Dependent Type Theory with Abstractable Names

Andrew Pitts

UNIVERSITY OF CAMBRIDGE
Computer Laboratory
Aim

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

Motivation:

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

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

- Nominal sets
- Motivation for a ‘nominal’ MLTT
- Prior art
- Definitional freshness
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 semantics, with ‘possible worlds’ as follows. . .
\( \mathbb{I} = \text{category of finite ordinals} \)
\[
n = \{0, 1, \ldots, n-1\}
\]
and injective functions

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

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

\([\mathbb{I}, \mathbf{Set}] = \text{(covariant) presheaf category: set-valued functors } \mathbf{X} \text{ \& natural transformations.}\)
\[\mathbf{X}^n = \text{set of objects (of some type) possibly involving } n \text{ distinct names}\]
\[ \mathbb{I} = \text{category of finite ordinals} \]
\[ n = \{0, 1, \ldots, n - 1\} \]
and injective functions

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

generic decidable object
\[ U = \text{inclusion functor:} \]
\[ U_n = \{0, 1, \ldots, n - 1\} \]

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

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

The diagonal subobject $U \hookrightarrow U \times U$ has a boolean complement $\not\hookrightarrow U \times U$. 
Generic decidable object

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

\[
a =_\mathbb{U} b \land a \neq b \Rightarrow \text{false}
\]

\[
\text{true} \Rightarrow a =_\mathbb{U} b \lor a \neq b
\]
Generic infinite decidable object

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

\[
\begin{align*}
 a & =_U b \land a \neq b \Rightarrow \text{false} \\
 \text{true} & \Rightarrow a =_U b \lor a \neq b
\end{align*}
\]

but it does not satisfy ‘finite inexhaustibility’

\[
\begin{align*}
 \land_{0 \leq i < j \leq n} a_i & \neq a_j \Rightarrow \lor_{b:U} \land_{0 \leq i \leq n} b \neq a_i
\end{align*}
\]

which we need to model freshness.
Generic infinite decidable object

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

\[
\begin{align*}
& a =_\mathbb{U} b \land a \neq b \Rightarrow \text{false} \\
& \text{true} \Rightarrow a =_\mathbb{U} b \lor a \neq b
\end{align*}
\]

but it does not satisfy ‘finite inexhaustibility’

\[
\bigwedge_{0 \leq i < j \leq n} a_i \neq a_j \Rightarrow \bigvee_{b : \mathbb{U}} \bigwedge_{0 \leq i \leq n} b \neq a_i
\]

**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
From \textbf{Sch} to \textbf{Nom}

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

An analogy:

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

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

The category of nominal sets **Nom** is ‘merely’ an equivalent presentation of the category **Sch**:

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

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

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

(cf. homogeneity property (Fraïssé limit))
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’ $A \in \mathcal{P}_{\text{fin}} A$ implicit by taking the colimit $\tilde{X}$ of the directed system of sets and (injective) functions

$$A \subseteq B \in \mathcal{P}_{\text{fin}} A \mapsto (X A \rightarrow XB)$$

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

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

**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

$$a \not\in x \triangleq a \notin A$$

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 \( A \subseteq \mathbb{A} \) 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

\[
\forall x. \exists a. \ a \notin A \quad \triangleq \quad a \# x
\]

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

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

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

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

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

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 $x$. 
Fact: name abstraction functor

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

is right adjoint to ‘separated product’ functor

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

where \( X* A \triangleq \{(x, a) \mid a \# x\} \subseteq X \times A \).
so $[A]X$ is a kind of (affine) function space (with a right adjoint!)

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

is right adjoint to ‘separated product’ functor

$(\_) \ast A: \text{Nom} \to \text{Nom}$

Co-unit of the adjunction is ‘concretion’ of an abstraction

$\_ \circ \_ : ([A]X) \ast A \to X$

defined by computation rule:

$(\langle a \rangle x) \circ b = (b \ a) \cdot x$, if $b \neq \langle a \rangle x$
If you want to know more about nominal sets...

Nominal Sets
Names and Symmetry in Computer Science

Nom and dependent types
Families of nominal sets

A family over \( X \in \text{Nom} \) is specified by:

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

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

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

\[
\pi \cdot e = e \\
\bigcap A = \bigcap A \\
E_{\pi \cdot x} = E_x
\]
Families of nominal sets

A 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 [Schöpp’s PhD, mainly]. What’s it good for?

I’m interested in two applications:

- meta-programming/proving with name-binding structures [this talk]
- higher-dimensional type theory (HoTT) [not this talk]
Type Theory with names, freshness and name-abstraction

(joint work with Justus Matthiesen)
Original motivation for Gabbay & AMP to introduce nominal sets and name abstraction:

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

E.g. the initial algebra for \( \mathcal{A} + (\_ \times \_ ) + [\mathcal{A}](\_ ) \) 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:

\[
\begin{array}{ccc}
\text{Coq} & \sim & \text{Agda} & \sim & ? \\
\text{OCaml} & \sim & \text{Haskell} & \sim & \text{FreshML}
\end{array}
\]
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)
```
User-declared sorts of names (possibly with parameters) + user-defined inductive types, with name-abstraction types used to indicate binding constructs. E.g.

