Lecture 3

To be explained:

- Nominal sets, support and the freshness relation, $(-) \# (-)$.
- How is α-structural recursion proved?
- How to generalise α-structural recursion from the example language Λ to general languages with binders?
- What's involved with applying α-structural recursion in any particular case?
- Example: normalisation by evaluation.
- Machine-assisted support?

Example: normalisation by evaluation

U. Berger and H. Schwichtenberg, "An inverse of the evaluation functional for typed λ-calculus" (Proc. LICS 1991)
[and subsequent works by several authors].

Example: normalisation by evaluation

Produces $\beta\eta$-long normal forms of simply-typed λ-terms (fast!) by:

- taking denotation of terms in standard extensional functions model over a ground type of ASTs
- and then reifying elements of the model as ASTs in normal form.

[LN p 35]

Example: normalisation by evaluation

Produces $\beta\eta$-long normal forms of simply-typed λ-terms (fast!) by:

- taking denotation of terms in standard extensional functions model over a ground type of ASTs
- and then reifying elements of the model as ASTs in normal form.

Use the Freshness Theorem [LN p 19] for nominal sets to make sense of the naive definition of reifying an extensional function into a λ-abstraction (need to choose a fresh λ-bound variable)

Example: normalisation by evaluation

Produces $\beta\eta$-long normal forms of simply-typed λ-terms (fast!) by:

- t … ional f …
- a … STs in r …

"The problem is the "v fresh" condition; what exactly does it mean? Unlike such conditions as "x does not occur free in E", it is not even locally checkable whether a variable is fresh; freshness is a global property, defined with respect to a term that may not even be fully constructed yet." [8, p 157]

Use the Freshness Theorem [LN p 19] for nominal sets to make sense of the naive definition of reifying an extensional function into a λ-abstraction (need to choose a fresh λ-bound variable)

Simply-typed λ-calculus

$$
\begin{array}{llll}
\text{types} & \tau \in Ty & ::= & \iota \mid \tau \dot{\to} \tau \\
\text{terms} & t \in \Lambda & ::= & a \qquad\qquad (a \in \mathbb{V}) \\
 & & \mid & t \quad t \\
 & & \mid & \lambda a.\, t \qquad (a \in \mathbb{V})
\end{array}
$$

Simply-typed λ-calculus

$$
\begin{array}{llcll}
\text{types} & \tau \in Ty & ::= & \iota \mid \tau \mathrel{\dot{\to}} \tau & \\
\text{terms} & t \in \Lambda & ::= & (a)^{\tau} & (a \in \mathbb{V}_{\tau}) \\
 & & \mid & (t^{\tau \dot{\to} \tau'} t^{\tau})^{\tau'} & \\
 & & \mid & (\lambda a.\, t^{\tau'})^{\tau \dot{\to} \tau'} & (a \in \mathbb{V}_{\tau})
\end{array}
$$

Simply-typed λ-calculus

$$
\begin{array}{llcll}
\text{types} & \tau \in \mathit{Ty} & ::= & \iota \mid \tau \dot{\to} \tau & \\
\text{terms} & t \in \Lambda & ::= & (a)^{\tau} & (a \in \mathbb{V}_{\tau}) \\
 & & \mid & (t^{\tau \dot{\to} \tau'} t^{\tau})^{\tau'} & \\
 & & \mid & (\lambda a.\, t^{\tau'})^{\tau \dot{\to} \tau'} & (a \in \mathbb{V}_{\tau})
\end{array}
$$

$$
\left\{
\begin{array}{llcll}
\beta\eta\text{-long NFs} & n \in N & ::= & (\lambda a.\, n^{\tau'})^{\tau \dot{\to} \tau'} & (a \in \mathbb{V}_{\tau}) \\
 & & \mid & (u)^{\iota} & \\
\text{neutrals} & u \in U & ::= & (a)^{\tau} & (a \in \mathbb{V}_{\tau}) \\
 & & \mid & (u^{\tau \dot{\to} \tau'} n^{\tau})^{\tau'} &
\end{array}
\right.
$$

N.B. can (and will) regard N and U as subsets of Λ.

