Running example

Set of ASTs for λ-terms

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

where $a \in \mathbb{A}$, fixed infinite set of names of variables.

Operations for constructing these ASTs:

$$\begin{array}{lcl} \mathtt{V} & : & \mathbb{A} \to Tr \\ \mathtt{A} & : & Tr \times Tr \to Tr \\ \mathtt{L} & : & \mathbb{A} \times Tr \to Tr \end{array}$$

Structural recursion for Tr

Theorem.

$$\frac{\text{Given} \quad \begin{array}{rcl} f_1 & \in & \mathbb{A} \to X \\ f_2 & \in & X \times X \to X \\ f_3 & \in & \mathbb{A} \times X \to X \end{array}}{\text{exists unique} \quad \hat{f} \in Tr \to X \quad \text{satisfying}}$$

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

Structural recursion for Tr

E.g. the finite set $\mathrm{var}\, t$ of variables occurring in $t \in Tr$:

$$\begin{array}{rcl} \mathrm{var}(\mathtt{V}\, a) & = & \{a\} \\ \mathrm{var}(\mathtt{A}(t, t')) & = & (\mathrm{var}\, t) \cup (\mathrm{var}\, t') \\ \mathrm{var}(\mathtt{L}(a, t)) & = & (\mathrm{var}\, t) \cup \{a\} \end{array}$$

is defined by structural recursion using

- $X = \mathrm{P_f}(\mathbb{A})$ (finite sets of variables)
- $f_1\, a = \{a\}$
- $f_2(S, S') = S \cup S'$
- $f_3(a, S) = S \cup \{a\}$.

Structural recursion for Tr

E.g. swapping: $(a\ b) \cdot t$ = result of transposing all occurrences of a and b in t

For example

$$(a\ b) \cdot \mathtt{L}(a, \mathtt{A}(\mathtt{V}\,b, \mathtt{V}\,c)) = \mathtt{L}(b, \mathtt{A}(\mathtt{V}\,a, \mathtt{V}\,c))$$

Structural recursion for Tr

E.g. swapping: $(a\ b) \cdot t$ = result of transposing all occurrences of a and b in t

$$
\begin{array}{rcl}
(a\ b) \cdot \mathtt{V}\, c & = & \text{if } c = a \text{ then } \mathtt{V}\, b \text{ else} \\
 & & \text{if } c = b \text{ then } \mathtt{V}\, a \text{ else } \mathtt{V}\, c \\
(a\ b) \cdot \mathtt{A}(t, t') & = & \mathtt{A}((a\ b) \cdot t, (a\ b) \cdot t') \\
(a\ b) \cdot \mathtt{L}(c, t) & = & \text{if } c = a \text{ then } \mathtt{L}(b, (a\ b) \cdot t) \\
 & & \text{ else if } c = b \text{ then } \mathtt{L}(a, (a\ b) \cdot t) \\
 & & \text{ else } \mathtt{L}(c, (a\ b) \cdot t)
\end{array}
$$

is defined by structural recursion using...

Structural recursion for Tr

Theorem.

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

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

Structural recursion for Tr

Theorem.

Given $f_1 \in \mathbb{A} \to X$

$f_2 \in X \times$ …

$f_3 \in$ …

exists unique … $\to X$ satisfying

$$\ldots = f_1\, a$$

$$\ldots(t, t')) = f_2(\hat{f}\, t, \hat{f}\, t')$$

$$\hat{f}(\mathtt{L}(a, t)) = f_3(a, \hat{f}\, t)$$

Doesn't take binding into account!

Alpha-equivalence

Smallest binary relation $=_\alpha$ on Tr closed under the rules:

$$\frac{a \in \mathbb{A}}{\mathtt{V}\,a =_\alpha \mathtt{V}\,a} \qquad \frac{t_1 =_\alpha t_1' \qquad t_2 =_\alpha t_2'}{\mathtt{A}(t_1, t_2) =_\alpha \mathtt{A}(t_1', t_2')}$$

$$\frac{(a\ b) \cdot t =_\alpha (a'\ b) \cdot t' \qquad b \notin \{a, a'\} \cup \mathrm{var}(t) \cup \mathrm{var}(t')}{\mathtt{L}(a, t) =_\alpha \mathtt{L}(a', t')}$$

E.g.

$$\begin{array}{lll} \mathtt{A}(\mathtt{L}(a, \mathtt{A}(\mathtt{V}\,a, \mathtt{V}\,b)), \mathtt{V}\,c) & =_\alpha & \mathtt{A}(\mathtt{L}(c, \mathtt{A}(\mathtt{V}\,c, \mathtt{V}\,b)), \mathtt{V}\,c) \\ & \neq_\alpha & \mathtt{A}(\mathtt{L}(b, \mathtt{A}(\mathtt{V}\,b, \mathtt{V}\,b)), \mathtt{V}\,c) \end{array}$$

Fact: $=_\alpha$ is transitive (and reflexive & symmetric). [Ex. 1]

ASTs mod alpha equivalence

Dealing with issues to do with binders and alpha equivalence is

- pervasive (very many languages involve binding operations)
- difficult to formalise/mechanise without losing sight of common informal practice:

ASTs mod alpha equivalence

Dealing with issues to do with binders and alpha equivalence is

- pervasive (very many languages involve binding operations)
- difficult to formalise/mechanise without losing sight of common informal practice:

> "We identify expressions up to alpha-equivalence"...

ASTs mod alpha equivalence

Dealing with issues to do with binders and alpha equivalence is

- pervasive (very many languages involve binding operations)
- difficult to formalise/mechanise without losing sight of common informal practice:

> "We identify expressions up to alpha-equivalence"…
> …and then forget about it, referring to alpha-equivalence classes $[t]_\alpha$ only via representatives t.

ASTs mod alpha equivalence

Dealing with issues to do with binders and alpha equivalence is

- pervasive (very many languages involve binding operations)
- difficult to formalise/mechanise without losing sight of common informal practice:

E.g. notation for λ-terms:

$$\Lambda \triangleq \{[t]_\alpha \mid t \in Tr\}$$

$$\begin{array}{rll} a & \text{means} & [\mathtt{V}\ a]_\alpha\ (\, = \{\mathtt{V}\ a\}) \\ e\ e' & \text{means} & [\mathtt{A}(t, t')]_\alpha, \text{ where } e = [t]_\alpha \text{ and } e' = [t']_\alpha \\ \lambda a.e & \text{means} & [\mathtt{L}(a, t)]_\alpha \text{ where } e = [t]_\alpha \end{array}$$

Informal structural recursion

E.g. capture-avoiding substitution:

$$f = (-)[e_1/a_1] : \Lambda \to \Lambda$$

$$\begin{aligned}
f\,a &= \text{if } a = a_1 \text{ then } e_1 \text{ else } a \\
f\,(e\,e') &= (f\,e)\,(f\,e') \\
f(\lambda a.\,e) &= \text{if } a \notin \mathrm{var}(a_1, e_1) \text{ then } \lambda a.\,(f\,e) \\
&\qquad \text{else don't care!}
\end{aligned}$$

Not an instance of structural recursion for Tr.

Why is f well-defined and total?

Informal structural recursion

E.g. denotation of λ-term in a suitable domain D:

$$[\![-]\!] : \Lambda \to ((\mathbb{A} \to D) \to D)$$

$$\begin{array}{rcl} [\![a]\!]\rho & = & \rho\, a \\ [\![e\, e']\!]\rho & = & app([\![e]\!]\rho\,,\, [\![e']\!]\rho) \\ [\![\lambda a.\, e]\!]\rho & = & fun(\lambda(d \in D) \to [\![e]\!](\rho[a \to d])) \end{array}$$

$$\text{where} \begin{cases} app & \in & D \times D \to_{cts} D \\ fun & \in & (D \to_{cts} D) \to_{cts} D \end{cases}$$

are continuous functions satisfying…

Informal structural recursion

E.g. denotation of λ-term in a suitable domain D:

$$[\![-]\!] : \Lambda \to ((\mathbb{A} \to D) \to D)$$

$$\begin{aligned}
[\![a]\!]\rho &= \rho\, a \\
[\![e\, e']\!]\rho &= app([\![e]\!]\rho\, , [\![e']\!]\rho) \\
[\![\lambda a.\, e]\!]\rho &= fun(\lambda(d \in D) \to [\![e]\!](\rho[a \to d]))
\end{aligned}$$

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

$\rho[a \to d]$ is the element of $\mathbb{A} \to D$ that maps a to d and otherwise acts like ρ

Is there a recursion principle for Λ that legitimises these 'definitions' of $(-)[e_1/a_1] : \Lambda \to \Lambda$ and $[\![-]\!] : \Lambda \to D$ (and many other e.g.s)?

Is there a recursion principle for Λ that legitimises these 'definitions' of $(-)[e_1/a_1] : \Lambda \to \Lambda$ and $[\![-]\!] : \Lambda \to D$ (and many other e.g.s)?

Yes! — α-structural recursion.

