Local names

- **Local variables** in Algol-like languages:
  \[
  \text{new } X \text{ in } \langle \text{command} \rangle
  \]

- **Generativity** + local declarations in ML-like languages:
  \[
  \text{let } x = \text{ref}\langle \text{val} \rangle \text{ in } \langle \text{exp} \rangle
  \]

- **Channel-name restriction** in \(\pi\)-like process calculi:
  \[
  (\nu a)\langle \text{process} \rangle
  \]

- **Use of fresh names** in meta-programming/reasoning, e.g.
  \[
  \text{A-nf}(e_1 e_2) \triangleq \text{let } v_1 = e_1, v_2 = e_2 \text{ in } v_1 v_2 \\
  \text{where } v_1 \ v_2 \text{ are fresh}
  \]
Local names

- Local variables in Algol-like languages:
  \[ \text{new } X \text{ in } \langle \text{command} \rangle \]

- Generativity + local declarations in ML-like languages:
  \[ \text{let } x = \text{ref}<\text{val} \text{ in } \langle \text{exp} \rangle \]

- Channel-name restriction in \( \pi \)-like process calculi:
  \[ (\nu a)\langle \text{process} \rangle \]

- Use of fresh names in meta-programming/reasoning, e.g.
  \[
  \text{A-nf}(e_1 e_2) \triangleq \text{let } v_1 = e_1, v_2 = e_2 \text{ in } v_1 v_2
  \]
  \[
  \text{where } v_1 v_2 \text{ are fresh}
  \]

What is the mathematical foundation for these locality constructs? Is it the same in each case?
What is the mathematical foundation for these locality constructs? Is it the same in each case?

I’ve had a 20+ year interest (obsession?) in such questions, aided by

James Cheney, Ranald Clouston, Roy Crole, Jasper Derikx, Dan Ghica, Marcelo Fiore, Jamie Gabbay, Matthew Hennessy, Matt Lakin, Steffen Lösch, Justus Matthiesen, Frank Nebel, Eike Ritter, Uli Schöpp, Mark Shinwell, Ian Stark, Sam Staton, Alley Stoughton, Christian Urban, . . .
What is the mathematical foundation for these locality constructs? Is it the same in each case?

I’ve had a 20+ year interest (obsession?) in such questions, aided by

James Cheney, Ranald Clouston, Roy Crole, Jasper Derikx, Dan Ghica, Marcelo Fiore, Jamie Gabbay, Matthew Hennessy, Matt Lakin, Steffen Lösch, Justus Matthiesien, Frank Nebel, Eike Ritter, Uli Schöpp, Mark Shinwell, Ian Stark, Sam Staton, Alley Stoughton, Christian Urban, . . .

What’s new?
What is the mathematical foundation for these locality constructs? Is it the same in each case?

What’s new:

Integration with dependent type theory (DTT)

Motivation: programming or proving?
What is the mathematical foundation for these locality constructs? Is it the same in each case?

What’s new:

Integration with dependent type theory (DTT)

Motivation: programming or proving.

- DTT with generative local names
- Nominal techniques
- Judgemental freshness
DTT with generative names
DTT judgements \( \{ \) typing \( T \) type \( t : T \) 
\( T = T' \) \( t = t' : T \) \( \) equality \( \}\) 
are intertwined:

\[
\begin{align*}
(x : T_1) & \vdash T_2(x) \text{ type} \quad t = t' : T_1 \\
T_2(t) & = T_2(t')
\end{align*}
\]

\[
\begin{align*}
t : T_1 & \quad T_1 = T_2 \\
t : T_2
\end{align*}
\]
DTT judgements \[ \{ \]

\begin{align*}
\text{typing} & \quad T \text{ type} \quad t : T \\
\text{equality} & \quad T = T' \quad t = t' : T
\end{align*}

If we allow generative, locally scoped names \( \nu x. t(x) \), what are the rules for (decidable) equality judgements?
DTT judgements

\[
\begin{align*}
& \text{typing} & T & \text{type} & t : T \\
& \text{equality} & T = T' & t = t' : T
\end{align*}
\]

If we allow \textit{generative, locally scoped names} \( \nu x. t(x) \), what are the rules for (decidable) equality judgements?

Familiar generative dynamics of locally scoped names

\[(\nu x. t(x), \text{state}) \rightarrow (t(a), \text{state} \uplus \{a\})\]
DTT judgements \[
\{ \text{typing} \quad T \text{ type} \quad t : T \\
\text{equality} \quad T = T' \quad t = t' : T \}
\]

If we allow generative, locally scoped names \( \nu x. t(x) \), what are the rules for (decidable) equality judgements?