(By gad! they're GADTs)

Mutually inductively defined (nominal) sets Λ_τ of simply typed ASTs of type $\tau \in Ty$:

$$\Lambda_\tau = \mathbb{V}_\tau + \sum_{(\tau_1,\tau_2) \mid \tau_1 = (\tau_2 \dot{\to} \tau)} (\Lambda_{\tau_1} \times \Lambda_{\tau_2}) + \sum_{(\tau_1,\tau_2) \mid \tau = (\tau_1 \dot{\to} \tau_2)} (\mathbb{V}_{\tau_1} \times \Lambda_{\tau_2})$$

Mutually inductively defined (nominal) sets N_τ & U_τ, of $\beta\eta$-long NFs and neutrals of type $\tau \in Ty$:

$$N_\tau = \sum_{(\tau_1,\tau_2) \mid \tau = (\tau_1 \dot{\to} \tau_2)} (\mathbb{V}_{\tau_1} \times N_{\tau_2}) + U_\iota$$

$$U_\tau = \mathbb{V}_\tau + \sum_{(\tau_1,\tau_2) \mid \tau_1 = (\tau_2 \dot{\to} \tau)} (U_{\tau_1} \times N_{\tau_2})$$

The nominal signatures

Σ^{STL}

atom-sorts	data-sorts	constructors
v_τ	t_τ	$Vr_\tau : \mathsf{v}_\tau \longrightarrow \mathsf{t}_\tau$
		$Ap_{\tau,\tau'} : \mathsf{t}_{\tau \dot{\to} \tau'} * \mathsf{t}_\tau \longrightarrow \mathsf{t}_{\tau'}$
$(\tau \in Ty ::= \iota \mid \tau \dot{\to} \tau)$		$Lm_{\tau,\tau'} : \langle\!\langle \mathsf{v}_\tau \rangle\!\rangle \mathsf{t}_{\tau'} \longrightarrow \mathsf{t}_{\tau \dot{\to} \tau'}$

Σ^{LNF}

atom-sorts	data-sorts	constructors
v_τ	n_τ	$V_\tau : \mathsf{v}_\tau \longrightarrow \mathsf{u}_\tau$
	u_τ	$A_{\tau,\tau'} : \mathsf{u}_{\tau \dot{\to} \tau'} * \mathsf{n}_\tau \longrightarrow \mathsf{u}_{\tau'}$
		$L_{\tau,\tau'} : \langle\!\langle \mathsf{v}_\tau \rangle\!\rangle \mathsf{n}_{\tau'} \longrightarrow \mathsf{n}_{\tau \dot{\to} \tau'}$
$(\tau \in Ty ::= \iota \mid \tau \dot{\to} \tau)$		$I : \mathsf{u}_\iota \longrightarrow \mathsf{n}_\iota$

Terms / $\beta\eta$-long NFs / neutrals are identified up to α-equivalence $=_\alpha$ (definition as for any nominal signature).

$$\begin{aligned}
\boldsymbol{\Lambda}(\boldsymbol{\tau}) &\triangleq \boldsymbol{\Lambda}_{\boldsymbol{\tau}}/{=_\alpha} && \text{(typical element } \boldsymbol{e}\text{)}\\
\boldsymbol{N}(\boldsymbol{\tau}) &\triangleq \boldsymbol{N}_{\boldsymbol{\tau}}/{=_\alpha} && \text{(typical element } \boldsymbol{n}\text{)}\\
\boldsymbol{U}(\boldsymbol{\tau}) &\triangleq \boldsymbol{U}_{\boldsymbol{\tau}}/{=_\alpha} && \text{(typical element } \boldsymbol{u}\text{)}
\end{aligned}$$

There are injections

$$\begin{aligned}
\boldsymbol{i}_{\boldsymbol{\tau}} &: \boldsymbol{N}(\boldsymbol{\tau}) \longrightarrow \boldsymbol{\Lambda}(\boldsymbol{\tau})\\
\boldsymbol{j}_{\boldsymbol{\tau}} &: \boldsymbol{U}(\boldsymbol{\tau}) \longrightarrow \boldsymbol{\Lambda}(\boldsymbol{\tau})
\end{aligned}$$

induced by the inclusions $\boldsymbol{N}_{\boldsymbol{\tau}} \subseteq \boldsymbol{\Lambda}_{\boldsymbol{\tau}}$, $\boldsymbol{U}_{\boldsymbol{\tau}} \subseteq \boldsymbol{\Lambda}_{\boldsymbol{\tau}}$.