Is there a recursion principle for Λ that legitimises these 'definitions' of $(-)[e_1/a_1] : \Lambda \to \Lambda$ and $[\![-]\!] : \Lambda \to D$ (and many other e.g.s)?

Yes! — α-structural recursion.

What about other languages with binders?

Is there a recursion principle for Λ that legitimises these 'definitions' of $(-)[e_1/a_1] : \Lambda \to \Lambda$ and $[\![-]\!] : \Lambda \to D$ (and many other e.g.s)?

Yes! — α-structural recursion.

What about other languages with binders?

Yes! — available for any nominal signature.

Is there a recursion principle for Λ that legitimises these 'definitions' of $(-)[e_1/a_1] : \Lambda \to \Lambda$ and $[\![-]\!] : \Lambda \to D$ (and many other e.g.s)?

Yes! — α-structural recursion.

What about other languages with binders?

Yes! — available for any nominal signature.

Great. What's the catch?

Is there a recursion principle for Λ that legitimises these 'definitions' of $(-)[e_1/a_1] : \Lambda \to \Lambda$ and $[\![-]\!] : \Lambda \to D$ (and many other e.g.s)?

Yes! — α-structural recursion.

What about other languages with binders?

Yes! — available for any nominal signature.

Great. What's the catch?

Need to learn a bit of possibly unfamiliar math, to do with permutations and support.

Preliminaries on name-permutations

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

Preliminaries on name-permutations

- $\mathbb{A}$ = fixed countably infinite set of names (a,b,...)
- $\mathrm{Perm}\,\mathbb{A}$ = 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 countably infinite set of names (a,b,...)
- $\mathrm{Perm}\,\mathbb{A}$ = 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.
- swapping: $(a\ b) \in \mathrm{Perm}\,\mathbb{A}$ is the function mapping a to b, b to a and fixing all other names.

Fact: every $\pi \in \mathrm{Perm}\,\mathbb{A}$ is equal to

$$(a_1\ b_1) \circ \cdots \circ (a_n\ b_n)$$

for some a_i & b_i (with $\pi\, a_i \neq a_i \neq b_i \neq \pi\, b_i$).

Preliminaries on name-permutations

- $\mathbb{A}$ = fixed countably infinite set of names (a,b,...)
- $\mathrm{Perm}\,\mathbb{A}$ = group of finite permutations of $\mathbb{A}$ (π, π',...)
- action of $\mathrm{Perm}\,\mathbb{A}$ on a set X is a function

$$(-)\cdot(-) : \mathrm{Perm}\,\mathbb{A} \times X \to X$$

  satisfying for all $x \in X$
  - $\pi' \cdot (\pi \cdot x) = (\pi' \circ \pi) \cdot x$
  - $\iota \cdot x = x$

Running example

Action of $\mathrm{Perm}\,\mathbb{A}$ on set of ASTs for λ-terms

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

$$\begin{array}{rcl} \pi \cdot \mathtt{V}\,a & = & \mathtt{V}(\pi\,a) \\ \pi \cdot \mathtt{A}(t,t') & = & \mathtt{A}(\pi \cdot t, \pi \cdot t') \\ \pi \cdot \mathtt{L}(a,t) & = & \mathtt{L}(\pi\,a, \pi \cdot t) \end{array}$$

This respects α-equivalence and so induces an action on set of λ-terms $\Lambda = \{[t]_\alpha \mid t \in Tr\}$:

$$\pi \cdot [t]_\alpha = [\pi \cdot t]_\alpha$$

Nominal sets

are sets X with with a $\mathrm{Perm}\,\mathbb{A}$-action satisfying

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

$$((\forall a \in \overline{a})\ \pi\, a = a) \implies \pi \cdot x = x$$

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

(Swan: this **Fact** relies on a (weak form of) the Law of Excluded Middle in classical logic; see `arXiv:1702.01556`.)

Nominal sets

are sets X with with a $\mathrm{Perm}\,\mathbb{A}$-action satisfying

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

$$((\forall a \in \overline{a})\ \pi\ a = a) \implies \pi \cdot x = x$$

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

E.g. Tr and Λ are nominal sets—any $\overline{a}$ containing all the variables occurring (free, binding, or bound) in $t \in Tr$ supports t and (hence) $[t]_\alpha$.

Fact: for $e \in \Lambda$, $\mathrm{supp}\,e$ = set of *free variables* of e. [Ex. 2]

Further examples of support

[$\mathrm{Perm}\,\mathbb{A}$ acts of sets of names $S \subseteq \mathbb{A}$ pointwise:
$\pi \cdot S \triangleq \{\pi\, a \mid a \in S\}$.]

What is a support for the following sets of names?

- $S_1 \triangleq \{a\}$

- $S_2 \triangleq \mathbb{A} - \{a\}$

- $S_3 \triangleq \{a_0, a_2, a_4, \ldots\}$, where $\mathbb{A} = \{a_0, a_1, a_2, \ldots\}$

Further examples of support

[$\mathrm{Perm}\,\mathbb{A}$ acts of sets of names $S \subseteq \mathbb{A}$ pointwise:
$\pi \cdot S \triangleq \{\pi\, a \mid a \in S\}$.]

What is a support for the following sets of names?

- $S_1 \triangleq \{a\}$
  Answer: $\{a\}$ is smallest support.
- $S_2 \triangleq \mathbb{A} - \{a\}$
- $S_3 \triangleq \{a_0, a_2, a_4, \ldots\}$, where $\mathbb{A} = \{a_0, a_1, a_2, \ldots\}$

Further examples of support

[$\mathrm{Perm}\,\mathbb{A}$ acts of sets of names $S \subseteq \mathbb{A}$ pointwise:
$\pi \cdot S \triangleq \{\pi\, a \mid a \in S\}$.]

What is a support for the following sets of names?

- $S_1 \triangleq \{a\}$
  Answer: $\{a\}$ is smallest support.
- $S_2 \triangleq \mathbb{A} - \{a\}$
  Answer: $\{a\}$ is smallest support.
- $S_3 \triangleq \{a_0, a_2, a_4, \ldots\}$, where $\mathbb{A} = \{a_0, a_1, a_2, \ldots\}$

Further examples of support

[$\mathrm{Perm}\,\mathbb{A}$ acts of sets of names $S \subseteq \mathbb{A}$ pointwise:
$\pi \cdot S \triangleq \{\pi\, a \mid a \in S\}$.]

What is a support for the following sets of names?

- $S_1 \triangleq \{a\}$
  Answer: $\{a\}$ is smallest support.
- $S_2 \triangleq \mathbb{A} - \{a\}$
  Answer: $\{a\}$ is smallest support.
- $S_3 \triangleq \{a_0, a_2, a_4, \ldots\}$, where $\mathbb{A} = \{a_0, a_1, a_2, \ldots\}$
  Answer: $\{a_0, a_2, a_4, \ldots\}$ is a support

Further examples of support

[$\mathrm{Perm}\,\mathbb{A}$ acts of sets of names $S \subseteq \mathbb{A}$ pointwise:
$\pi \cdot S \triangleq \{\pi\, a \mid a \in S\}$.]

What is a support for the following sets of names?

- $S_1 \triangleq \{a\}$
  Answer: $\{a\}$ is smallest support.
- $S_2 \triangleq \mathbb{A} - \{a\}$
  Answer: $\{a\}$ is smallest support.
- $S_3 \triangleq \{a_0, a_2, a_4, \ldots\}$, where $\mathbb{A} = \{a_0, a_1, a_2, \ldots\}$
  Answer: $\{a_0, a_2, a_4, \ldots\}$ is a support, and so is $\{a_1, a_3, a_5, \ldots\}$—but there is no finite support. S_3 does not exist in the 'world of nominal sets'—in that world $\mathbb{A}$ is infinite, but not enumerable.

Category of nominal sets, **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 $\pi \in \mathrm{Perm}\,\mathbb{A}$, $x \in X$.

Category of nominal sets, **Nom**

Fact. **Nom** is equivalent to the Schanuel topos, a well-known Grothendieck topos classifying the geometric theory of an infinite decidable object.

So in particular **Nom** is a model of Church's classical higher-order logic.

Category of nominal sets, **Nom**

Fact. **Nom** 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 Perm $\mathbb{A}$-action

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

which satisfies

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

[Ex. 3]

Category of nominal sets, Nom

Fact. **Nom** is equivalent to the Schanuel topos, a well-known Grothendieck topos classifying the geometric theory of an infinite decidable object.

Coproducts are given by disjoint union. [Ex. 7]

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

Category of nominal sets, **Nom**

Fact. **Nom** is equivalent to the Schanuel topos, a well-known Grothendieck topos classifying the geometric theory of an infinite decidable object.

Exponentials: $X \to_{\mathrm{fs}} Y$ is the set of functions $f \in Y^X$ that are finitely supported w.r.t. the $\mathrm{Perm}\,\mathbb{A}$-action

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

[Ex. 5]

(Can be tricky to see when $f \in Y^X$ is in $X \to_{\mathrm{fs}} Y$.)

