LSFA 2014

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...

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

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

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

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

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

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

generic decidable object
U = inclusion functor:
$\mathsf{U}\, n = \{0, 1, \ldots, n-1\}$

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

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

Generic decidable object

$\mathbf{U}$ is a 'decidable' object of the presheaf topos $[\mathbb{I}, \mathbf{Set}]$

diagonal subobject $\mathbf{U} \rightarrowtail \mathbf{U} \times \mathbf{U}$ has a boolean complement $\neq \rightarrowtail \mathbf{U} \times \mathbf{U}$

Generic decidable object

$\mathbf{U}$ is a 'decidable' object of the presheaf topos $[\mathbb{I}, \mathbf{Set}]$

$$a =_{\mathsf{U}} b \;\wedge\; a \neq b \;\Rightarrow\; \mathbf{false}$$
$$\mathbf{true} \;\Rightarrow\; a =_{\mathsf{U}} b \;\vee\; a \neq b$$

Generic infinite decidable object

U is a 'decidable' object of the presheaf topos $[\mathbb{I}, \mathbf{Set}]$

$$a =_{\mathsf{U}} b \;\wedge\; a \neq b \;\Rightarrow\; \mathbf{false}$$
$$\mathbf{true} \;\Rightarrow\; a =_{\mathsf{U}} b \;\vee\; a \neq b$$

but it does not satisfy 'finite inexhaustibility'

$$\bigwedge_{0 \leq i < j \leq n} a_i \neq a_j \;\Rightarrow\; \bigvee_{b:\mathsf{U}} \bigwedge_{0 \leq i \leq n} b \neq a_i$$

which we need to model freshness.

Generic infinite decidable object

U is a 'decidable' object of the presheaf topos $[\mathbb{I}, \mathbf{Set}]$

$$a =_{\mathsf{U}} b \;\wedge\; a \neq b \;\Rightarrow\; \mathbf{false}$$
$$\mathbf{true} \;\Rightarrow\; a =_{\mathsf{U}} b \;\vee\; a \neq b$$

but it does not satisfy 'finite inexhaustibility'

$$\bigwedge_{0 \leq i < j \leq n} a_i \neq a_j \;\Rightarrow\; \bigvee_{b:\mathsf{U}} \bigwedge_{0 \leq i \leq n} b \neq a_i$$

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

$\mathbf{Sch} \subseteq [\mathbb{I}, \mathbf{Set}]$ is the full subcategory consisting of functors $\mathbb{I} \longrightarrow \mathbf{Set}$ that preserve pullbacks

From $\mathbf{Sch}$ to $\mathbf{Nom}$

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

An analogy:

$$\frac{\mathbf{Nom}}{\mathbf{Sch}} \sim \frac{\text{named bound variables}}{\text{de Bruijn indexes (levels)}}$$

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

From **Sch** to **Nom**

The category of nominal sets $\boxed{\mathbf{Nom}}$ is 'merely' an equivalent presentation of the category $\mathbf{Sch}$:

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

$$A \subseteq B \in \mathbf{P_{fin}}\,\mathbb{A} \;\mapsto\; (X\,A \to X\,B)$$

Each set $\tilde{X}$ carries an action of $\mathbb{A}$-permutations

(cf. homogeneity property (Fraïssé limit) $\boxed{\begin{tikzcd} \mathbb{A} \arrow[r, dashed, "\cong"] & \mathbb{A} \\ A \arrow[u, hook] \arrow[r, tail, "f"'] & B \arrow[u, hook] \end{tikzcd}}$)

From $\mathbf{Sch}$ to $\mathbf{Nom}$

The category of nominal sets $\boxed{\mathbf{Nom}}$ is 'merely' an equivalent presentation of the category $\mathbf{Sch}$:

Step 2: make the dependence of each $X \in \mathbf{Sch}$ on 'possible worlds' $A \in \mathbf{P}_{\mathbf{fin}}\,\mathbb{A}$ implicit by taking the colimit $\tilde{X}$ of the directed system of sets and (injective) functions

$$A \subseteq B \in \mathbf{P}_{\mathbf{fin}}\,\mathbb{A} \;\mapsto (X\,A \to X\,B)$$

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

Finite support property

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

Finite support property

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

A set of names $A \subseteq \mathbb{A}$ supports x if permutations π 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 $\mathbf{Perm}\,\mathbb{A}$ (= group of all (finite) permutations of $\mathbb{A}$) acts on a set X and that $x \in X$.

A set of names $A \subseteq \mathbb{A}$ supports x if permutations π 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.

