An Introduction to Nominal Sets

Andrew Pitts

UNIVERSITY OF CAMBRIDGE

Computer Science & Technology

EWSCS 2020

Lecture 4

Outline

L1 Structural recursion and induction in the presence of name-binding operations.

L2 Introducing the category of nominal sets.

L3 Nominal algebraic data types and α-structural recursion.

L4 Dependently typed λ-calculus with locally fresh names and name-abstraction.

References:

AMP, *Nominal Sets: Names and Symmetry in Computer Science*, CUP 2013

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 $\mathbf{Nom} \to \mathbf{Nom}$ 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.

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

$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X$ 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.

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

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

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

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

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

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

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

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

$= \lambda a'.\, e'$

$= \nu a'.\, f_3(a', \hat{f}\, e')$ *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:

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

[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 $a \# x \;\Rightarrow\; a \# f(x, a)$,
we get $\hat{f} \in \mathbf{Nom}(X, Y)$ well-defined by:
$\hat{f}(x) = f(x, a)$ for some/any $a \# x$.
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:

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

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

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

$$\Gamma \vdash a\# \; t : T$$

bunched contexts, generated by

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

definitional freshness

definitional equality

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:

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

Definitional freshness

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

Freshness info in bunched contexts gets used via:

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

A type theory

$\boxed{\Gamma \vdash}$

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

$\boxed{\Gamma \vdash T}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

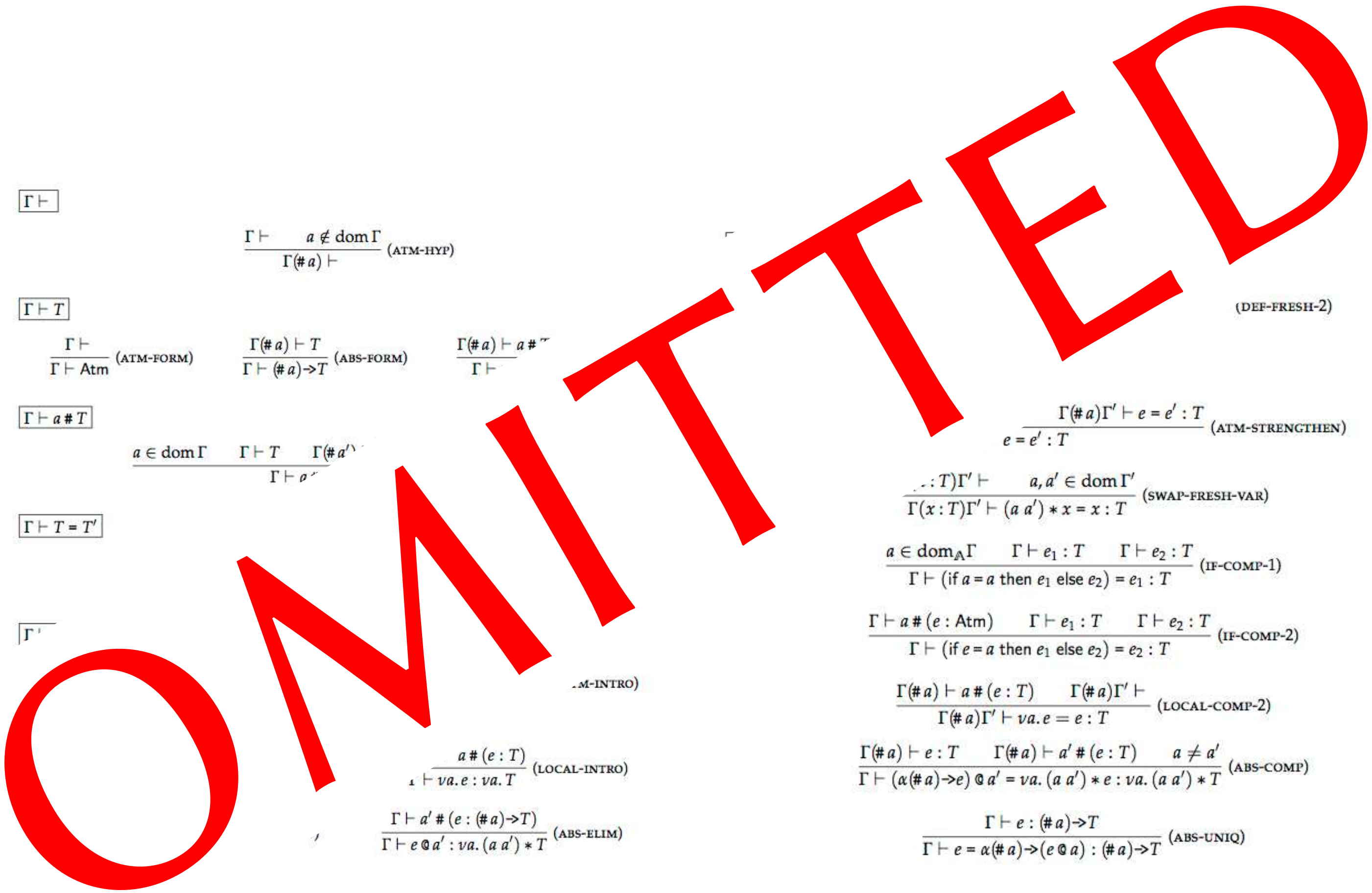
$$\frac{\Gamma \vdash a \# (e : \mathsf{Atm}) \qquad \Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash (\mathsf{if}\ e = a\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = e_2 : T}\ \textsc{(if-comp-2)}$$

