Nominal Sets and Dependent Type Theory

Andrew Pitts

UNIVERSITY OF CAMBRIDGE
Computer Laboratory
Plan

\{ 
\begin{align*}
\text{Type Theory} \\
\text{presheaf categories}
\end{align*}
\} \quad \text{audience listens to talk}

\}
Plan

Type Theory
presheaf categories

audience listens to talk

nominal sets:
freshness
name abstraction

aim to explain the notions of
freshness and name-abstraction

from the theory of nominal sets

and discuss two (on-going) applications involving dependent types:

1. Cubical sets model of Homotopy Type Theory.

2. A version of Type Theory with names, freshness and name-abstraction.
Freshness
What is a fresh name?

Possible definition: name $a$ is fresh if it is not ‘stale’: $a$ is not equal to any name in the current (finite) set of used names (and we extend that set with $a$)
What is a fresh name?

Possible definition: name $a$ is fresh if it is not ‘stale’:

- $a$ is not equal to any name in the current (finite) set of used names (and we extend that set with $a$)

- need to be able to test names for equality – that is the only attribute we assume names have (atomic names)
What is a fresh name?

Possible definition: name $a$ is fresh if it is not ‘stale’:

- $a$ is not equal to any name in the current (finite) set of used names (and we extend that set with $a$)

- need to be able to test names for equality – that is the only attribute we assume names have (atomic names)

- freshness has a modal character – suggests using Kripke-Beth-Joyal (possible worlds) semantics with...
Presheaf semantics

\[ \mathbb{I} = \text{category of finite ordinals} \]
\[ n = \{0, 1, \ldots, n - 1\} \]
and injective functions

\[ U \in [\mathbb{I}, \text{Set}] \]
Presheaf semantics

\[ \mathbb{I} = \text{category of finite ordinals} \]
\[ n = \{0, 1, \ldots, n - 1\} \]
and injective functions

\[ U \in [\mathbb{I}, \text{Set}] \]

\[ [\mathbb{I}, \text{Set}] = \text{(covariant) presheaf category: set-valued functors } X \text{ & natural transformations.} \]
\[ X_n = \text{set of objects (of some type) possibly involving } n \text{ distinct names} \]
Presheaf semantics

\[ \mathcal{I} = \text{category of finite ordinals} \]
\[ n = \{0, 1, \ldots, n-1\} \]  
and injective functions

\[ \mathbb{U} = \text{inclusion functor:} \]
\[ \mathbb{U} n = \{0, 1, \ldots, n-1\} \]

\[ \mathcal{I}, \text{Set} = (\text{covariant}) \text{ presheaf category:} \]
set-valued functors \( X \) & natural transformations.
\[ X n = \text{set of objects (of some type)} \]
possibly involving \( n \) distinct names
$U$ is a ‘decidable’ object of the topos $[\mathbb{I}, \text{Set}]$.

The diagonal subobject $U \hookrightarrow U \times U$ has a boolean complement $\not\hookrightarrow U \times U$. 
$U$ is a ‘decidable’ object of the topos $[\mathbb{I}, \text{Set}]$

\[ a =_U b \land \neq (a,b) \Rightarrow \text{false} \]
\[ \text{true} \Rightarrow a =_U b \lor \neq (a,b) \]
**Generic infinite decidable object**

$\mathbb{U}$ is a ‘decidable’ object of the topos $[\mathbb{I}, \text{Set}]$

$$a =_\mathbb{U} b \land \neq (a, b) \Rightarrow \text{false}$$

$$\text{true} \Rightarrow a =_\mathbb{U} b \lor \neq (a, b)$$

but it does not satisfy ‘finite inexhaustibility’

$$\land_{0 \leq i < j \leq n} \neq (a_i, a_j) \Rightarrow \lor_{b: \mathbb{U}} \land_{0 \leq i \leq n} \neq (b, a_i)$$

which we need to model freshness.
Generic infinite decidable object

$U$ is a ‘decidable’ object of the topos $[\mathbb{I}, \text{Set}]$

$$a =_U b \land \neq (a, b) \Rightarrow \text{false}$$

$$\text{true} \Rightarrow a =_U b \lor \neq (a, b)$$

but it does not satisfy ‘finite inexhaustibility’

$$\land_{0 \leq i < j \leq n} \neq (a_i, a_j) \Rightarrow \lor_{b : U} \land_{0 \leq i \leq n} \neq (b, a_i)$$