Familiar generative dynamics of locally scoped names

\[
(\nu x. t(x), \text{state}) \rightarrow (t(a), \text{state} \cup \{a\})
\]

can be reformulated with evaluation contexts \( E[\_] \)

\[
E[\nu x. t(x)] \rightarrow \nu x. E[t(x)]
\]

to answer this question.
DTT judgements \{ 
\begin{align*}
&\text{typing} \quad T \, \text{type} \quad t : T \\
&\text{equality} \quad T = T' \quad t = t' : T
\end{align*}
\}

If we allow generative, locally scoped names $\nu x. t(x)$, what are the rules for (decidable) typing judgements?
Typing generative local names

\[
\frac{T \text{ type} \quad (x : \text{Name}) \vdash t(x) : T}{\nu x. t(x) : T}
\]

is safe, but inexpressive – seems inevitable that type expressions as well as term expressions may involve name generation:

\[
\frac{(x : \text{Name}) \vdash t(x) : T(x)}{\nu x. t(x) : \nu x. T(x)}
\]
Typing generative local names

\[
T \text{ type} \quad (x : \text{Name}) \vdash t(x) : T \\
\nu x. t(x) : T
\]

is safe, but inexpressive – seems inevitable that type expressions as well as term expressions may involve name generation:

\[
(x : \text{Name}) \vdash t(x) : T(x) \\
\nu y. t(y) : \nu z. T(z)
\]

If \( t \) & \( T \) are both generative, what does \( t : T \) mean? Are there models to guide us?
Nominal techniques
Fix countably infinite set $A$ (elements $a, b, c, \ldots$ called atoms).

Nominal set $= D + \left( (\_\_ \_ \_ : A \to A \to D \to D) \right)$ + finite supports

Morphism of nominal sets $=$

function that commutes with atom swapping
Nominal sets overview

Fix countably infinite set $\mathbb{A}$ (elements $a, b, c, \ldots$ called atoms).

Nominal set =

$$\text{set } D = \left( \begin{array}{c}
\text{atom-swapping function} \\
(\_ \_ \_ ) \cdot \_ : \mathbb{A} \rightarrow \mathbb{A} \rightarrow D \rightarrow D \\
\text{with simple algebraic properties}
\end{array} \right) + \text{finite supports}$$

Morphism of nominal sets =

function that commutes with atom swapping

for every $d \in D$ there is a finite list of names $S_d \in \text{List } \mathbb{A}$ satisfying $(\forall a, b \notin S_d) (a \ b) \cdot d = d$

($S_d$ lists the names that $d$ may involve. In this talk I will try to be constructive, so no use of least support sets.)
Nominal sets overview

Fix countably infinite set $A$ (elements $a, b, c, \ldots$ called atoms).

Nominal set =

$$\text{set } D + \left( \begin{array}{c} \text{atom-swapping function} \\
\left( \_ \_ \right) \cdot \_ : A \to A \to D \to D \\
\text{with simple algebraic properties} \end{array} \right) + \text{finite supports}$$

Morphism of nominal sets =

function that commutes with atom swapping

If you want to know more, then read
Families of nominal sets

Equivalent presentation of slice categories $\text{Nom}/D$
making pullbacks associate ‘on the nose’:
 Equivalent presentation of slice categories \( \text{Nom}\!/D \) making pullbacks associate ‘on the nose’:

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

- \( D \)-indexed family of sets \( (E_d \mid d \in D) \)
- dependently typed atom-swapping

\[
(a \ b) \cdot _d : E_d \to E_{(a \ b) \cdot d}
\]

with dependent version of finite support property.

Get a category with families (CwF) [Dybjer, 1996] modelling extensional MLTT...
DTT

<table>
<thead>
<tr>
<th>contexts</th>
<th>[ \Gamma \vdash D \in \text{Nom} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>types</td>
<td>[ \Gamma \vdash T \text{ type} ]</td>
</tr>
<tr>
<td>terms</td>
<td>[ \Gamma \vdash t : T ]</td>
</tr>
</tbody>
</table>

CwF Nom

<table>
<thead>
<tr>
<th>objects</th>
<th>[ D \in \text{Nom} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>families</td>
<td>[ \left( \sum_{d \in D} E_d \right) \in \text{Nom}/D ]</td>
</tr>
<tr>
<td>global sections</td>
<td>[ \left( D \rightarrow \sum_{d \in D} E_d \right) \in \text{Nom}/D ]</td>
</tr>
</tbody>
</table>