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

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

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

A type theory



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
$$\begin{array}{ccc} \pi \cdot e & = & e \\ \Cap & & \Cap \\ Y_{\pi \cdot x} & = & Y_x \end{array}$$

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
$\text{И}a.\ \varphi(a)$	$\longleftrightarrow$	$[a \in \mathbb{A}]\, Y_a$

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 $\mathbb{A}$

Other applications of nominal sets

- **Computational logic**
  - Higher-order logic: Urban & Berghofer's Nominal package for the interactive theorem-prover Isabelle/HOL.
  - Equational logic: unification & rewriting for nominal terms [Fernandez+Gabbay+Levy+Villaret+$\cdots$]
    Logic programming mod α (e.g. Cheney's αProlog)

Other applications of nominal sets

- **Computational logic**
  - Higher-order logic: Urban & Berghofer's Nominal package for the interactive theorem-prover Isabelle/HOL.
  - Equational logic: unification & rewriting for nominal terms [Fernandez+Gabbay+Levy+Villaret+$\cdots$]
    Logic programming mod α (e.g. Cheney's αProlog)
- **Automata theory & verification**
  - HD-automata [Montanari el al]
  - fresh-register automata [Tzevelekos]
  - orbit-finite computation theory [Bojańczyk et al]

Other applications of nominal sets

- **Homotopy Type Theory** (HoTT)
  Cubical sets [Bezem-Coquand-Huber] model of Voevodsky's axiom of univalence makes use of nominal sets equipped with an operation of substitution $x \mapsto x(i/a)$ where $i \in \{0, 1\}$.
  - names are names of directions (cartesian axes)
    (so e.g., if an object has support $\{a, b, c\}$ it is 3-dimensional)
  - freshness ($a \# x$) = degeneracy ($x(i/a) = x$)
  - identity types are modelled by name-abstraction: $\langle a \rangle x$ is a proof that $x(0/a)$ is equal to $x(1/a)$.

HoTT and univalence is about (computable) *mathematical foundations* (a topic no longer very popular with mathematicians). That's where the mathematics of nominal sets came from…

Impact can take a long time

The mathematics behind nominal sets goes back a long way…

Abraham Fraenkel, *Der Begriff "definit" und die Unabhängigkeit des Auswahlsaxioms*, Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse (1922), 253–257.

Andrzej Mostowski, *Uber die Unabhängigkeit des Wohlordnungssatzes vom Ordnungsprinzip*, Fundamenta Mathematicae 32 (1939), 201–252.

Impact can take a long time

The mathematics behind nominal sets goes back a long way…

…and it's still too early to tell what will be the impact of the applications of it to CS developed over the last 20 years.

Two take-home messages from these lectures:

- Specific: in meta-programming/proving, permutation comes before substitution and (hence) name-abstraction before lambda-abstraction
- General: computation modulo symmetry deserves further exploration.