Category of nominal sets, **Nom**

Fact. **Nom** is equivalent to the Schanuel topos, a well-known Grothendieck topos classifying the geometric theory of an infinite decidable object.

Subobject classifier: $\Omega = \{\mathrm{true}, \mathrm{false}\}$ with trivial $\mathrm{Perm}\,\mathbb{A}$-action: $\pi \cdot b \triangleq b$ (so $\mathrm{supp}\, b = \emptyset$).

(**Nom** is a Boolean topos: $\Omega = 1 + 1$.)

Power objects: $X \to_{\mathrm{fs}} \Omega \cong \mathrm{P}_{\mathrm{fs}}\, X$, the set of subsets $S \subseteq X$ that are finitely supported w.r.t. the $\mathrm{Perm}\,\mathbb{A}$-action

$$\pi \cdot S \triangleq \{\pi \cdot x \mid x \in S\}$$

The nominal set of names

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

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

which satisfies $\operatorname{supp} a = \{a\}$.

N.B. $\mathbb{A}$ is not $\mathbb{N}$! Although $\mathbb{A} \in \mathbf{Set}$ is a countable, any $f \in \mathbb{N} \to_{\mathrm{fs}} \mathbb{A}$ has to satisfy

$$\{f\,n\} = \operatorname{supp}(f\,n) \subseteq \operatorname{supp} f \cup \operatorname{supp} n = \operatorname{supp} f$$

for all $n \in \mathbb{N}$, and so f cannot be surjective.

Nom $\not\models$ choice

Nom models classical higher-order logic, but not Hilbert's ε-operation $\varepsilon x.\,\varphi(x)$, which satisfies

$$(\forall x : X)\ \varphi(x) \Rightarrow \varphi(\varepsilon x.\,\varphi(x))$$

Theorem. There is no equivariant function $c : \{S \in \mathrm{P}_{\mathrm{fs}}\,\mathbb{A} \mid S \neq \emptyset\} \longrightarrow \mathbb{A}$ satsifying $c(S) \in S$ for all non-empty $S \in \mathrm{P}_{\mathrm{fs}}\,\mathbb{A}$.

Proof. Suppose there were such a c. Putting $a \triangleq c\,\mathbb{A}$ and picking some $b \in \mathbb{A} - \{a\}$, we get a contradiction to $a \neq b$:

$$a = c\,\mathbb{A} = c((a\ b)\cdot\mathbb{A}) = (a\ b)\cdot c\,\mathbb{A} = (a\ b)\cdot a = b$$

Nom $\not\models$ choice

Nom models classical higher-order logic, but not Hilbert's ε-operation $\varepsilon x.\, \varphi(x)$, which satisfies

$$(\forall x : X)\ \varphi(x) \Rightarrow \varphi(\varepsilon x.\, \varphi(x))$$

In fact **Nom** does not model even very weak forms of choice, such as Dependent Choice.

Freshness

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

$$a \# x \;\triangleq\; a \notin \operatorname{supp} x$$

Freshness

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

$$a \# x \;\triangleq\; a \notin \operatorname{supp} x$$

- In $\mathbb{N}$, $a \# n$ always.
- In $\mathbb{A}$, $a \# b$ iff $a \neq b$.
- In Λ, $a \# t$ iff $a \notin \operatorname{fv} t$.
- In $X \times Y$, $a \# (x, y)$ iff $a \# x$ and $a \# y$.
- In $X \to_{\mathrm{fs}} Y$, $a \# f$ can be subtle!
  (and hence ditto for $\mathrm{P}_{\mathrm{fs}} X$)

Lecture 2

Outline

L1 Structural recursion and induction in the presence of name-binding operations. Introducing the category of nominal sets.

L2 Nominal algebraic data types and α-structural recursion.

L3 Dependently typed λ-calculus with locally fresh names and name-abstraction.

References:

AMP, *Alpha-Structural Recursion and Induction*, JACM 53(2006)459-506.

AMP, J. Matthiesen and J. Derikx,
A Dependent Type Theory with Abstractable Names, ENTCS 312(2015)19-50.

Recall: Alpha-equivalence

Smallest binary relation $=_\alpha$ on Tr closed under the rules:

$$\frac{a \in \mathbb{A}}{\mathtt{V}\,a =_\alpha \mathtt{V}\,a} \qquad \frac{t_1 =_\alpha t_1' \qquad t_2 =_\alpha t_2'}{\mathtt{A}(t_1, t_2) =_\alpha \mathtt{A}(t_1', t_2')}$$

$$\frac{(a\ b)\cdot t =_\alpha (a'\ b)\cdot t' \qquad b \notin \{a, a'\} \cup \mathrm{var}(t) \cup \mathrm{var}(t')}{\mathtt{L}(a, t) =_\alpha \mathtt{L}(a', t')}$$

E.g.

$$\begin{aligned} \mathtt{A}(\mathtt{L}(a, \mathtt{A}(\mathtt{V}\,a, \mathtt{V}\,b)), \mathtt{V}\,c) &=_\alpha \mathtt{A}(\mathtt{L}(c, \mathtt{A}(\mathtt{V}\,c, \mathtt{V}\,b)), \mathtt{V}\,c) \\ &\neq_\alpha \mathtt{A}(\mathtt{L}(b, \mathtt{A}(\mathtt{V}\,b, \mathtt{V}\,b)), \mathtt{V}\,c) \end{aligned}$$

Fact: $=_\alpha$ is transitive (and reflexive & symmetric). [Ex. 1]

Name abstraction

Each $X \in \mathbf{Nom}$ yields a nominal set $\boxed{[\mathbb{A}]X}$ of

name-abstractions $\langle a \rangle x$ are $\sim$-equivalence classes of pairs $(a, x) \in \mathbb{A} \times X$, where

$$
\begin{aligned}
(a, x) \sim (a', x') \;\Leftrightarrow\; & \exists b \mathrel{\#} (a, x, a', x') \\
& (b\ a) \cdot x = (b\ a') \cdot x'
\end{aligned}
$$

The $\mathrm{Perm}\,\mathbb{A}$-action on $[\mathbb{A}]X$ is well-defined by

$$\pi \cdot \langle a \rangle x = \langle \pi(a) \rangle (\pi \cdot x)$$

Fact: $\mathrm{supp}(\langle a \rangle x) = \mathrm{supp}\, x - \{a\}$, so that

$$b \mathrel{\#} \langle a \rangle x \;\Leftrightarrow\; b = a \;\vee\; b \mathrel{\#} x$$

Name abstraction

Each $X \in \mathbf{Nom}$ yields a nominal set $\boxed{[\mathbb{A}]X}$ of

name-abstractions $\langle a \rangle x$ are $\sim$-equivalence classes of pairs $(a, x) \in \mathbb{A} \times X$, where

$$\begin{aligned}(a, x) \sim (a', x') \;\Leftrightarrow\; & \exists b \,\#\, (a, x, a', x') \\ & (b\ a) \cdot x = (b\ a') \cdot x'\end{aligned}$$

We get a functor $[\mathbb{A}](-) : \mathbf{Nom} \to \mathbf{Nom}$ sending $f \in \mathbf{Nom}(X, Y)$ to $[\mathbb{A}]f \in \mathbf{Nom}([\mathbb{A}]X, [\mathbb{A}]Y)$ where

$$[\mathbb{A}]f\,(\langle a \rangle x) = \langle a \rangle (f\ x)$$

Name abstraction

$[\mathbb{A}](-) : \mathbf{Nom} \to \mathbf{Nom}$ is a kind of (affine) function space—it is right adjoint to the functor $\mathbb{A} \otimes (-) : \mathbf{Nom} \to \mathbf{Nom}$ sending X to $\mathbb{A} \otimes X = \{(a, x) \mid a \# x\}$.

Co-unit of the adjunction is 'concretion' of an abstraction

$$\_ @ \_ : ([\mathbb{A}]X) \otimes \mathbb{A} \longrightarrow X$$

defined by computation rule:

$$(\langle a \rangle x) @ b = (b\ a) \cdot x, \text{ if } b \# \langle a \rangle x$$

[Ex. 6]

Name abstraction

Generalising concretion, we have the following characterization of morphisms out of $[\mathbb{A}]X$

Theorem. $f \in (\mathbb{A} \times X) \to_{\mathrm{fs}} Y$ factors through the subquotient $\mathbb{A} \times X \supseteq \{(a, x) \mid a \# f\} \twoheadrightarrow [\mathbb{A}]X$ to give a unique element of $\overline{f} \in ([\mathbb{A}]X) \to_{\mathrm{fs}} Y$ satisfying

$$\overline{f}(\langle a \rangle x) = f(a, x) \quad \text{if } a \# f$$

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

iff $(\exists a \in \mathbb{A})\ a \# f \ \wedge\ (\forall x \in X)\ a \# f(a, x)$.