See [M. Hofmann, *Syntax and Semantics of Dependent Types*, 1997].
Judgemental freshness
Local names

- **Local variables** in Algol-like languages:
  \[ \text{new } X \text{ in } \langle \text{command} \rangle \]

- **Generativity** + local declarations in ML-like languages:
  \[ \text{let } x = \text{ref } \langle \text{val} \rangle \text{ in } \langle \text{exp} \rangle \]

- **Channel-name restriction** in \( \pi \)-like process calculi:
  \( (\nu a) \langle \text{process} \rangle \)

- **Use of fresh names** in meta-programming/reasoning, e.g.

  \[ \text{A-nf}(e_1 e_2) \triangleq \text{let } v_1 = e_1, v_2 = e_2 \text{ in } v_1 v_2 \]
  \[ \text{where } v_1 v_2 \text{ are fresh} \]

  Thesis: fresh names in metaprogramming/reasoning are always used in way that is **semantically trivial**.
Semantic freshness

freshness relation \(_\#\_\subseteq A \times D\) (for \(D \in \text{Nom}\))

\[a \# d \triangleq (\exists b \notin S_d) (a \ b) \cdot d = d \]
\[\iff (\forall b \notin S_d) (a \ b) \cdot d = d\]

provides a syntax-independent notion of freeness/non-occurrence
Semantic freshness

freshness relation $\_ \# \_ \subseteq A \times D$ (for $D \in \text{Nom}$)

$$a \# d \triangleq (\exists b \notin S_d) (a \ b) \cdot d = d$$

$$\iff (\forall b \notin S_d) (a \ b) \cdot d = d$$

If $d$ is described by a term $t$ with no free variables, then \{free atoms of $t$\} will do for $S_d$. 
Semantic freshness

freshness relation \( _\# _ \subseteq A \times D \) (for \( D \in \text{Nom} \))

\[
a \# d \triangleq (\exists b \notin S_d) (a \ b) \cdot d = d
\]

\[
\iff (\forall b \notin S_d) (a \ b) \cdot d = d
\]

If \( d \) is described by a term \( t \) with no free variables, then \( \{\text{free atoms of } t\} \) will do for \( S_d \).

But what if \( t \) does have free variables?
<table>
<thead>
<tr>
<th>DTT</th>
<th>CwF <strong>Nom</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>extend context with a variable</td>
<td>dependent product</td>
</tr>
<tr>
<td>$\Gamma \vdash T \text{ type}$</td>
<td>$(E_d \mid d \in D)$</td>
</tr>
<tr>
<td>$\Gamma(x : T) \vdash (x \notin \Gamma)$</td>
<td>$\sum_{d \in D} E_d$</td>
</tr>
</tbody>
</table>
**Bunched contexts**

<table>
<thead>
<tr>
<th>DTT+names</th>
<th>CwF Nom</th>
</tr>
</thead>
<tbody>
<tr>
<td>extend context with a variable</td>
<td></td>
</tr>
<tr>
<td>[ \Gamma \vdash T \text{ type} ]</td>
<td>[ (E_d \mid d \in D) ]</td>
</tr>
<tr>
<td>[ \Gamma(x : T) \vdash(x \notin \Gamma) ]</td>
<td>[ \sum_{d \in D} E_d ]</td>
</tr>
<tr>
<td>extend context with a fresh name</td>
<td></td>
</tr>
<tr>
<td>[ \Gamma \vdash ]</td>
<td>[ D ]</td>
</tr>
<tr>
<td>[ \Gamma[a : \text{Name}] \vdash(a \notin \Gamma) ]</td>
<td>[ D \otimes A ]</td>
</tr>
</tbody>
</table>

In DTT+names:

- *Name* is a type of names,
- \([Name] = A\) (nominal set of atoms),
- *variables* \(x\) and *atoms* \(a\) are disjoint classes of identifier.
Bunched contexts