Normalisation

Wish to show the existence of

normalisation functions $\boxed{norm_\tau : \Lambda(\tau) \rightarrow N(\tau)}$

satisfying:

- $e_1 =_{\beta\eta} e_2 \;\Rightarrow\; norm_\tau\, e_1 = norm_\tau\, e_2$
- $norm_\tau(i_\tau\, n) = n$
- $i_\tau(norm_\tau\, e) =_{\beta\eta} e$

[LN p 37]

Normalisation

Wish to show the existence of

normalisation functions $\boxed{norm_\tau : \Lambda(\tau) \to N(\tau)}$

satisfying:

- $e_1 =_{\beta\eta} e_2 \;\Rightarrow\; norm_\tau\, e_1 = norm_\tau\, e_2$
- $norm_\tau(i_\tau\, n) = n$
- $i_\tau(norm_\tau\, e) =_{\beta\eta} e$

$\beta\eta$-conversion
= least congruence satisfying:

$$(\lambda a.\, e_1)e_2 \;=_{\beta\eta}\; (a := e_2)e_1$$

$$a \# e \;\Rightarrow\; e \;=_{\beta\eta}\; \lambda a.\, e\, a$$

Example: normalisation by evaluation

Produces $\beta\eta$-long normal forms of simply-typed λ-terms (fast!) by:

- taking denotation of terms in standard extensional functions model over a ground type of ASTs
- and then reifying elements of the model as ASTs in normal form.

Denotation

- Denotation of types as nominal sets:

$$
\begin{aligned}
D(\iota) &\triangleq N(\iota) \\
D(\tau \mathrel{\dot{\to}} \tau') &\triangleq D(\tau) \to_{\mathrm{fs}} D(\tau')
\end{aligned}
$$

Denotation

- Denotation of types as nominal sets:

$$D(\iota) \triangleq N(\iota)$$
$$D(\tau \dot{\rightarrow} \tau') \triangleq D(\tau) \rightarrow_{\mathrm{fs}} D(\tau')$$

- Terms $e \in \Lambda(\tau)$ in a given environment $\rho \in Env$ denote finitely supported elements $[\![e]\!]\rho \in D(\tau)$, satisfying:

$$[\![a]\!]\rho = \rho\, a$$
$$[\![e_1\, e_2]\!]\rho = [\![e_1]\!]\rho\,([\![e_2]\!]\rho)$$
$$[\![\lambda a^{\tau}.e]\!]\rho = \lambda d \in D(\tau).\, [\![e]\!](\rho\{a \mapsto d\})$$

Denota

type-respecting, finitely supported function from variables to denotations

- Denotation of types as nomi

$$
\begin{aligned}
D(\iota) &\triangleq N \\
D(\tau \dot{\to} \tau') &\triangleq D(\tau) \to_{\mathrm{fs}} D(\tau')
\end{aligned}
$$

- Terms $e \in \Lambda(\tau)$ in a given environment $\rho \in Env$ denote finitely supported elements $[\![e]\!]\rho \in D(\tau)$, satisfying:

$$
\begin{aligned}
[\![a]\!]\rho &= \rho\, a \\
[\![e_1\, e_2]\!]\rho &= [\![e_1]\!]\rho\,([\![e_2]\!]\rho) \\
[\![\lambda a^{\tau}.e]\!]\rho &= \lambda d \in D(\tau).\, [\![e]\!](\rho\{a \mapsto d\})
\end{aligned}
$$

Denotation

- Denotation of types as nominal sets:

$$D(\iota) \triangleq N(\iota)$$
$$D(\tau \dot{\to} \tau') \triangleq D(\tau) \to_{\mathrm{fs}} D(\tau')$$

- Terms $e \in \Lambda(\tau)$ in a given environment $\rho \in Env$ denote finitely supported elements $[\![e]\!]\rho \in D(\tau)$, satisfying:

$$[\![a]\!]\rho = \rho\, a$$
$$[\![e_1\, e_2]\!]\rho = [\![e_1]\!]\rho\,([\![e_2]\!]\rho)$$
$$[\![\lambda a^\tau . e]\!]\rho = \lambda d \in D(\tau).\, [\![e]\!](\rho\{a \mapsto d\})$$

updated environment

Denotation

- Denotation of types as nominal sets:

$$D(\iota) \triangleq N(\iota)$$
$$D(\tau \dot{\to} \tau') \triangleq D(\tau) \to_{\mathrm{fs}} D(\tau')$$