Initial algebras

- $[\mathbb{A}](-)$ has excellent exactness properties. It can be combined with $\times$, $+$ and $X \to_{\mathrm{fs}} (-)$ to give functors $\mathrm{T} : \mathbf{Nom} \to \mathbf{Nom}$ that have initial algebras $I : \mathrm{T}\,D \to D$

$$\begin{tikzcd} \mathrm{T}\,D \arrow[d, "I"'] & \mathrm{T}\,X \arrow[d, "F", "\text{for all}"'] \\ D & X \end{tikzcd}$$

Initial algebras

- $[\mathbb{A}](-)$ has excellent exactness properties. It can be combined with $\times$, $+$ and $X \to_{\mathrm{fs}} (-)$ to give functors $\mathrm{T} : \mathbf{Nom} \to \mathbf{Nom}$ that have initial algebras $I : \mathrm{T}\, D \to D$

$$\begin{tikzcd}
\mathrm{T}\, D \arrow[r, dashed, "\mathrm{T}\, \hat{F}"] \arrow[d, "I"'] & \mathrm{T}\, X \arrow[d, "F"] \\
D \arrow[r, dashed, "\text{exists unique}", "\hat{F}"'] & X
\end{tikzcd}$$

Initial algebras

- $[\mathbb{A}](-)$ has excellent exactness properties. It can be combined with $\times$, $+$ and $X \to_{\mathrm{fs}} (-)$ to give functors $\mathrm{T} : \mathbf{Nom} \to \mathbf{Nom}$ that have initial algebras $I : \mathrm{T}\,D \to D$
- For a wide class of such functors (nominal algebraic functors) the initial algebra D coincides with ASTs/α-equivalence.
  E.g. Λ is the initial algebra for

$$\mathrm{T}(-) \triangleq \mathbb{A} + (- \times -) + [\mathbb{A}](-)$$

Nominal algebraic signatures

- Sorts $\mathtt{S} \ ::= \ \mathtt{N}$ name-sort (here just one, for simplicity)
  $\mid \ \mathtt{D}$ data-sorts
  $\mid \ \mathtt{1}$ unit
  $\mid \ \mathtt{S}\,,\,\mathtt{S}$ pairs
  $\mid \ \mathtt{N}\,.\,\mathtt{S}$ name-binding
- Typed operations $\mathtt{op} : \mathtt{S} \to \mathtt{D}$

Signature Σ is specified by the stuff in red.

Nominal algebraic signatures

Example: λ-calculus

name-sort `Var` for variables, data-sort `Term` for terms, and operations

$$
\begin{aligned}
&\mathtt{V} : \mathtt{Var} \longrightarrow \mathtt{Term} \\
&\mathtt{A} : \mathtt{Term}\,,\mathtt{Term} \longrightarrow \mathtt{Term} \\
&\mathtt{L} : \mathtt{Var}\,.\,\mathtt{Term} \longrightarrow \mathtt{Term}
\end{aligned}
$$

Nominal algebraic signatures

Example: π-calculus

name-sort $\mathtt{Chan}$ for channel names, data-sorts $\mathtt{Proc}$, $\mathtt{Pre}$ and $\mathtt{Sum}$ for processes, prefixed processes and summations, and operations

$$
\begin{aligned}
\mathtt{S} &: \mathtt{Sum} \longrightarrow \mathtt{Proc} \\
\mathtt{Comp} &: \mathtt{Proc}\,,\mathtt{Proc} \longrightarrow \mathtt{Proc} \\
\mathtt{Nu} &: \mathtt{Chan}\,.\,\mathtt{Proc} \longrightarrow \mathtt{Proc} \\
\mathtt{!} &: \mathtt{Proc} \longrightarrow \mathtt{Proc} \\
\mathtt{P} &: \mathtt{Pre} \longrightarrow \mathtt{Sum} \\
\mathtt{0} &: 1 \longrightarrow \mathtt{Sum} \\
\mathtt{Plus} &: \mathtt{Sum}\,,\mathtt{Sum} \longrightarrow \mathtt{Sum} \\
\mathtt{Out} &: \mathtt{Chan}\,,\mathtt{Chan}\,,\mathtt{Proc} \longrightarrow \mathtt{Pre} \\
\mathtt{In} &: \mathtt{Chan}\,,(\mathtt{Chan}\,.\,\mathtt{Proc}) \longrightarrow \mathtt{Pre} \\
\mathtt{Tau} &: \mathtt{Proc} \longrightarrow \mathtt{Pre} \\
\mathtt{Match} &: \mathtt{Chan}\,,\mathtt{Chan}\,,\mathtt{Pre} \longrightarrow \mathtt{Pre}
\end{aligned}
$$

Nominal algebraic signatures

Closely related notions:

- *binding signatures* of Fiore, Plotkin & Turi (LICS 1999)
- *nominal algebras* of Honsell, Miculan & Scagnetto (ICALP 2001)

N.B. all these notions of signature restrict attention to iterated, but *unary* name-binding—there are other kinds of lexically scoped binder (e.g. see Pottier's Cαml language, or Blanchette *et al* POPL 2019.)

$\Sigma(\mathtt{S})$ = raw terms over Σ of sort $\mathtt{S}$

$$\frac{a \in \mathbb{A}}{a \in \Sigma(\mathtt{N})} \qquad \frac{t \in \Sigma(\mathtt{S}) \qquad \mathtt{op} : \mathtt{S} \longrightarrow \mathtt{D}}{\mathtt{op}\, t \in \Sigma(\mathtt{D})} \qquad \frac{}{() \in \Sigma(\mathtt{1})}$$

$$\frac{t_1 \in \Sigma(\mathtt{S}_1) \qquad t_2 \in \Sigma(\mathtt{S}_2)}{t_1\,,\,t_2 \in \Sigma(\mathtt{S}_1\,,\,\mathtt{S}_2)} \qquad \frac{a \in \mathbb{A} \qquad t \in \Sigma(\mathtt{S})}{a\,.\,t \in \Sigma(\mathtt{N}\,.\,\mathtt{S})}$$

Each $\Sigma(\mathtt{S})$ is a nominal set once equipped with the obvious $\mathrm{Perm}\,\mathbb{A}$-action—any finite set of atoms containing all those occurring in t supports $t \in \Sigma(\mathtt{S})$.

Alpha-equivalence $=_\alpha \subseteq \Sigma(\mathsf{S}) \times \Sigma(\mathsf{S})$

$$\frac{a \in \mathbb{A}}{a =_\alpha a} \qquad \frac{t =_\alpha t'}{\mathsf{op}\, t =_\alpha \mathsf{op}\, t'} \qquad \frac{}{() =_\alpha ()}$$

$$\frac{t_1 =_\alpha t_1' \qquad t_2 =_\alpha t_2'}{t_1 \,,\, t_2 =_\alpha t_1' \,,\, t_2'}$$

$$\frac{(a_1\ a) \cdot t_1 =_\alpha (a_2\ a) \cdot t_2 \qquad a \,\#\, (a_1, t_1, a_2, t_2)}{a_1 \,.\, t_1 =_\alpha a_2 \,.\, t_2}$$

Alpha-equivalence $=_\alpha \subseteq \Sigma(\mathrm{S}) \times \Sigma(\mathrm{S})$

Fact: $=_\alpha$ is equivariant ($t_1 =_\alpha t_2 \implies \pi \cdot t_1 =_\alpha \pi \cdot t_2$) and each quotient

$$\Sigma_\alpha(\mathrm{S}) \triangleq \{[t]_\alpha \mid t \in \Sigma(\mathrm{S})\}$$

is a nominal set with

$$\begin{array}{rcl} \pi \cdot [t]_\alpha & = & [\pi \cdot t]_\alpha \\ \mathrm{supp}\,[t]_\alpha & = & \mathit{fn}\, t \\ & \text{where} & \\ \mathit{fn}(a\,.\,t) & = & \mathit{fn}\, t - \{a\} \\ \mathit{fn}(t_1\,,\,t_2) & = & \mathit{fn}\, t_1 \cup \mathit{fn}\, t_2 \\ & \text{etc.} & \end{array}$$

Theorem. Given a nominal algebraic signature Σ
(for simplicity, assume Σ has a single data-sort $\mathtt{D}$ as well as a single name-sort $\mathtt{N}$)
$\Sigma_\alpha(\mathtt{D})$ is an initial algebra for the associated functor $T_\Sigma : \mathbf{Nom} \to \mathbf{Nom}$.

Theorem. Given a nominal algebraic signature Σ (for simplicity, assume Σ has a single data-sort $\mathtt{D}$ as well as a single name-sort $\mathtt{N}$)
$\Sigma_\alpha(\mathtt{D})$ is an initial algebra for the associated functor $T_\Sigma : \mathbf{Nom} \longrightarrow \mathbf{Nom}$.

$$T_\Sigma(-) = [\![\mathtt{S}_1]\!](-) + \cdots + [\![\mathtt{S}_n]\!](-)$$

where Σ has operations $\mathtt{op}_i : \mathtt{S}_i \longrightarrow D$ $(i = 1..n)$

and $[\![\mathtt{S}]\!](-) : \mathbf{Nom} \longrightarrow \mathbf{Nom}$ is defined by:

$$\begin{array}{rcl}
[\![\mathtt{N}]\!](-) & = & \mathbb{A} \\
[\![\mathtt{D}]\!](-) & = & (-) \\
[\![\mathtt{1}]\!](-) & = & 1 \\
[\![\mathtt{S}_1\,,\,\mathtt{S}_2]\!](-) & = & [\![\mathtt{S}_1]\!](-) \times [\![\mathtt{S}_2]\!](-) \\
[\![\mathtt{N}\,.\,\mathtt{S}]\!](-) & = & [\mathbb{A}]([\![\mathtt{S}]\!](-))
\end{array}$$

Theorem. Given a nominal algebraic signature Σ (for simplicity, assume Σ has a single data-sort $\mathtt{D}$ as well as a single name-sort $\mathtt{N}$)
$\Sigma_\alpha(\mathtt{D})$ is an initial algebra for the associated functor $T_\Sigma : \mathbf{Nom} \to \mathbf{Nom}$.