**FACT:** we get this form of infinity (in a geometrically generic way) if we cut down to the Schanuel topos:

$\text{Sch} \subseteq [\mathbb{I}, \text{Set}]$ is the full subcategory consisting of functors $\mathbb{I} \to \text{Set}$ that preserve pullbacks
Generic infinite decidable object

\( U \) is a ‘decidable’ object of the topos \([\mathbb{I}, \text{Set}]\)

\[
a =_U b \land \neq (a, b) \Rightarrow \text{false}
\]

\[
\text{true} \Rightarrow a =_U b \lor \neq (a, b)
\]

but it does not satisfy ‘finite inexhaustibility’

\[
\land_{0 \leq i < j \leq n} \neq (a_i, a_j) \Rightarrow \lor_{b: U} \land_{0 \leq i \leq n} \neq (b, a_i)
\]

**FACT:** we get this form of infinity (in a geometrically generic way) if we cut down to the Schanuel topos.

What is the history of this notion? (Kuratowski?)
From Sch to Nom

The category of nominal sets $\text{Nom}$ is ‘merely’ an equivalent presentation of the category $\text{Sch}$:

An analogy:

\[
\begin{array}{cccc}
\text{Nom} & \sim & \text{named bound variables} \\
\text{Sch} & \sim & \text{de Bruijn indexes (levels)}
\end{array}
\]

Step 1: fix a countably infinite set $\mathbb{A}$ (of atomic names) and modify $\text{Sch}$ up to equivalence by replacing $\mathbb{I}$ by the equivalent category whose objects are finite subsets $I \in P_{\text{fin}} \mathbb{A}$ and whose morphisms are injective functions.
From **Sch** to **Nom**

The category of nominal sets \( \textbf{Nom} \) is ‘merely’ an equivalent presentation of the category \( \textbf{Sch} \):

Step 2: make the dependence of each \( X \in \textbf{Sch} \) on ‘possible worlds’ \( A \in P_{\text{fin}} A \) implicit by taking the colimit \( \tilde{X} \) of the directed system of sets and (injective) functions

\[
A \subseteq B \in P_{\text{fin}} A \mapsto (X A \rightarrow X B)
\]

Each set \( \tilde{X} \) carries an action of \( A \)-permutations

(cf. homogeneity property (Fraïssé limit))

\[
\begin{array}{ccc}
A & \overset{\cong}{\longrightarrow} & A \\
\downarrow & & \uparrow \\
A & \overset{f}{\longrightarrow} & B
\end{array}
\]
From $\text{Sch}$ to $\text{Nom}$

The category of nominal sets $\text{Nom}$ is ‘merely’ an equivalent presentation of the category $\text{Sch}$:

Step 2: make the dependence of each $X \in \text{Sch}$ on ‘possible worlds’ $A \in P_{\text{fin}} A$ implicit by taking the colimit $\tilde{X}$ of the directed system of sets and (injective) functions

$$A \subseteq B \in P_{\text{fin}} A \mapsto (X A \to X B)$$

Each set $\tilde{X}$ carries an action of $A$-permutations with finite support property, and every such arises this way up to iso.
Finite support property

Suppose \( \text{Perm} A (= \text{group of all (finite) permutations of } A) \) acts on a set \( X \) and that \( x \in X \).