- Terms $e \in \Lambda(\tau)$ in a given environment $\rho \in Env$ denote finitely supported elements $[\![e]\!]\rho \in D(\tau)$, satisfying:

$$[\![a]\!]\rho = \rho\, a$$
$$[\![e_1\, e_2]\!]\rho = [\![e_1]\!]\rho\,([\![e_2]\!]\rho)$$
$$[\![\lambda a^\tau . e]\!]\rho = \lambda d \in D(\tau).\, [\![e]\!](\rho\{a \mapsto d\})$$

Why is $[\![-]\!]$ well-defined?

Denotation

- Denotation of types as nominal sets:

$$D(\iota) \triangleq N(\iota)$$
$$D(\tau \dot{\rightarrow} \tau') \triangleq D(\tau) \rightarrow_{\mathrm{fs}} D(\tau')$$

- Terms $e \in \Lambda(\tau)$ in a given environment $\rho \in Env$ denote finitely supported elements $[\![e]\!]\rho \in D(\tau)$, satisfying:

$$[\![a]\!]\rho = \rho\, a$$
$$[\![e_1\, e_2]\!]\rho = [\![e_1]\!]\rho\,([\![e_2]\!]\rho)$$
$$[\![\lambda a^\tau . e]\!]\rho = \lambda d \in D(\tau).\, [\![e]\!](\rho\{a \mapsto d\})$$

Use α-structural recursion for Σ^{STL} to define $[\![-]\!]$...

First attempt

$$
\begin{aligned}
X_{\mathsf{t}_\tau} &\triangleq Env \longrightarrow_{\mathrm{fs}} D(\tau) \\
f_{Vr_\tau} &\triangleq \lambda a \in \mathbb{A}_{\mathsf{v}_\tau}. \\
&\qquad \lambda \rho \in Env.\, \rho\, a \\
f_{Ap_{\tau,\tau'}} &\triangleq \lambda(\xi_1, \xi_2) \in X_{\mathsf{t}_{\tau \dot{\rightarrow} \tau'}} \times X_{\mathsf{t}_\tau}. \\
&\qquad \lambda \rho \in Env.\, \xi_1\, \rho\, (\xi_2\, \rho) \\
f_{Lm_{\tau,\tau'}} &\triangleq \lambda(a, \xi) \in \mathbb{A}_{\mathsf{v}_\tau} \times X_{\mathsf{t}_{\tau'}}. \\
&\qquad \lambda \rho \in Env. \lambda d \in D(\tau).\, \xi(\rho\{a \mapsto d\}) \\
A &\triangleq \emptyset
\end{aligned}
$$

First attempt

FCB for this function is

$$a \mathrel{\#} \lambda\rho \in Env.\lambda d \in D(\tau).\,\xi(\rho\{a \mapsto d\})$$

and is not true of every $\xi \in (Env \rightarrow_{\mathrm{fs}} D(\tau'))$!

$$\lambda\rho \in Env.\,\xi_1\,\rho\,(\xi_2\,\rho)$$

$$\begin{aligned} f_{Lm_{\tau,\tau'}} &\triangleq \lambda(a,\xi) \in \mathbb{A}_{\mathsf{v}_\tau} \times X_{\mathsf{t}_{\tau'}}. \\ &\qquad \lambda\rho \in Env.\lambda d \in D(\tau).\,\xi(\rho\{a \mapsto d\}) \\ A &\triangleq \emptyset \end{aligned}$$

First attempt

FCB for this function is

$$a \,\#\, \lambda\rho \in Env.\lambda d \in D(\tau).\,\xi(\rho\{a \mapsto d\})$$

and is not true of every $\xi \in (Env \to_{\mathrm{fs}} D(\tau'))$!