E.g. for the λ-calculus signature with operations

$\mathtt{V} : \mathtt{Var} \longrightarrow \mathtt{Term}$

$\mathtt{A} : \mathtt{Term}\,,\,\mathtt{Term} \longrightarrow \mathtt{Term}$

$\mathtt{L} : \mathtt{Var}\,.\,\mathtt{Term} \longrightarrow \mathtt{Term}$

we have

$T_\Sigma(-) = \mathbb{A} + (- \times -) + [\mathbb{A}](-)$

Theorem. Given a nominal algebraic signature Σ (for simplicity, assume Σ has a single data-sort $\mathtt{D}$ as well as a single name-sort $\mathtt{N}$)
$\Sigma_\alpha(\mathtt{D})$ is an initial algebra for the associated enriched functor $T_\Sigma : \mathbf{Nom} \longrightarrow \mathbf{Nom}$.

T_Σ not only acts on equivariant (=emptily supported) functions, but also on finitely supported functions:

$$\begin{array}{rcl} (X \to_{\mathrm{fs}} Y) & \longrightarrow & (T_\Sigma\, X \to_{\mathrm{fs}} T_\Sigma\, Y) \\ F & \longmapsto & T_\Sigma\, F \end{array}$$

α-Structural recursion

For λ-terms:

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 & \in & \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 & \in & X \times X \to_{\mathrm{fs}} X \\ f_3 & \in & [\mathbb{A}]X \to_{\mathrm{fs}} X \end{cases}$

$$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X \text{ s.t. } \begin{cases} \hat{f}\, a = f_1\, a \\ \hat{f}\,(e_1\, e_2) = f_2(\hat{f}\, e_1, \hat{f}\, e_2) \\ \hat{f}(\lambda a.e) = f_3(\langle a\rangle(\hat{f}\, e)) \quad \text{if} a \mathbin{\#} (f_1, f_2, f_3) \end{cases}$$

The enriched functor $[\mathbb{A}](-) : \mathbf{Nom} \to \mathbf{Nom}$ sends $f \in X \to_{\mathrm{fs}} Y$ to $[\mathbb{A}]f \in [\mathbb{A}]X \to_{\mathrm{fs}} [\mathbb{A}]Y$ where

$$[\mathbb{A}]f\,(\langle a\rangle x) = \langle a\rangle(f\, x) \quad \text{if } a \mathbin{\#} f$$

α-Structural recursion

For λ-terms:

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 & \in & \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 & \in & X \times X \to_{\mathrm{fs}} X \\ f_3 & \in & \mathbb{A} \times X \to_{\mathrm{fs}} X \end{cases}$ s.t.

$$(\forall a)\ a \mathrel{\#} (f_1, f_2, f_3) \implies (\forall x)\ a \mathrel{\#} f_3(a, x) \qquad \text{(FCB)}$$

$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X$ s.t. $\begin{cases} \hat{f}\, a = f_1\, a \\ \hat{f}\,(e_1\, e_2) = f_2(\hat{f}\, e_1, \hat{f}\, e_2) \\ \hat{f}(\lambda a.e) = f_3(a, \hat{f}\, e) \quad \text{if } a \mathrel{\#} (f_1, f_2, f_3) \end{cases}$

Name abstraction

Recall:

Theorem. $f \in (\mathbb{A} \times X) \to_{\mathrm{fs}} Y$ factors through the subquotient $\mathbb{A} \times X \supseteq \{(a, x) \mid a \# f\} \twoheadrightarrow [\mathbb{A}]X$ to give a unique element of $\overline{f} \in ([\mathbb{A}]X) \to_{\mathrm{fs}} Y$ satisfying

$$\overline{f}(\langle a \rangle x) = f(a, x) \quad \text{if } a \# f$$

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

iff $(\exists a \in \mathbb{A})\ a \# f \wedge (\forall x \in X)\ a \# f(a, x)$.

α-Structural recursion

For λ-terms:

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 & \in & \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 & \in & X \times X \to_{\mathrm{fs}} X \\ f_3 & \in & \mathbb{A} \times X \to_{\mathrm{fs}} X \end{cases}$ s.t.

$$(\forall a)\ a \# (f_1, f_2, f_3) \implies (\forall x)\ a \# f_3(a, x) \qquad \text{(FCB)}$$

$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X$ s.t. $\begin{cases} \hat{f}\ a = f_1\ a \\ \hat{f}\,(e_1\ e_2) = f_2(\hat{f}\ e_1, \hat{f}\ e_2) \\ \hat{f}(\lambda a.e) = f_3(a, \hat{f}\ e) \quad \text{if } a \# (f_1, f_2, f_3) \end{cases}$

E.g. capture-avoiding substitution $(-)[e'/a'] : \Lambda \to \Lambda$ is the $\hat{f}$ for

$$\begin{aligned} f_1\ a &\triangleq \text{if } a = a' \text{ then } e' \text{ else } a \\ f_2(e_1, e_2) &\triangleq e_1\ e_2 \\ f_3(a, e) &\triangleq \lambda a.e \end{aligned}$$

for which (FCB) holds, since $a \# \lambda a.e$

α-Structural recursion

For λ-terms:

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 & \in & \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 & \in & X \times X \to_{\mathrm{fs}} X \\ f_3 & \in & \mathbb{A} \times X \to_{\mathrm{fs}} X \end{cases}$ s.t.

$$(\forall a)\ a \mathrel{\#} (f_1, f_2, f_3) \implies (\forall x)\ a \mathrel{\#} f_3(a, x) \qquad \text{(FCB)}$$

$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X$ s.t. $\begin{cases} \hat{f}\ a = f_1\ a \\ \hat{f}\,(e_1\ e_2) = f_2(\hat{f}\ e_1, \hat{f}\ e_2) \\ \hat{f}(\lambda a.e) = f_3(a, \hat{f}\ e) \quad \text{if } a \mathrel{\#} (f_1, f_2, f_3) \end{cases}$

E.g. size function $\Lambda \to \mathbb{N}$ is the $\hat{f}$ for

$$\begin{array}{rcl} f_1\ a & \triangleq & 0 \\ f_2(n_1, n_2) & \triangleq & n_1 + n_2 \\ f_3(a, n) & \triangleq & n + 1 \end{array}$$

for which (FCB) holds, since $a \mathrel{\#} (n + 1)$

α-Structural recursion

For λ-terms:

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 & \in & \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 & \in & X \times X \to_{\mathrm{fs}} X \\ f_3 & \in & \mathbb{A} \times X \to_{\mathrm{fs}} X \end{cases}$ s.t.

$$(\forall a)\ a \# (f_1, f_2, f_3) \implies (\forall x)\ a \# f_3(a, x) \qquad \text{(FCB)}$$

$$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X \text{ s.t. } \begin{cases} \hat{f}\ a = f_1\ a \\ \hat{f}\ (e_1\ e_2) = f_2(\hat{f}\ e_1, \hat{f}\ e_2) \\ \hat{f}(\lambda a.e) = f_3(a, \hat{f}\ e) \quad \text{if } a \# (f_1, f_2, f_3) \end{cases}$$

Non-example: trying to list the bound variables of a λ-term

$$\begin{array}{rcl} f_1\ a & \triangleq & \mathsf{nil} \\ f_2(\ell_1, \ell_2) & \triangleq & \ell_1 \mathbin{@} \ell_2 \\ f_3(a, \ell) & \triangleq & a :: \ell \end{array}$$

for which (FCB) does not hold, since $a \in \mathrm{supp}(a :: \ell)$.

α-Structural recursion

For λ-terms:

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 & \in & \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 & \in & X \times X \to_{\mathrm{fs}} X \\ f_3 & \in & \mathbb{A} \times X \to_{\mathrm{fs}} X \end{cases}$ s.t.

$$(\forall a)\ a \mathbin{\#} (f_1, f_2, f_3) \implies (\forall x)\ a \mathbin{\#} f_3(a, x) \qquad \text{(FCB)}$$

$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X$ s.t. $\begin{cases} \hat{f}\ a = f_1\ a \\ \hat{f}\,(e_1\ e_2) = f_2(\hat{f}\ e_1, \hat{f}\ e_2) \\ \hat{f}(\lambda a.e) = f_3(a, \hat{f}\ e) \quad \text{if } a \mathbin{\#} (f_1, f_2, f_3) \end{cases}$

Similar results hold for any nominal algebraic signature—see J ACM 53(2006)459–506.