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

FACT: $\mathbf{Nom}$ and $\mathbf{Sch}$ are equivalent categories.

Within $\mathbf{Nom}$, objects are 'set-like' and the modal character of freshness becomes implicit...

Finite support property

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

A set of names $A \subseteq \mathbb{A}$ supports x if permutations π 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 <u>finite</u> support.

Freshness, nominally, is a binary relation

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

'name a is fresh for x'

Finite support property

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

A set of names $A \subseteq \mathbb{A}$ supports x if permutations π 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 <u>finite</u> support.

Freshness, nominally, is a binary relation

$$a \mathbin{\#} x \triangleq a \notin A \text{ for some finite } A \text{ supporting } x.$$

satisfying $\boxed{\forall x. \exists a.\ a \mathbin{\#} x}$ (not Skolemizable!)

Name abstraction

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

$$(a, x) \sim (a', x') \Leftrightarrow \exists b \# (a, x, a', x') \; (b\ a) \cdot x = (b\ a') \cdot x'$$

generalizes α-equivalence from sets of syntax to arbitrary nominal sets

the permutation that swaps a and b

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

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

Action of name permutations on $[\mathbb{A}]X$ 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

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

is right adjoint to 'separated product' functor

$$(\_) * \mathbb{A} : \mathbf{Nom} \to \mathbf{Nom}$$

where $\boxed{X * \mathbb{A} \triangleq \{(x, a) \mid a \mathbin{\#} x\}} \subseteq X \times \mathbb{A}$.

so $[\mathbb{A}]X$ is a kind of (affine) function space
(with a right adjoint!)

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

is right adjoint to 'separated product' functor

$$(\_) * \mathbb{A} : \mathbf{Nom} \to \mathbf{Nom}$$

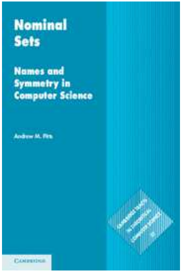
Co-unit of the adjunction is 'concretion' of an abstraction

$$\_ @ \_ : ([\mathbb{A}]X) * \mathbb{A} \to X$$

defined by computation rule:

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

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

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

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

$$\prod_{\pi \in \mathbf{Perm}\,\mathbb{A}} \prod_{x \in X} (E_x \to E_{\pi \cdot x})$$

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 π fixing each $a \in A$ satisfies

$$\begin{array}{ccc} \pi \cdot e & = & e \\ \rotatebox{90}{$\in$} & & \rotatebox{90}{$\in$} \\ E_{\pi \cdot x} & = & E_x \end{array}$$

Families of nominal sets

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

> $[\mathbb{A}](\_)$ 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.

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:

$$\frac{\textbf{Coq}}{\textbf{OCaml}} \sim \frac{\textbf{Agda}}{\textbf{Haskell}} \sim \frac{\textbf{?}}{\textbf{FreshML}}$$

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.

```
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)
```

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.

set of λ-terms mod α

```
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 proofs that a is fresh for `x : X`

Families of nominal sets

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
$\mathsf{N}a.\, \varphi(a, \vec{x})$	$\longleftrightarrow$	$[a \in \mathbb{A}]E_a$

Families of nominal sets

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

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

```
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.

```
_/_ :  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

```
_/_ :  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)
```

that automatically respect α-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:

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

Locally fresh names

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

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

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: $\boxed{\textbf{fresh}\ a\ \textbf{in}\ f(x,a)} \triangleq \hat{f}(x)$

Locally fresh names

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

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

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

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) 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 $\texttt{if}\ t = a\ \texttt{then}\ t_1\ \texttt{else}\ t_2$

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 $(\pm)$
- Crole-Nebel [MFPS 2013]

  simple types $(-)$, definitional freshness $(+)$

We cherry pick, aiming for user-friendliness.

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 $\texttt{if}\ t = a\ \texttt{then}\ t_1\ \texttt{else}\ t_2$

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

Definitional freshness

In a nominal set of (higher-order) functions, proving $a \mathrel{\#} f$ 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

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

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

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

Freshness info in bunched contexts gets used via:

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

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

$\boxed{\Gamma \vdash}$

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

$\boxed{\Gamma \vdash T}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A type theory

OMITTED

$\boxed{\Gamma \vdash}$

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

$\boxed{\Gamma \vdash T}$

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

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

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

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

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

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

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

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

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

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

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 $[a : \mathbb{A}](\_)$ (e.g. propositional freshness & nominal logic). Maybe definitional freshness is too weak (cf. experience with FreshML2000)?