Have to strengthen the "recursion hypothesis" by suitably restricting the class of functions ξ used for the α-structural recursion.

$$\lambda\rho \in Env.\,\xi_1\,\rho\,(\xi_2\,\rho)$$

$$\begin{aligned} f_{Lm_{\tau,\tau'}} &\triangleq \lambda(a,\xi) \in \mathbb{A}_{\mathsf{v}_\tau} \times X_{\mathsf{t}_{\tau'}}. \\ &\qquad \lambda\rho \in Env.\lambda d \in D(\tau).\,\xi(\rho\{a \mapsto d\}) \end{aligned}$$

$$A \triangleq \emptyset$$

Strengthen the "recursion hypothesis" by restricting $\xi \in (Env \rightarrow_{\mathrm{fs}} D(\tau))$ to functions having the following two expected properties of $[\![-]\!]$.

1. $[\![e]\!]\rho$ only depends on the value of ρ at the free variables of e.
2. $[\![\pi \cdot e]\!]\rho = [\![e]\!](\rho \circ \pi)$
   (special case of substitution property of denotations).

Second attempt

$$
\begin{aligned}
X_{\mathsf{t}_\tau} &\triangleq \{\xi \in Env \to_{\mathrm{fs}} D(\tau) \mid \Phi_1(\xi) \;\&\; \Phi_2(\xi)\} \\
f_{Vr_\tau} &\triangleq \lambda a \in \mathbb{A}_{\mathsf{v}_\tau}. \\
&\qquad \lambda \rho \in Env.\, \rho\, a \\
f_{Ap_{\tau,\tau'}} &\triangleq \lambda(\xi_1, \xi_2) \in X_{\mathsf{t}_{\tau \dot{\to} \tau'}} \times X_{\mathsf{t}_\tau}. \\
&\qquad \lambda \rho \in Env.\, \xi_1\, \rho\, (\xi_2\, \rho) \\
f_{Lm_{\tau,\tau'}} &\triangleq \lambda(a, \xi) \in \mathbb{A}_{\mathsf{v}_\tau} \times X_{\mathsf{t}_{\tau'}}. \\
&\qquad \lambda \rho \in Env. \lambda d \in D(\tau).\, \xi(\rho\{a \mapsto d\}) \\
A &\triangleq \emptyset
\end{aligned}
$$

Second attempt

$$X_{\mathbf{t}_\tau} \triangleq \{\xi \in Env \longrightarrow_{\mathrm{fs}} D(\tau) \mid \Phi_1(\xi) \;\&\; \Phi_2(\xi)\}$$

$$\begin{aligned}
\Phi_1(\xi) \;&\triangleq\; (\exists A \in P_{\mathrm{fin}}(\mathbb{A}))\\
&\quad (\forall \tau \in Ty, a \in \mathbb{A}_{\mathsf{v}_\tau}, d \in D(\tau), \rho \in Env)\\
&\qquad a \notin A \;\Rightarrow\; \xi(\rho\{a \mapsto d\}) = \xi\,\rho
\end{aligned}$$

$$\begin{aligned}
\Phi_2(\xi) \;&\triangleq\; (\forall \pi \in Perm, \rho \in Env)\\
&\qquad (\pi \cdot \xi)\,\rho = \xi(\rho \circ \pi)
\end{aligned}$$

Second attempt

$$
\begin{aligned}
X_{\mathbf{t}_\tau} &\triangleq \{\xi \in Env \to_{\mathrm{fs}} D(\tau) \mid \Phi_1(\xi) \;\&\; \Phi_2(\xi)\} \\
f_{Vr_\tau} &\triangleq \lambda a \in \mathbb{A}_{\mathsf{v}_\tau}. \\
&\qquad \lambda \rho \in Env.\, \rho\, a \\
f_{Ap_{\tau,\tau'}} &\triangleq \lambda(\xi_1, \xi_2) \in X_{\mathbf{t}_{\tau \dot{\to} \tau'}} \times X_{\mathbf{t}_\tau}. \\
&\qquad \lambda \rho \in Env.\, \xi_1\, \rho\, (\xi_2\, \rho) \\
f_{Lm_{\tau,\tau'}} &\triangleq \lambda(a, \xi) \in \mathbb{A}_{\mathsf{v}_\tau} \times X_{\mathbf{t}_{\tau'}}. \\
&\qquad \lambda \rho \in Env. \lambda d \in D(\tau).\, \xi(\rho\{a \mapsto d\})
\end{aligned}
$$

Have to prove the $f_{(-)}$ map into $X_{\mathbf{t}_\tau}$ and prove FCB for $f_{Lm_{\tau,\tau'}}$: $(\forall a \in \mathbb{A}_{\mathsf{v}_\tau}, \xi \in X_{\mathbf{t}_{\tau'}})\; a \# f_{Lm_{\tau,\tau'}}(a, \xi)$.