Implemented in Urban & Berghofer's Nominal package for Isabelle/HOL (classical higher-order logic).

Seems to capture informal usage well, but (FCB) can be tricky…

Counting occurrences of bound variables

For each $e \in \Lambda$, $\boxed{\mathrm{cbv}\, e \triangleq f\, e\, \rho_0 \in \mathbb{N}}$

where we want $f \in \Lambda \to_{\mathrm{fs}} X$ with $X = (\mathbb{A} \to_{\mathrm{fs}} \mathbb{N}) \to_{\mathrm{fs}} \mathbb{N}$ to satisfy

$$
\begin{array}{rcl}
f\, a\, \rho & = & \rho\, a \\
f\, (e_1\, e_2)\, \rho & = & (f\, e_1 \rho) + (f\, e_2\, \rho) \\
f(\lambda a.e)\, \rho & = & f\, e\, (\rho[a \mapsto 1])
\end{array}
$$

and where $\rho_0 \in \mathbb{A} \to_{\mathrm{fs}} \mathbb{N}$ is $\lambda(a \in \mathbb{A}) \to 0$.

E.g. when $e = (\lambda a.\, \lambda b.\, a)\, b$ (with $a \neq b$), then e has a single occurrence of a bound variable (called a) and $\mathrm{cbv}\, e = 1$.

Counting occurrences of bound variables

For each $e \in \Lambda$, $\boxed{\mathrm{cbv}\, e \triangleq f\, e\, \rho_0 \in \mathbb{N}}$

where we want $f \in \Lambda \to_{\mathrm{fs}} X$ with $X = (\mathbb{A} \to_{\mathrm{fs}} \mathbb{N}) \to_{\mathrm{fs}} \mathbb{N}$ to satisfy

$$\begin{aligned} f\, a\, \rho &= \rho\, a \\ f\,(e_1\, e_2)\, \rho &= (f\, e_1 \rho) + (f\, e_2\, \rho) \\ f(\lambda a.e)\, \rho &= f\, e\, (\rho[a \mapsto 1]) \end{aligned}$$

and where $\rho_0 \in \mathbb{A} \to_{\mathrm{fs}} \mathbb{N}$ is $\lambda(a \in \mathbb{A}) \to 0$.

Looks like we should take $f_3(a, x) = \lambda(\rho \in \mathbb{A} \to_{\mathrm{fs}} \mathbb{N}) \to x(\rho[a \mapsto 1])$, *but this does not satisfy* (FCB). Solution: take X to be a certain nominal subset of $(\mathbb{A} \to_{\mathrm{fs}} \mathbb{N}) \to_{\mathrm{fs}} \mathbb{N}$. [See Nominal Sets book, Example 8.20]

Lecture 3

Outline

L1 Structural recursion and induction in the presence of name-binding operations. Introducing the category of nominal sets.

L2 Nominal algebraic data types and α-structural recursion.

L3 Dependently typed λ-calculus with locally fresh names and name-abstraction.

References:

AMP, *Alpha-Structural Recursion and Induction*, JACM 53(2006)459-506.

AMP, J. Matthiesen and J. Derikx,
A Dependent Type Theory with Abstractable Names, ENTCS 312(2015)19-50.

Original motivation for Gabbay & AMP to introduce nominal sets and name abstraction:

> $[\mathbb{A}](\_)$ can be combined with $\times$ and $+$ to give functors $\mathbf{Nom} \rightarrow \mathbf{Nom}$ that have initial algebras coinciding with sets of abstract syntax trees modulo α-equivalence.

E.g. the initial algebra for $\mathbb{A} + (\_ \times \_) + [\mathbb{A}](\_)$ is isomorphic to the usual set of untyped λ-terms.

Recall: α-Structural recursion

For λ-terms:

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 & \in & \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 & \in & X \times X \to_{\mathrm{fs}} X \\ f_3 & \in & \mathbb{A} \times X \to_{\mathrm{fs}} X \end{cases}$ s.t.

$$(\forall a)\ a \mathrel{\#} (f_1, f_2, f_3) \implies (\forall x)\ a \mathrel{\#} f_3(a, x) \qquad \text{(FCB)}$$

$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X$ s.t. $\begin{cases} \hat{f}\, a = f_1\, a \\ \hat{f}\,(e_1\, e_2) = f_2(\hat{f}\, e_1, \hat{f}\, e_2) \\ \hat{f}(\lambda a.e) = f_3(a, \hat{f}\, e) \quad \text{if } a \mathrel{\#} (f_1, f_2, f_3) \end{cases}$

Can we avoid explicit reasoning about finite support, # and (FCB) when computing 'mod α'?

Want definition/computation to be separate from proving.

$$
\begin{aligned}
\hat{f}\,a &= f_1\, a \\
\hat{f}(e_1\, e_2) &= f_2(\hat{f}\, e_1, \hat{f}\, e_2) \\
\hat{f}(\lambda a.\, e) &= f_3(a, \hat{f}\, e) \qquad \text{if } a \mathbin{\#} (f_1, f_2, f_2)
\end{aligned}
$$

$= \lambda a'.\, e'$ $\qquad\qquad$ $= f_3(a', \hat{f}\, e')$

Q: how to get rid of this inconvenient proof obligation?

$$
\begin{aligned}
\hat{f} &= f_1\, a \\
\hat{f}(e_1\, e_2) &= f_2(\hat{f}\, e_1, \hat{f}\, e_2) \\
\hat{f}(\lambda a.\, e) &= \nu a.\, f_3(a, \hat{f}\, e) \quad [\, a \mathrel{\#} (f_1, f_2, f_2)\,]
\end{aligned}
$$

$= \lambda a'.\, e'$ $\qquad$ $= \nu a'.\, f_3(a', \hat{f}\, e')$ OK!

Q: how to get rid of this inconvenient proof obligation?

A: use a local scoping construct $\nu a.\,(-)$ for names

$$
\begin{aligned}
\hat{f} \;&=\; f_1\, a\\
\hat{f}(e_1\, e_2) \;&=\; f_2(\hat{f}\, e_1, \hat{f}\, e_2)\\
\hat{f}(\lambda a.\, e) \;&=\; \nu a.\, f_3(a, \hat{f}\, e) \quad [\, a \mathrel{\#} (f_1, f_2, f_2)\,]
\end{aligned}
$$

$= \lambda a'.\, e'$ $\qquad$ $= \nu a'.\, f_3(a', \hat{f}\, e')$ OK!

Q: how to get rid of this inconvenient proof obligation?

A: use a local scoping construct $\nu a.\,(-)$ for names

which one?!

Dynamic allocation

- Stateful: $\nu a.\, t$ means "add a fresh name a' to the current state and return $t[a'/a]$".
- Used in Shinwell's Fresh OCaml = OCaml +
  - name types and name-abstraction type former
  - name-abstraction patterns
    —matching involves dynamic allocation of fresh names

  [MR Shinwell, AMP, MJ Gabbay,
  FreshML: Programming with Binders Made Simple, Proc. ICFP 2003.]

  [`www.cl.cam.ac.uk/users/amp12/fresh-ocaml`]

Sample Fresh OCaml code

```
(* syntax *)
type t;;
type var = t name;;
type term = Var of var | Lam of <<var>>term | App of term*term;;

 (* semantics *)
type sem = L of ((unit -> sem) -> sem) | N of neu
and  neu = V of var | A of neu*sem;;

 (* reify : sem -> term *)
let rec reify d =
  match d with L f -> let x = fresh in Lam(<<x>>(reify(f(function () -> N(V x)))))
             | N n -> reifyn n
and reifyn n =
  match n with V x -> Var x
             | A(n',d') -> App(reifyn n', reify d');;

(* evals : (var * (unit -> sem))list -> term -> sem *)
let rec evals env t  =
  match t with Var x -> (match env with [] -> N(V x)
                                   | (x',v)::env -> if x=x' then v() else evals env (Var x))
             | Lam(<<x>>t) -> L(function v -> evals ((x,v)::env) t)
             | App(t1,t2) -> (match evals env t1 with L f -> f(function () -> evals env t2)
                                                  | N n -> N(A(n,evals env t2)));;

(* eval : term -> sem *)
let rec eval t = evals [] t;;

(* norm : lam -> lam *)
let norm t = reify(eval t);;
```

Dynamic allocation

- Stateful: $\nu a.\, t$ means "add a fresh name a' to the current state and return $t[a'/a]$".
- Used in Shinwell's Fresh OCaml = OCaml +
  - name types and name-abstraction type former
  - name-abstraction patterns
    —matching involves dynamic allocation of fresh names

[MR Shinwell, AMP, MJ Gabbay,
FreshML: Programming with Binders Made Simple, Proc. ICFP 2003.]

[www.cl.cam.ac.uk/users/amp12/fresh-ocaml]

Dynamic allocation

- Stateful: $\nu a.\, t$ means "add a fresh name a' to the current state and return $t[a'/a]$".