A set of names \( A \subseteq A \) supports \( x \) if permutations \( \pi \) that fix every \( a \in A \) also fix \( x \) (i.e. \( \pi \cdot x = x \)).

\( X \) is a nominal set if every \( x \in X \) has a finite support.

\( \text{Nom} \) = category of nominal sets and functions that preserve the permutation action \( (f(\pi \cdot x) = \pi \cdot (f \cdot x)) \).

**FACT:** \( \text{Nom} \) and \( \text{Sch} \) are equivalent categories.

Within \( \text{Nom} \), objects are ‘set-like’ and the modal character of freshness becomes implicit...
Finite support property

Suppose $\text{Perm} A$ (\(= \text{group of all (finite) permutations of } A\)) acts on a set $X$ and that $x \in X$.

A set of names $A \subseteq A$ supports $x$ if permutations $\pi$ that fix every $a \in A$ also fix $x$ (i.e. $\pi \cdot x = x$).

$X$ is a nominal set if every $x \in X$ has a finite support.

Freshness, nominally, is a binary relation $a \# x \triangleq a \not\in A$ for some finite $A$ supporting $x$.

‘name $a$ is fresh for $x$’
Finite support property

Suppose \( \text{Perm} A \) (= group of all (finite) permutations of \( A \)) acts on a set \( X \) and that \( x \in X \).

A set of names \( A \subseteq A \) supports \( x \) if permutations \( \pi \) that fix every \( a \in A \) also fix \( x \) (i.e. \( \pi \cdot x = x \)).

\( X \) is a nominal set if every \( x \in X \) has a finite support.

Freshness, nominally, is a binary relation

\[ a \# x \triangleq a \notin A \text{ for some finite } A \text{ supporting } x. \]

satisfying \( \forall x. \exists a. a \# x \) (not Skolemizable!)
Name abstraction
Name abstraction

Each $X \in \text{Nom}$ yields a nominal set $[A]X$ of name-abstractions $\langle a \rangle x$ are $\sim$-equivalence classes of pairs $(a, x) \in A \times X$, where

$$(a, x) \sim (a', x') \iff \exists b \# (a, x, a', x')$$

$$(b \ a) \cdot x = (b \ a') \cdot x'$$

generalizes $\alpha$-equivalence from sets of syntax to arbitrary nominal sets

the permutation that swaps $a$ and $b$
Name abstraction

Each $X \in \text{Nom}$ yields a nominal set $[A]X$ of name-abstractions $\langle a \rangle x$ are $\sim$-equivalence classes of pairs $(a, x) \in A \times X$, where

$$(a, x) \sim (a', x') \iff \exists b \neq (a, x, a', x') \quad (b a) \cdot x = (b a') \cdot x'$$

Action of name permutations on $[A]X$ is well-defined by

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

and for this action, $A - \{a\}$ supports $\langle a \rangle x$ if $A$ supports $x$. 
If you want to know more about nominal sets...

**Nominal Sets**
*Names and Symmetry in Computer Science*

Cambridge Tracts in Theoretical Computer Science, Vol. 57  
(CUP, 2013)
Nom and dependent types
Families of nominal sets

Family over $X \in \text{Nom}$ is specified by:

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

\[
\prod_{\pi \in \text{Perm}} A \prod_{x \in X} (E_x \rightarrow E_{\pi \cdot x})
\]

with dependent version of finite support property:

for all $x \in X, e \in E_x$ there is a finite set $A$ of names supporting $x$ in $X$ and such that any $\pi$ fixing each $a \in A$ satisfies $\pi \cdot e = e \in E_{\pi \cdot x} = E_x$. 
Families of nominal sets

Family over $X \in \text{Nom}$ is specified by...

Get a category with families (cwf) [Dybjer] modelling extensional MLTT...

This cwf is relatively unexplored, so far.