Given a & ξ, choosing any sufficiently fresh a', then

$$a =$$
$$(a\,a') \cdot a'$$
$$\#$$
$$(a\,a') \cdot \lambda\rho \in Env.\lambda d \in D(\tau).\,\xi(\rho\{a \mapsto d\})$$
$$=$$
$$\lambda\rho \in Env.\lambda d \in D(\tau).\,((a\,a') \cdot \xi)(\rho\{a' \mapsto d\})$$
$$= \{\text{since } \Phi_2(\xi)\}$$
$$\lambda\rho \in Env.\lambda d \in D(\tau).\,\xi(\rho\{a' \mapsto d\} \circ (a\,a'))$$
$$= \{\text{since } a' \neq a\}$$
$$\lambda\rho \in Env.\lambda d \in D(\tau).\,\xi(\rho\{a \mapsto d\}\{a' \mapsto \rho\,a\})$$
$$= \{\text{since } \Phi_1(\xi)\}$$
$$\lambda\rho \in Env.\lambda d \in D(\tau).\,\xi(\rho\{a \mapsto d\})$$
$$\triangleq f_{Lm_{\tau,\tau'}}(a, \xi) \qquad \square$$

Example: normalisation by evaluation

Produces $\beta\eta$-long normal forms of simply-typed λ-terms (fast!) by:

- taking denotation of terms in standard extensional functions model over a ground type of ASTs
- and then reifying elements of the model as ASTs in normal form.

Reification ($\downarrow_\tau$) & reflection ($\uparrow_\tau$)

$\tau \in Ty, d \in D(\tau) \mapsto \downarrow_\tau d \in N(\tau)$:

$$\boxed{\begin{array}{rcl} \downarrow_\iota n & \triangleq & n \\ \downarrow_{\tau \dot{\to} \tau'} f & \triangleq & fresh(\lambda a \in \mathbb{A}_{\mathsf{v}_\tau}.\, \lambda a^\tau.\, \downarrow_{\tau'}(f(\uparrow_\tau a))) \end{array}}$$

$\tau \in Ty, u \in U(\tau) \mapsto \uparrow_\tau u \in D(\tau)$:

$$\boxed{\begin{array}{rcl} \uparrow_\iota u & \triangleq & u \\ \uparrow_{\tau \dot{\to} \tau'} u & \triangleq & \lambda d \in D(\tau).\, \uparrow_{\tau'}(u\,(\downarrow_\tau d)) \end{array}}$$

Reification ($\downarrow_\tau$) & reflection ($\uparrow_\tau$)

$\tau \in Ty, d \in D(\tau) \mapsto \downarrow_\tau d \in N(\tau)$:

$$\begin{aligned} \downarrow_\iota n &\triangleq n \\ \downarrow_{\tau \dot\to \tau'} f &\triangleq fresh(\lambda a \in \mathbb{A}_{\mathsf{v}_\tau}.\, \lambda a^\tau.\, \downarrow_{\tau'}(f(\uparrow_\tau a))) \end{aligned}$$

finitely supported function $D(\tau) \to_{\mathrm{fs}} D(\tau')$

AST/α in $N(\tau \dot\to \tau')$

τ … $\mapsto \uparrow_\tau u \in D($…

$$u$$

$$\lambda d \in D(\tau).\, \uparrow_{\tau'}(u\,(\downarrow_\tau d))$$

[LN p 43]

Reification ($\downarrow_\tau$) & reflection ($\uparrow_\tau$)

$\tau \in Ty, d \in D(\tau) \mapsto \downarrow_\tau d \in N(\tau)$:

$$\boxed{\begin{aligned} \downarrow_\iota n &\triangleq n \\ \downarrow_{\tau \dot{\to} \tau'} f &\triangleq \mathit{fresh}(\lambda a \in \mathbb{A}_{\mathsf{v}_\tau} . \, \lambda a^\tau . \downarrow_{\tau'}(f(\uparrow_\tau a))) \end{aligned}}$$

Uses an easily proved application of the <u>Freshness theorem</u> [LN p 19]