Statefulness disrupts familiar mathematical properties of pure datatypes. So let's try to reject it in favour of…

Aim

A version of Martin-Löf Type Theory
enriched with constructs for
locally fresh names and name-abstraction
from the theory of nominal sets.

Motivation:

Machine-assisted construction of
humanly understandable formal proofs
about software (PL semantics).

Aim

More specifically: extend (dependently typed) λ-calculus with

- names a
- name swapping $\texttt{swap}\ a, b\ \texttt{in}\ t$
- name abstraction $\langle a \rangle t$ and concretion $t\ @\ a$
- locally fresh names $\texttt{fresh}\ a\ \texttt{in}\ t$
- name equality $\texttt{if}\ \ t = a\ \texttt{then}\ t_1\ \texttt{else}\ t_2$

Locally fresh names

For example, here are some isomorphisms, described in an informal pseudocode:

$$
\begin{array}{rcl}
i : [\mathbb{A}](X + Y) & \cong & [\mathbb{A}]X + [\mathbb{A}]Y \\
i(z) & = & \text{fresh } a \text{ in case } z \mathbin{@} a \text{ of} \\
 & & \quad \mathrm{inl}(x) \to \langle a \rangle x \\
 & & \mid \mathrm{inr}(y) \to \langle a \rangle y
\end{array}
$$

[Ex. 7]

Locally fresh names

For example, here are some isomorphisms, described in an informal pseudocode:

$$\begin{array}{rcl} i : [\mathbb{A}](X+Y) & \cong & [\mathbb{A}]X + [\mathbb{A}]Y \\ i(z) & = & \text{fresh } a \text{ in case } z \mathbin{@} a \text{ of} \\ & & \quad \mathrm{inl}(x) \to \langle a\rangle x \\ & & \mid \mathrm{inr}(y) \to \langle a\rangle y \end{array}$$

given $f \in \mathbf{Nom}(X * \mathbb{A}, Y)$
satisfying $a \mathrel{\#} x \implies a \mathrel{\#} f(x,a)$,
we get $\hat{f} \in \mathbf{Nom}(X, Y)$ well-defined by:
$\hat{f}(x) = f(x,a)$ for some/any $a \mathrel{\#} x$.
Notation: $\boxed{\text{fresh } a \text{ in } f(x,a)} \triangleq \hat{f}(x)$

Locally fresh names

For example, here are some isomorphisms, described in an informal pseudocode:

$$
\begin{array}{rcl}
i : [\mathbb{A}](X + Y) & \cong & [\mathbb{A}]X + [\mathbb{A}]Y \\
i(z) & = & \text{fresh } a \text{ in case } z \mathbin{@} a \text{ of} \\
 & & \quad \mathrm{inl}(x) \to \langle a \rangle x \\
 & & \mid \mathrm{inr}(y) \to \langle a \rangle y \\
j : ([\mathbb{A}]X \to [\mathbb{A}]Y) & \cong & [\mathbb{A}](X \to Y) \\
j(f) & = & \text{fresh } a \text{ in} \\
 & & \qquad \langle a \rangle(\lambda x.\ f(\langle a \rangle x) \mathbin{@} a)
\end{array}
$$

Can one turn the pseudocode into terms in a formal 'nominal' λ-calculus?

Prior art

- Stark-Schöpp [CSL 2004]

  bunched contexts (+), extensional & undecidable (–)
- Westbrook-Stump-Austin [LFMTP 2009] CNIC

  semantics/expressivity?
- Cheney [LMCS 2012] DNTT

  bunched contexts (+), no local fresh names (–)
- Fairweather-Fernández-Szasz-Tasistro [2012]

  based on nominal terms (+), explicit substitutions (–), first-order (±)
- Crole-Nebel [MFPS 2013]

  simple types (–), definitional freshness (+)

Our art

- Stark-Schöpp [CSL 2004]

  bunched contexts (+), extensional & undecidable (–)
- Westbrook-Stump-Austin [LFMTP 2009] CNIC

  semantics/expressivity?
- Cheney [LMCS 2012] DNTT

  bunched contexts (+), no local fresh names (–)
- Fairweather-Fernández-Szasz-Tasistro [2012]

  based on nominal terms (+), explicit substitutions (–), first-order (±)
- Crole-Nebel [MFPS 2013]

  simple types (–), definitional freshness (+)

AMP, J. Matthiesen and J. Derikx, *A Dependent Type Theory with Abstractable Names*, ENTCS 312(2015)19-50.

Aim

More specifically: extend (dependently typed) λ-calculus with

- names a
- name swapping $\texttt{swap}\ a, b\ \texttt{in}\ t$
- name abstraction $\langle a \rangle t$ and concretion $t \mathbin{@} a$
- locally fresh names $\texttt{fresh}\ a\ \texttt{in}\ t$
- name equality $\texttt{if}\ \ t = a\ \texttt{then}\ t_1\ \texttt{else}\ t_2$

Difficulty: concretion and locally fresh names are partially defined – have to check freshness conditions.

e.g. for fresh a in $f(x, a)$ to be well-defined, we need

$$\boxed{a \mathbin{\#} x \;\Rightarrow\; a \mathbin{\#} f(x, a)}$$

Definitional freshness

In a nominal set of (higher-order) functions, proving $a \# f$ can be tricky (undecidable). Common proof pattern:

> Given $a, f, \ldots$, pick a fresh name b and prove $(a\ b) \cdot f = f$. (For functions, equivalent to proving $\forall x.\ (a\ b) \cdot f(x) = f((a\ b) \cdot x)$.)

Definitional freshness

In a nominal set of (higher-order) functions, proving $a \# f$ can be tricky (undecidable). Common proof pattern:

> Given $a, f, \ldots$, pick a fresh name b and prove $(a\ b) \cdot f = f$.
> Since by choice of b we have $b \# f$, we also get $a = (a\ b) \cdot b \# (a\ b) \cdot f = f$, QED.

Definitional freshness

$$\frac{\Gamma \vdash a \# T \qquad \Gamma \vdash t : T \qquad \Gamma\#(b : \mathbb{A}) \vdash (\mathtt{swap}\ a, b\ \mathtt{in}\ t) = t : T}{\Gamma \vdash a \# t : T}$$

bunched contexts, generated by

$$\begin{array}{lcl} \Gamma & \mapsto & \Gamma(x : T) \\ \Gamma & \mapsto & \Gamma\#(a : \mathbb{A}) \end{array}$$

definitional freshness

definitional equality

Definitional freshness

$$\frac{\Gamma \vdash a \mathbin{\#} T \qquad \Gamma \vdash t : T \qquad \Gamma\#(b : \mathbb{A}) \vdash (\mathtt{swap}\ a, b\ \mathtt{in}\ t) = t : T}{\Gamma \vdash a \# t : T}$$

Freshness info in bunched contexts gets used via:

$$\frac{\Gamma(x : T)\Gamma' \text{ ok} \qquad a, b \in \Gamma'}{\Gamma(x : T)\Gamma' \vdash (\mathtt{swap}\ a, b\ \mathtt{in}\ x) = x : T}$$

Definitional freshness

$$\frac{\Gamma \vdash a \# T \qquad \Gamma \vdash t : T \qquad \Gamma\#(b : \mathbb{A}) \vdash (\mathtt{swap}\ a, b\ \mathtt{in}\ t) = t : T}{\Gamma \vdash a \# t : T}$$

definitional freshness for types:

$$\frac{\Gamma \vdash T \qquad a \in \Gamma \qquad \Gamma\#(b : \mathbb{A}) \vdash (\mathtt{swap}\ a, b\ \mathtt{in}\ T) = T}{\Gamma \vdash a \# T}$$

A type theory

$\boxed{\Gamma \vdash}$

$$\frac{\Gamma \vdash \qquad a \notin \mathrm{dom}\,\Gamma}{\Gamma(\# a) \vdash}\ (\textsc{atm-hyp})$$

$\boxed{\Gamma \vdash T}$

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Atm}}\ (\textsc{atm-form}) \qquad \frac{\Gamma(\# a) \vdash T}{\Gamma \vdash (\# a) \to T}\ (\textsc{abs-form}) \qquad \frac{\Gamma(\# a) \vdash a \# T}{\Gamma \vdash \nu a.\, T}\ (\textsc{local-form})$$

$\boxed{\Gamma \vdash a \# T}$

$$\frac{a \in \mathrm{dom}\,\Gamma \qquad \Gamma \vdash T \qquad \Gamma(\# a') \vdash (a\ a') * T = T}{\Gamma \vdash a \# T}\ (\textsc{def-fresh-1})$$

$\boxed{\Gamma \vdash T = T'}$

$$\frac{\Gamma(\# a) \vdash a \# T \qquad \Gamma(\# a)\Gamma' \vdash}{\Gamma(\# a)\Gamma' \vdash \nu a.\, T = T}\ (\textsc{local-comp})$$