But what’s it good for? Two possible applications:

1. higher-dimensional type theory
2. meta-programming/proving with name-binding structures
Bezem-Coquand-Huber cubical sets model of HoTT

(just the connection with the nominal sets notion of name abstraction)
One can view cubical sets as nominal sets $X$ equipped with some extra structure, whose names $a, b, c \ldots \in \mathbb{A}$ we think of as names of cartesian directions.
One can view cubical sets as nominal sets $X$ equipped with some extra structure, whose names $a, b, c \ldots \in \mathbb{A}$ we think of as names of cartesian directions.

$x \in X$

supported by

$\{a, b, c\}$
One can view cubical sets as nominal sets $X$ equipped with some extra structure, namely face maps

$$x \in X \mapsto (i/a)x \in X, \text{ for } i = 0, 1$$
One can view cubical sets as nominal sets $X$ equipped with some extra structure,

$$d_i : \left[ \forall A \right] X \rightarrow X$$

$$\langle a \rangle x \mapsto (i/a)x \quad (i \in 2)$$

satisfying

(binding: $a \not\# (i/a)x$ — follows from the type of $d_i$)

degeneracy: $a \not\# x \Rightarrow (i/a)x = x$

independence: $a \neq b \Rightarrow (i/a)(j/b)x = (j/b)(i/a)x$
\textbf{Theorem} (Staton). \textbf{Cub} is equivalent to the presheaf category \([\mathcal{C}, \text{Set}]\) originally used by Bezem, Coquand & Huber.

\(\mathcal{C}\) is [equivalent to] the category whose objects are finite ordinals and whose morphisms are given by:

\[
\mathcal{C}(m, n) = \{ f \in \text{Set}(m+2, n+2) \mid f 0 = 0 \land f 1 = 1 \land \\
\forall i, j > 1. f i = f j > 1 \Rightarrow i = j \}
\]
Name abstractions $\langle a \rangle x$ as paths (proofs of identity) from $(0/a)x$ to $(1/a)x$:

\[
\begin{array}{c}
\text{degenerate path} \\
\text{refl } x = \langle a \rangle x \\
\text{for some/any} \\
a \neq x
\end{array}
\]

Can these be the formation and introduction for an (intensional) identity type $Id_X$ for cubical set $X$?
Name abstractions $\langle a \rangle x$ as paths (proofs of identity) from $\langle 0/a \rangle x$ to $\langle 1/a \rangle x$:

$$[\forall] X \quad \langle a \rangle x$$

$$X \xrightarrow{\langle \text{id}, \text{id} \rangle} X \times X \quad ((0/a)x, (1/a)x)$$

Can these be the formation and introduction for an (intensional) identity type $\text{Id}_X$ for cubical set $X$?

Bezen-Coquand-Huber: yes (albeit with propositional eliminator), if we take the ‘fibrant’ families to be given by cubical sets satisfying a uniform Kan filling condition.
Name abstractions $\langle a \rangle x$ as paths (proofs of identity) from $(0/a)x$ to $(1/a)x$:

$$\begin{array}{c}
\text{[A]}X \\
\downarrow \langle d_0,d_1 \rangle \\
X \\
\downarrow \langle \text{id},\text{id} \rangle \\
X \times X
\end{array} \quad \begin{array}{c}
\langle a \rangle x \\
\downarrow \\
((0/a)x,(1/a)x)
\end{array}$$

Can these be the formation and introduction for an (intensional) identity type $\text{Id}_X$ for cubical set $X$?

Bezen-Coquand-Huber: yes (albeit with propositional eliminator), if we take the ‘fibrant’ families to be given by cubical sets satisfying a uniform Kan filling condition and one also gets a Voevodsky (univalent) universe.
Name abstractions $\langle a \rangle x$ as paths (proofs of identity) from $(0/a)x$ to $(1/a)x$:

$$
\begin{array}{ccc}
\mathbb{A} X & \xrightarrow{\text{refl}} & \langle a \rangle x \\
\downarrow & & \downarrow \\
X & \xrightarrow{\langle \text{id}, \text{id} \rangle} & X \times X (\langle 0/a \rangle x, \langle 1/a \rangle x)
\end{array}
$$

Can these be the formation and introduction for an (intensional) identity type $\text{Id}_X$ for cubical set $X$?