```
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
Families of nominal sets

A 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

A 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
  \[ \forall a. \varphi(a, \vec{x}) \]
  \[ \equiv \exists a \neq \vec{x}. \varphi(a, \vec{x}) \]
  \[ \equiv \forall a \neq \vec{x}. \varphi(a, \vec{x}) \]
  ‘some/any fresh $a$’

- Curry-Howard name-abstraction
  \[ \exists a \in A \] \(E_a\)
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)
```

Do inductive definitions with constructor arities like this make sense?
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.

\[
_/\_ : \text{Term} \rightarrow \text{Var} \rightarrow \text{Term} \rightarrow \text{Term}\\
(t/x)(V\ y) = \text{if}\ x == y\ \text{then}\ t\ \text{else}\ V\ y\\
(t/x)(A\ t1\ t2) = A\ ((t/x)t1)\ ((t/x)t2)\\
(t/x)(L\ <x>t1) = L\ <x>((t/x)t1)
\]

*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

\[ (_/_) : \text{Term} \to \text{Var} \to \text{Term} \to \text{Term} \]
\[ (t/x)(V\ y) = \text{if } x == y \text{ then } t \text{ else } V\ y \]
\[ (t/x)(A\ t1\ t2) = A\ ((t/x)t1)\ ((t/x)t2) \]
\[ (t/x)(L\ <x>t1) = L\ <x>\ ((t/x)t1) \]

that automatically respect \( \alpha \)-equivalence:

FreshML uses impure generativity to ensure this.

How to do it while maintaining Curry-Howard?
Locally fresh names

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

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

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

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

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

\[
\begin{align*}
\text{inl}(x) & \rightarrow \langle a \rangle x \\
\text{inr}(y) & \rightarrow \langle a \rangle y
\end{align*}
\]

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

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:

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

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

Can one turn the pseudocode into terms in a formal ‘nominal’ \( \lambda \)-calculus?
Aim

A version of Martin-Löf Type Theory enriched with constructs for freshness 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) $\lambda$-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 (−)

- Fairweather-Fernández-Szasz-Tasistro [2012]
  based on nominal terms (+), explicit substitutions (−), first-order (±)

- Crole-Nebel [MFPS 2013]
  simple types (−), definitional freshness (+)
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 (+)

We cherry pick, **aiming for user-friendliness**.
Aim

More specifically: 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 $a \ # x \Rightarrow a \ # f(x, a)$
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 $a \not\approx f$ 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 $b \not\approx f$, we also get $a = (a \ b) \cdot b \not\approx (a \ b) \cdot f = f$, QED.
Definitional freshness

Γ ⊢ a # T  Γ ⊢ t : T
Γ#(b : A) ⊢ (swap a, b in t) = t : T
Γ ⊢ a # t : T

bunched contexts, generated by
Γ ↦ Γ(x : T)
Γ ↦ Γ#(a : A)

definitional freshness

definitional equality
Definitional freshness

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

Freshness info in bunched contexts gets used via:

\[
\Gamma(x : T)\Gamma' \text{ ok } \quad a,b \in \Gamma' \\
\Gamma(x : T)\Gamma' \vdash (\text{swap } a,b \text{ in } x) = x : T
\]
Definitional freshness

\[ \Gamma \vdash a \not\in T \quad \Gamma \vdash t : T \]
\[ \Gamma\#(b : A) \vdash (\text{swap } a, b \text{ in } t) = t : T \]
\[ \Gamma \vdash a \not\in t : T \]

definitional freshness for types:
\[ \Gamma \vdash T \quad a \in \Gamma \]
\[ \Gamma\#(b : A) \vdash (\text{swap } a, b \text{ in } T) = T \]
\[ \Gamma \vdash a \not\in T \]
A type theory

\[ \Gamma \vdash a \in \text{Type} \]

\[ \Gamma \vdash \text{let } x \rightarrow t : \text{Type} \]

\[ \Gamma \vdash a \in \text{Type} \]

\[ \Gamma \vdash e : T \]

\[ \Gamma \vdash (\lambda x : T). e \]
A type theory
To do

- Decidability of typing & definitional equality judgements: normal forms and algorithmic version of the type system and hence...
- ...an implementation.
- Dependently typed pattern-matching with name-abstraction patterns.
- Inductively defined types involving $[a : A](\_)$ (e.g. propositional freshness & nominal logic). Maybe definitional freshness is too weak (cf. experience with FreshML2000)?