$\boxed{\Gamma \vdash e : T}$

$$\frac{\Gamma(x : T)\Gamma' \vdash \qquad \mathrm{supp}\,\pi \subseteq \mathrm{dom}\,\Gamma\Gamma'}{\Gamma(x : T)\Gamma' \vdash \pi * x : \pi * T}\ (\textsc{susp}) \qquad \frac{\Gamma \vdash \qquad a \in \mathrm{dom}\,\Gamma}{\Gamma \vdash a : \mathsf{Atm}}\ (\textsc{atm-intro})$$

$$\frac{\begin{array}{c}\Gamma \vdash e : \mathsf{Atm} \qquad a \in \mathrm{dom}\,\Gamma \\ \Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T\end{array}}{\Gamma \vdash \mathsf{if}\ e = a\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : T}\ (\textsc{if-intro}) \qquad \frac{\Gamma(\# a) \vdash a \# (e : T)}{\Gamma \vdash \nu a.\, e : \nu a.\, T}\ (\textsc{local-intro})$$

$$\frac{\Gamma(\# a) \vdash e : T}{\Gamma \vdash \alpha(\# a) \to e : (\# a) \to T}\ (\textsc{abs-intro}) \qquad \frac{\Gamma \vdash a' \# (e : (\# a) \to T)}{\Gamma \vdash e \mathbin{@} a' : \nu a.\, (a\ a') * T}\ (\textsc{abs-elim})$$

$\boxed{\Gamma \vdash a \# (e : T)}$

$$\frac{\Gamma \vdash a \# T \qquad \Gamma \vdash e : T \qquad \Gamma(\# a') \vdash (a\ a') * e = e : T}{\Gamma \vdash a \# (e : T)}\ (\textsc{def-fresh-2})$$

$\boxed{\Gamma \vdash e = e' : T}$

$$\frac{\Gamma\Gamma' \vdash e : T \qquad \Gamma\Gamma' \vdash e' : T \qquad \Gamma(\# a)\Gamma' \vdash e = e' : T}{\Gamma\Gamma' \vdash e = e' : T}\ (\textsc{atm-strengthen})$$

$$\frac{\Gamma(x : T)\Gamma' \vdash \qquad a, a' \in \mathrm{dom}\,\Gamma'}{\Gamma(x : T)\Gamma' \vdash (a\ a') * x = x : T}\ (\textsc{swap-fresh-var})$$

$$\frac{a \in \mathrm{dom}_{\mathbb{A}}\Gamma \qquad \Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash (\mathsf{if}\ a = a\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = e_1 : T}\ (\textsc{if-comp-1})$$

$$\frac{\Gamma \vdash a \# (e : \mathsf{Atm}) \qquad \Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash (\mathsf{if}\ e = a\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = e_2 : T}\ (\textsc{if-comp-2})$$

$$\frac{\Gamma(\# a) \vdash a \# (e : T) \qquad \Gamma(\# a)\Gamma' \vdash}{\Gamma(\# a)\Gamma' \vdash \nu a.\, e = e : T}\ (\textsc{local-comp-2})$$

$$\frac{\Gamma(\# a) \vdash e : T \qquad \Gamma(\# a) \vdash a' \# (e : T) \qquad a \neq a'}{\Gamma \vdash (\alpha(\# a) \to e) \mathbin{@} a' = \nu a.\, (a\ a') * e : \nu a.\, (a\ a') * T}\ (\textsc{abs-comp})$$

$$\frac{\Gamma \vdash e : (\# a) \to T}{\Gamma \vdash e = \alpha(\# a) \to (e \mathbin{@} a) : (\# a) \to T}\ (\textsc{abs-uniq})$$

A type theory

OMITTED

$\boxed{\Gamma \vdash}$

$$\frac{\Gamma \vdash \qquad a \notin \mathrm{dom}\,\Gamma}{\Gamma(\# a) \vdash}\ (\text{ATM-HYP})$$

$\boxed{\Gamma \vdash T}$

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Atm}}\ (\text{ATM-FORM}) \qquad \frac{\Gamma(\# a) \vdash T}{\Gamma \vdash (\# a) \to T}\ (\text{ABS-FORM}) \qquad \frac{\Gamma(\# a) \vdash a \#}{\Gamma \vdash}$$

$\boxed{\Gamma \vdash a \# T}$

$$\frac{a \in \mathrm{dom}\,\Gamma \qquad \Gamma \vdash T \qquad \Gamma(\# a')}{\Gamma \vdash a}$$

$\boxed{\Gamma \vdash T = T'}$

Γ

(…M-INTRO)

$$\frac{a \# (e : T)}{\vdash \nu a.e : \nu a.T}\ (\text{LOCAL-INTRO})$$

$$\frac{\Gamma \vdash a' \# (e : (\# a) \to T)}{\Gamma \vdash e\,@\,a' : \nu a.(a\ a') * T}\ (\text{ABS-ELIM})$$

(DEF-FRESH-2)

$$\frac{\Gamma(\# a)\Gamma' \vdash e = e' : T}{e = e' : T}\ (\text{ATM-STRENGTHEN})$$

$$\frac{: T)\Gamma' \vdash \qquad a, a' \in \mathrm{dom}\,\Gamma'}{\Gamma(x : T)\Gamma' \vdash (a\ a') * x = x : T}\ (\text{SWAP-FRESH-VAR})$$

$$\frac{a \in \mathrm{dom}_{\mathbb{A}}\Gamma \qquad \Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash (\mathsf{if}\ a = a\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = e_1 : T}\ (\text{IF-COMP-1})$$

$$\frac{\Gamma \vdash a \# (e : \mathsf{Atm}) \qquad \Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash (\mathsf{if}\ e = a\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = e_2 : T}\ (\text{IF-COMP-2})$$

$$\frac{\Gamma(\# a) \vdash a \# (e : T) \qquad \Gamma(\# a)\Gamma' \vdash}{\Gamma(\# a)\Gamma' \vdash \nu a.e = e : T}\ (\text{LOCAL-COMP-2})$$

$$\frac{\Gamma(\# a) \vdash e : T \qquad \Gamma(\# a) \vdash a' \# (e : T) \qquad a \neq a'}{\Gamma \vdash (\alpha(\# a) \to e)\,@\,a' = \nu a.(a\ a') * e : \nu a.(a\ a') * T}\ (\text{ABS-COMP})$$

$$\frac{\Gamma \vdash e : (\# a) \to T}{\Gamma \vdash e = \alpha(\# a) \to (e\,@\,a) : (\# a) \to T}\ (\text{ABS-UNIQ})$$

Nominal set semantics of dependent type theory

A family over $X \in \mathbf{Nom}$ is specified by:

- X-indexed family of sets $(Y_x \mid x \in X)$
- dependently type permutation action

$$\prod_{\pi \in \mathrm{Perm}\,\mathbb{A}} \prod_{x \in X} (Y_x \to Y_{\pi \cdot x})$$

with dependent version of finite support property:
for all $x \in X, e \in Y_x$ there is a finite set A of names supporting x in X and such that any π fixing each $a \in A$ satisfies
$$\begin{array}{ccc} \pi \cdot e & = & e \\ \mathrel{\rotatebox{90}{$\in$}} & & \mathrel{\rotatebox{90}{$\in$}} \\ Y_{\pi \cdot x} & = & Y_x \end{array}$$

Nominal set semantics of dependent type theory

A family over $X \in \mathbf{Nom}$ is specified by...

Get a category with families (CwF) [Dybjer] modelling extensional MLTT, plus

nominal logic's freshness quantifier	Curry-Howard	dependent name-abstraction
$\reflectbox{N}a.\ \varphi(a, \vec{x})$	$\longleftrightarrow$	$[a \in \mathbb{A}]Y_a$

Nominal set semantics of dependent type theory

A family over $X \in \mathbf{Nom}$ is specified by…

Get a category with families (CwF) [Dybjer] modelling extensional MLTT, plus

nominal logic's freshness quantifier	Curry-Howard	dependent name-abstraction
$\reflectbox{N}a.\ \varphi(a, \vec{x})$	$\longleftrightarrow$	$[a \in \mathbb{A}]Y_a$

$= \exists a \# \vec{x}.\ \varphi(a, \vec{x})$
$= \forall a \# \vec{x}.\ \varphi(a, \vec{x})$
'some/any fresh a'

For more details, see

AMP, J. Matthiesen and J. Derikx,
A Dependent Type Theory with Abstractable Names, ENTCS 312(2015)19-50

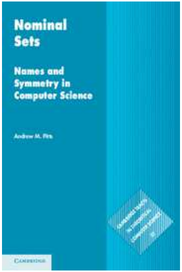
But much remains to do, e.g.

- Explore inductively defined types involving $[a : \mathbb{A}](\_)$ (e.g. propositional freshness).
- Dependently typed pattern-matching with name-abstraction patterns.

Difficulties:

- Is definitional freshness too weak? (cf. experience with FreshML2000)
- Name-swapping with variables of type $\mathbb{A}$

Advert



Nominal Sets

Names and Symmetry in Computer Science

Cambridge Tracts in Theoretical Computer Science, Vol. 57 (CUP, 2013)