Bezen-Coquand-Huber: yes (albeit with propositional eliminator), if we take the ‘fibrant’ families to be given by cubical sets satisfying a uniform Kan filling condition and one also gets a Voevodsky (univalent) universe.

Why use Kan-$\textbf{Cub}$ rather than Kan-$\textbf{[C, Set]}$?

Variations on Kan filling? ‘Nominal’ simplicial sets?
Type Theory with names, freshness and name-abstraction

(joint work with Justus Matthiesen)
Families of nominal sets

Family over $X \in \text{Nom}$ is specified by...

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

nominal logic’s freshness quantifier Curry-Howard dependent name-abstraction

$\forall a. \varphi(a, \vec{x}) \iff [a \in A]E_a$
Families of nominal sets

Family over $X \in \text{Nom}$ is specified by . . .

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

nominal logic’s freshness quantifier

$\forall a. \varphi(a, \vec{x})$

Curry-Howard name-abstraction

$[a \in A]E_a$

\[= \exists a \# \vec{x}. \varphi(a, \vec{x})\]

\[= \forall a \# \vec{x}. \varphi(a, \vec{x})\]

‘some/any fresh $a$’
Original motivation for Gabbay & AMP to introduce nominal sets and name abstraction:

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

E.g. the initial algebra for \( A + (\_ \times \_) + [A](\_\_\_) \) is isomorphic to the usual set of untyped \( \lambda \)-terms.
Original motivation for Gabbay & AMP to introduce nominal sets and name abstraction.

Initial-algebra universal property $\Rightarrow$ recursion/induction principles for syntax involving name-binding operations [see JACM 53(2006)459-506].

- Exploited in impure functional programming language FreshML [Shinwell, Gabbay & AMP] – recursion only.

- Pure total (recursive) functions and proof (by induction): how to solve the analogy:

\[
\begin{align*}
\text{Coq} & \sim \text{OCaml} & \text{Agda} & \sim \text{Haskell} & \sim \text{FreshML}
\end{align*}
\]
Requirements for ‘FreshAgda’

- User-declared sorts of names (possibly with parameters) + user-defined inductive types, with name-abstraction types used to indicate binding constructs. E.g.

```agda
names Var : Set

data Term : Set where
  V : Var -> Term
  A : Term -> Term -> Term
  L : (V Term) -> Term

data Fresh(X : Set)(x : X) : Var -> Set where
  fr : [a : Var](Fresh X x a)
```
Requirements for ‘FreshAgda’

- User-declared sorts of names (possibly with parameters) + user-defined inductive types, with name-abstraction types used to indicate binding constructs. E.g.

```agda
names Var : Set

data Term : Set where
  V : Var -> Term
  A : Term -> Term -> Term
  L : ([Var]Term) -> Term

data Fresh(X : Set)(x : X) : Var -> Set where
  fr : [a : Var](Fresh X x a)
```

set of λ-terms mod α

set of proofs that a is fresh for x:X
Requirements for ‘FreshAgda’

- User-declared sorts of names (possibly with parameters) + user-defined inductive types, with name-abstraction types used to indicate binding constructs.
- Extend (dependent) pattern-matching with name-abstraction patterns. E.g.

```
_/_ : Term -> Var -> Term -> Term
(t/x)(V y)    = if x == y then t else V y
(t/x)(A t1 t2) = A ((t/x)t1) ((t/x)t2)
(t/x)(L <x>t1) = L <x> ((t/x)t1)
```

capture-avoiding substitution of t for x in t1
Requirements for ‘FreshAgda’

- User-declared sorts of names (possibly with parameters) + user-defined inductive types, with name-abstraction types used to indicate binding constructs.
- Extend (dependent) pattern-matching with name-abstraction patterns.