<table>
<thead>
<tr>
<th><strong>DTT+names</strong></th>
<th><strong>CwF Nom</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>extend context with a variable</td>
<td>dependent product</td>
</tr>
<tr>
<td>[ \Gamma \vdash T \text{ type} ]</td>
<td>[ (E_d \mid d \in D) ]</td>
</tr>
<tr>
<td>[ \Gamma(x : T) \vdash (x \notin \Gamma) ]</td>
<td>[ \sum_{d \in D} E_d ]</td>
</tr>
<tr>
<td>extend context with a fresh name</td>
<td>separated product</td>
</tr>
<tr>
<td>[ \Gamma \vdash ]</td>
<td>[ D ]</td>
</tr>
<tr>
<td>[ \Gamma[a : \text{Name}] \vdash (a \notin \Gamma) ]</td>
<td>[ D \otimes A ]</td>
</tr>
</tbody>
</table>

\[ \{(d, a) \in D \times A \mid a \neq d\} \]

See [Stark-Schöpp, CSL 2004][Cheney, LMCS 2012].
Judgemental freshness

Judgemental freshness is derivable from judgemental equality

cf. [Clouston, LFMTP 2011] and [Crole-Nebel, MFPS 2013]

\[
\begin{align*}
\Gamma \vdash a : \text{Name} & \quad \Gamma \vdash t : T \\
\Gamma[b : \text{Name}] \vdash (\text{swap } a, b \text{ in } t) = t : T & \\
\Gamma \vdash a \# t : T & \quad (b \not\in \Gamma)
\end{align*}
\]

(\text{swap } a, b \text{ in } t \text{ is an } \text{explicit swapping expression})
Judgemental freshness

Judgemental freshness is derivable from judgemental equality
cf. [Clouston, LFMTP 2011] and [Crole-Nebel, MFPS 2013]

\[
\begin{array}{c}
\Gamma \vdash a \not\equiv T \quad \Gamma \vdash t : T \\
\Gamma[b : Name] \vdash (\text{swap } a, b \text{ in } t) = t : T \\
\hline
\Gamma \vdash a \not\equiv t : T \\
\end{array}
\] 

\(b \not\in \Gamma\)

\[
\begin{array}{c}
\Gamma \vdash a : Name \quad \Gamma \vdash T \\
\Gamma[b : Name] \vdash (\text{swap } a, b \text{ in } T) = T \\
\hline
\Gamma \vdash a \not\equiv T \\
\end{array}
\] 

\(b \not\in \Gamma\)
syntactic \subset \neq \text{judgemental} \subset \neq \text{semantic freshness}
$\text{syntactic freshness} \subset \text{judgemental freshness} \subset \text{semantic freshness}$

\begin{equation*}
\red{a \text{ occurs in if } a = b \text{ then } a \text{ else } b, \text{ but}} \\
[a \ b : \text{Name}] \vdash a \# (\text{if } a = b \text{ then } a \text{ else } b) : \text{Name}
\end{equation*}
syntactic freshness $\subseteq$ judgemental freshness $\not\equiv$ semantic freshness

\[ a \text{ occurs in } \text{if } a = b \text{ then } a \text{ else } b, \text{ but } [a \ b : \text{Name}] \vdash a \# (\text{if } a = b \text{ then } a \text{ else } b) : \text{Name} \]

RHS is not in general a decidable relation, but (conjecture) the LHS is.
Local names

- **Local variables** in Algol-like languages:
  \[
  \text{new } X \text{ in } \langle \text{command} \rangle
  \]

- **Generativity** + local declarations in ML-like languages:
  \[
  \text{let } x = \text{ref}\langle \text{val} \rangle \text{ in } \langle \text{exp} \rangle
  \]

- Channel-name restriction in \( \pi \)-like process calculi:
  \[
  (\nu a)\langle \text{process} \rangle
  \]

- Use of **fresh names** in meta-programming/reasoning, e.g.
  \[
  \text{A-nf}(e_1 e_2) \triangleq \text{let } v_1 = e_1, v_2 = e_2 \text{ in } v_1 v_2
  \]
  \[
  \text{where } v_1 v_2 \text{ are fresh}
  \]

  Thesis: fresh names in metaprogramming/reasoning are always used in way that is **semantically trivial**.
Freshness theorem for $\text{Nom}$

If $f \in \text{Nom}(A \times D, D')$ satisfies for all $a,d$

\[ a \# d \Rightarrow a \# f(a, d) \]

then $\exists$ unique $f' \in \text{Nom}(D, D')$ s.t. for all $a,d$

\[ a \# d \Rightarrow f'd = f(a, d) \]

(so $f'd$ is $f(a, d)$ for some/any fresh $a$)
Freshness theorem for \textbf{Nom}