Given $h \in (\mathbb{A}_{\mathsf{v}_\tau} \to_{\mathrm{fs}} N(\tau'))$ satisfying

$$(\exists a \in \mathbb{A}_{\mathsf{v}_\tau})\; a \# h \;\;\&\;\; a \# h(a)$$

then $\exists!$ element $\mathit{fresh}(h) \in N(\tau')$ satisfying

$$(\forall a \in \mathbb{A}_{\mathsf{v}_\tau})\; a \# h \;\Rightarrow\; h(a) = \mathit{fresh}(h)$$

Normalisation

$norm_\tau : \Lambda(\tau) \to N(\tau)$ is given by

$$\boxed{norm_\tau(e) \triangleq \downarrow_\tau(\llbracket e \rrbracket \rho_0)}$$

where $\rho_0 \in Env$ is the environment mapping $a \in \mathbb{A}_{\mathsf{V}_\tau} \mapsto a \in U(\tau) \mapsto \uparrow_\tau a \in D(\tau)$ (for all $\tau \in Ty$).

Normalisation

$norm_\tau : \Lambda(\tau) \to N(\tau)$ is given by

$$\boxed{norm_\tau(e) \triangleq \downarrow_\tau(\llbracket e \rrbracket \rho_0)}$$

where $\rho_0 \in Env$ is the environment mapping
$a \in \mathbb{A}_{\mathsf{v}_\tau} \mapsto a \in U(\tau) \mapsto \uparrow_\tau a \in D(\tau)$ (for all $\tau \in Ty$).

Of the three required properties of $norm$

(1) $e_1 =_{\beta\eta} e_2 \;\Rightarrow\; norm_\tau\, e_1 = norm_\tau\, e_2$

(2) $norm_\tau(i_\tau\, n) = n$

(3) $i_\tau(norm_\tau\, e) =_{\beta\eta} e$

(1) & (2) are proved using α-structural induction;
(3) is trickier, but can be proved using a logical relations argument [LN pp 44,45].

Pause

To be explained:

- Nominal sets, support and the freshness relation, $(-) \# (-)$.
- How is α-structural recursion proved?
- How to generalise α-structural recursion from the example language Λ to general languages with binders?
- What's involved with applying α-structural recursion in any particular case?
- Example: normalisation by evaluation.
- Machine-assisted support?

Machine-assisted support

- Norrish's HOL4 development. [TPHOLs '04]
- Urban & Tasson's Isabelle/HOL theory of nominal sets ("p-sets") and α-structural induction for λ-calculus. [CADE-20, 2005].

  Isabelle's axiomatic type classes are helpful.

  Wanted: full implementation of α-structural recursion/induction theorems parameterised by a user-declared nominal signature

  (in either HOL4, or Isabelle/HOL, or both).

Machine-assisted support

- Gabbay's FM-HOL [35yrs of Automath, 2002].

  Wanted: a new machine-assisted higher-order logic to support reasoning about ordinary sets and nominal sets simultaneously.

  - Should incorporate a reflection principle to exploit

    Fact The standard set-theoretic model of HOL (without choice) restricts to finitely supported elements; e.g. if we apply a construction of HOL-ε to finitely supported functions we get another such.

  - Also needs some (lightweight!) treatment of partial functions.

Nominal functional programming

- Shinwell's Fresh O'Caml patch of Objective Caml [ML Workshop 2005]
  - latest manifestation of AMP-Gabbay-Shinwell FreshML design
    [ICFP 2003] [TCS 342(2005) 28-55]
  - extends O'Caml datatypes with atoms, atom-binding and atom-unbinding via pattern-matching-with-freshening

Nominal signature:

atom-sorts	data-sorts	constructors
$\mathbf{v}$	$\mathbf{t}$	$V : \mathbf{v} \longrightarrow \mathbf{t}$
		$A : \mathbf{t} * \mathbf{t} \longrightarrow \mathbf{t}$
		$L : \langle\!\langle \mathbf{v} \rangle\!\rangle \mathbf{t} \longrightarrow \mathbf{t}$
		$F : \langle\!\langle \mathbf{v} \rangle\!\rangle ((\langle\!\langle \mathbf{v} \rangle\!\rangle \mathbf{t}) * \mathbf{t}) \longrightarrow \mathbf{t}$