\[
_/\_ : \text{Term} \to \text{Var} \to \text{Term} \to \text{Term}
\]

\[
(t/x)(V\ y) = \text{if } x == y \text{ then } t \text{ else } V\ y
\]

\[
(t/x)(A\ t1\ t2) = A\ ((t/x)t1)\ ((t/x)t2)
\]

\[
(t/x)(L\ <x>t1) = L\ <x>\ ((t/x)t1)
\]

- Automatically respect \(\alpha\)-equivalence:
  
  FreshML uses impure generativity to ensure this. How to do it while maintaining Curry-Howard?
Fact: name abstraction functor

\[ [A](\_): \text{Nom} \rightarrow \text{Nom} \]

is right adjoint to ‘separated product’ functor

\( (\_)*A: \text{Nom} \rightarrow \text{Nom} \)

where

\[ X* A \triangleq \{(x,a) | a \# x\} \subseteq X \times A. \]
Fact: \([A](\_): \text{Nom} \rightarrow \text{Nom}\)

is right adjoint to ‘separated product’ functor

\((\_)*A: \text{Nom} \rightarrow \text{Nom}\)

Counit of the adjunction is ‘concretion’ of an abstraction

\(_\hat{\_}: ([A]X)*A \rightarrow X\)

defined by computation rule:

\((\langle a \rangle x)@b = (b \ a) \cdot x, \text{ if } b \neq \langle a \rangle x\)
Locally fresh names

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

\[ i : [A](X + Y) \cong [A]X + [A]Y \]
\[ i(z) = \text{fresh } a \text{ in case } z @ a \text{ of } \]
\[ \text{inl}(x) \rightarrow \langle a \rangle x \]
\[ \mid \text{inr}(y) \rightarrow \langle a \rangle y \]
Locally fresh names

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

\[ i : [\mathcal{A}](X + Y) \cong [\mathcal{A}]X + [\mathcal{A}]Y \]

\[ i(z) = \text{fresh } a \text{ in case } z @ a \text{ of } \]

\[ \text{inl}(x) \rightarrow \langle a \rangle x \]

\[ \text{inr}(y) \rightarrow \langle a \rangle y \]

Given \( f \in \text{Nom}(X \times \mathcal{A}, Y) \)

satisfying \( a \# x \Rightarrow a \# f(x, a) \),

we get \( \hat{f} \in \text{Nom}(X, Y) \) well-defined by:

\[ \hat{f}(x) = f(x, a) \text{ for some/any } a \# x. \]

Notation: \( \text{fresh } a \text{ in } f(x, a) \triangleq \hat{f}(x) \)
Locally fresh names

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

\[
i : [\mathcal{A}](X + Y) \cong [\mathcal{A}]X + [\mathcal{A}]Y
\]
\[
i(z) = \text{fresh } a \text{ in case } z @ a 	ext{ of }
\]
\[
\text{inl}(x) \rightarrow \langle a \rangle x
\]
\[
| \text{inr}(y) \rightarrow \langle a \rangle y
\]

\[
j : ([\mathcal{A}]X \rightarrow [\mathcal{A}]Y) \cong [\mathcal{A}](X \rightarrow Y)
\]
\[
j(f) = \text{fresh } a \text{ in }
\]
\[
\langle a \rangle(\lambda x. f(\langle a \rangle x) @ a)
\]

Can one turn the pseudocode into terms in a formal ‘nominal’ \(\lambda\)-calculus?
**Aim:** extend (dependently typed) $\lambda$-calculus with

- names $a$
- name swapping $\text{swap } a, b \text{ in } t$
- name abstraction $\langle a \rangle t$ and concretion $t @ a$
- locally fresh names $\text{fresh } a \text{ in } t$
- name equality $\text{if } t = a \text{ then } t_1 \text{ else } t_2$
**Aim:** extend (dependently typed) \( \lambda \)-calculus with