If \( f \in \text{Nom}(A \times D, D') \) satisfies for all \( a, d \)

\[
a \# d \Rightarrow a \# f(a, d)
\]

then \( \exists \) unique \( f' \in \text{Nom}(D, D') \) s.t. for all \( a, d \)

\[
a \# d \Rightarrow f'd = f(a, d)
\]

(so \( f'd \) is \( f(a, d) \) for some/any fresh \( a \))

We can express this kind of \textit{semantically trivial} locally scoped name in DTT + judgemental freshness, replacing this with \( \Gamma[a : \text{Name}] \vdash a \# t : T \)

and introducing syntax for \( f' \) as a function of \( f \)…
Judgementally fresh locally scoped names

Formation and introduction:

\[
\frac{\Gamma[a : A] \vdash a \# T(a)}{\Gamma \vdash \nu b. T(b)} \quad \frac{\Gamma[a : A] \vdash a \# t(a) : T(a)}{\Gamma \vdash \nu b. t(b) : \nu c. T(c)}
\]
Judgementally fresh locally scoped names

Formation and introduction:

\[
\begin{align*}
\Gamma[a : A] & \vdash a \# T(a) \\
\Gamma & \vdash \nu b. T(b) \\
\end{align*}
\]

\[
\begin{align*}
\Gamma[a : A] & \vdash a \# t(a) : T(a) \\
\Gamma & \vdash \nu b. t(b) : \nu c. T(c) \\
\end{align*}
\]

Computationally, \( \nu a. \_ \) is a no-op...

\[
\begin{align*}
\Gamma[a : A] & \vdash a \# T(a) \\
\Gamma[a : A] & \vdash \nu b. T(b) = T(a) \\
\Gamma[a : A] & \vdash a \# t(a) : T(a) \\
\Gamma[a : A] & \vdash \nu b. t(b) = t(a) : T(a) \\
\end{align*}
\]
Judgementally fresh locally scoped names

Formation and introduction:

\[
\begin{align*}
\Gamma[a : A] & \vdash a \# T(a) \\
\Gamma & \vdash \nu b. T(b) \\
\Gamma[a : A] & \vdash a \# t(a) : T(a) \\
\Gamma & \vdash \nu b. t(b) : \nu c. T(c)
\end{align*}
\]

Computationally, \( \nu a. \_ \) is a no-op . . .

\[
\begin{align*}
\Gamma[a : A] & \vdash a \# T(a) \\
\Gamma[a : A] & \vdash \nu b. T(b) = T(a) \\
\Gamma[a : A] & \vdash a \# t(a) : T(a) \\
\Gamma[a : A] & \vdash \nu b. t(b) = t(a) : T(a)
\end{align*}
\]

Sound interpretation in the CwF Nom using the Freshness Theorem.
FreshMLTT

[AMP, J. Matthiesen and J. Derikx, *A Dependent Type Theory with Abstractable Names*, LSFA 2014.]

- intensional Martin-Löf Type Theory
  - swappable names
  - judgementally fresh, locally scoped names
  - (dependent) name-abstraction types.
- Sound semantics using the CwF of nominal sets.
- Prototype implementation in development by Matthiesen.
FreshMLTT

[AMP, J. Matthiesen and J. Derikx, A Dependent Type Theory with Abstractable Names, LSFA 2014.]

- intensional Martin-Löf Type Theory
  + swappable names
  + judgementally fresh, locally scoped names
  + (dependent) name-abstraction types.
- Sound semantics using the CwF of nominal sets.
- Prototype implementation in development by Matthiesen.

To do: FreshMLTT has interesting (?) new forms of inductively defined indexed families of types using constructors with dependent name-abstractions in their arities (e.g. propositional freshness type – Curry-Howard for nominal logic).
Typing generative local names

\[ T \text{ type} \quad (x : \text{Name}) \vdash t(x) : T \]
\[ \forall x. t(x) : T \]

is safe, but inexpressive – seems inevitable that type expressions as well as term expressions may involve name generation:

\[ (x : \text{Name}) \vdash t(x) : T(x) \]
\[ \forall y. t(y) : \forall z. T(z) \]

If \( t \) & \( T \) are both generative, what does \( t : T \) mean? Are there models to guide us?
If $t$ & $T$ are both generative, what does $t : T$ mean? Are there models to guide us?

Wanted: a design combining DTT with generative locally scoped names that is user-friendly (no monadic-style over-sequentialization of the effect of name creation) and with a simple semantic model.

If you have one, see me afterwards!
END