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

**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 (-)
- Crole-Nebel [MFPS 2013] – simple types (-), definitional freshness (+)
**Aim:** extend (dependently typed) $\lambda$-calculus with

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

**Prior art:**

- Stark-Schöpp [CSL 2004] – bunched contexts ($+$), extensional & undecidable ($-$)
- Westbrook-Stump-Austin [LFMTP 2009] CNIC – semantics/exp ressivity?
- Cheney [LMCS 2012] DNTT – bunched contexts ($+$), no local fresh names ($-$)
- Crole-Nebel [MFPS 2013] – simple types ($-$), definitional freshness ($+$)

We cherry pick, aiming for user-friendliness.
**Aim:** extend (dependently typed) $\lambda$-calculus with

names $a$

name swapping $\text{swap } a, b \text{ in } t$

name abstraction $\langle a \rangle t$ and concretion $t @ a$

locally fresh names $\text{fresh } a \text{ in } t$

name equality $\text{if } t = a \text{ then } t_1 \text{ else } t_2$

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

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

$$a \not\equiv x \Rightarrow a \not\equiv f(x, a)$$
Definitional freshness

In a nominal set of (higher-order) functions, proving $a \neq f$ 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 $a \not\equiv f$ 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 $b \not\equiv f$, we also get $a = (a \ b) \cdot b \not\equiv (a \ b) \cdot f = f$, QED.
Definitional freshness

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

\[
\begin{align*}
\Gamma \vdash a \# T & \quad \Gamma \vdash t : T \\
\Gamma#(b : A) \vdash (\text{swap } a, b \text{ in } t) = t : T \\
\hline
\Gamma \vdash a \# t : T
\end{align*}
\]

bunched contexts, generated by
\[
\begin{align*}
\Gamma & \mapsto \Gamma(x : T) \\
\Gamma & \mapsto \Gamma#(a : A)
\end{align*}
\]
definitional freshness
definitional equality
Definitional freshness

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

$$
\begin{align*}
\Gamma \vdash a \# T & \quad \Gamma \vdash t : T \\
\Gamma#(b : A) \vdash (\text{swap } a, b \text{ in } t) = t : T \\
\hline
\Gamma \vdash a \# t : T
\end{align*}
$$

Freshness info in bunched contexts gets used via:

$$
\begin{align*}
\Gamma(x : T)\Gamma' \text{ ok} & \quad a, b \in \Gamma' \\
\hline
\Gamma(x : T)\Gamma' \vdash (\text{swap } a, b \text{ in } x) = x : T
\end{align*}
$$
In a nominal set of (higher-order) functions, proving $a \not\equiv f$ can be tricky (undecidable). Common proof pattern:

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

$$\Gamma\#(b : A) \vdash (\text{swap } a, b \text{ in } t) = t : T$$

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

definitional freshness for types:

$$\Gamma \vdash T \quad a \in \Gamma$$

$$\Gamma\#(b : A) \vdash (\text{swap } a, b \text{ in } T) = T$$

$$\Gamma \vdash a \# T$$
A type theory
To do

- Decidability of typing & definitional equality judgements (normal forms and algorithmic version of the type system).
- Inductively defined types involving $[a : A](\_)$ (e.g. propositional freshness & nominal logic).
- Dependently typed pattern-matching with name-abstraction patterns.
- Implementation.
Conclusions

1. **Nom** vs **Sch**, **Cub** vs **[C, Set]**: names are convenient! (because unlike indexes, they survive weakening).

2. Possibility of a ‘nominal’ treatment of dimensions in higher-dimensional type theory & category theory seems intriguing: e.g. what are $\infty$-groupoids when $\infty = \text{finitely inexhaustible}$?

3. Nominal sets notion of implicit dependence does not sit easily with explicit functional dependence in type theory. (Permutations are mathematically pleasant, but not computationally pleasant?)