Fresh O'Caml declarations:

```
bindable_type v
type t  =  V of v
        |  A of t * t
        |  L of ≪v≫t
        |  F of ≪v≫((≪v≫t)*t)
```

Capture-avoiding substitution

- $(a := e)a_1 \triangleq$ if $a_1 = a$ then e else a_1
- $(a := e)(e_1\, e_2) \triangleq ((a := e)e_1)((a := e)e_2)$
- $(a := e)(\lambda a_1.e_1) \triangleq$

  if $a_1 \notin fv(a, e)$ then $\lambda a_1.(a := e)e_1$

  else don't care!
- $(a := e)(\mathtt{letrec}\, a_1\, a_2 = e_1 \;\mathtt{in}\; e_2) \triangleq$

  if $a_1, a_2 \# (a, e)$ & $a_2 \# (a_1, e_2)$

  then $\mathtt{letrec}\, a_1\, a_2 = (a := e)e_1 \;\mathtt{in}\; (a := e)e_2$

  else don't care!

Declaration of capture-avoiding substitution function in Fresh O'Caml:

```
let sub(a:v)(e:t) : t ⟶ t =
let rec s(e′:t) : t =
  match e′ with
    V a1 ⟶ if a1 = a then e else e′
  | A(e1,e2) ⟶ A(s e1 , s e2)
  | L(≪a1≫e1) ⟶ L(≪a1≫(s e1))
  | F(≪a1≫(≪a2≫e1 , e2)) ⟶
      F(≪a1≫(≪a2≫(s e1) , s e2))
in s
```

Declaration of capture-avoiding substitution function in Fresh O'Caml:

```
let sub(a:v)(e:t) : t → t =
let rec s(e′:t) : t =
  match e′ with
    V a1 → if a1 = a then e else e′
  | A(e1,e2) → A(s e1 , s e2)
  | L(«a1»e1) → L(«a1»(s e1))
  | F(«a1»(«a2»e1 , e2)) →
      F(«a1»(«a2»(s e1) , s e2))
in s
```

- dynamics of unbinding guarantees freshness preconditions
- RHS of match clauses <u>not</u> checked for FCB. . .

Declaration of capture-avoiding substitution function in Fresh O'Caml:

```
let sub(a:v)(e:t) : t → t =
```

Matching a value $\langle\!\langle a\rangle\!\rangle v$ against a pattern $\langle\!\langle x\rangle\!\rangle p$ causes:

1. value-environment to be updated to associate x with a globally fresh atom a'
2. p to be matched against the value obtained from v by [lazily?] renaming all occurrences of a in v to be a'

- dynamics of unbinding guarantees freshness preconditions
- RHS of match clauses not checked for FCB. . .

A mis-guided attempt to calculate the list of bound variables of an α-equivalence class of an AST:

```
let rec bv(e:t) : v list
  match e with
    V _ → []
  | A(e1,e2) → (bv e1)@(bv e2)
  | L(≪a1≫e1) → a1::(bv e1)
  | F(≪a1≫(≪a2≫e1 , e2)) →
      a1::a2::(bv e1)@(bv e2)
```

This results in a Fresh O'Caml function

`bv : t → v list`

that, when applied to a value `e:t`, returns a list of <u>fresh</u> atoms.

Nominal functional programming

- Shinwell's Fresh O'Caml patch of Objective Caml [ML Workshop 2005]
- Cheney's FreshLib library for Haskell/ghc 6.4 [ICFP 2005]
  - exploits generic programming features of latest ghc ("SYB")
- Pottier's Cαml code generation tool for O'Caml [ML Workshop 2005]
  - supports patterns of binding more general than those of nominal signatures

Neither FreshLib nor Cαml support unbinding via (nested) patterns

Nominal logic programming

- Theoretical basis: Urban-AMP-Gabbay nominal unification [TCS 323(2004) 473-497.]
  - best-known algorithm quadratic in size of nominal terms
  - unification variables only for data-sorts, not atom-sorts
- Experimental language: Cheney-Urban AlphaProlog [ICLP 2004]

Nominal logic programming

- Theoretical basis: Urban-AMP-Gabbay nominal unification [TCS 323(2004) 473-497.]
    - best-known algorithm quadratic in size of nominal terms
    - unification variables only for data-sorts, not atom-sorts
- Experimental language: Cheney-Urban AlphaProlog [ICLP 2004]

Is there a "nominal logical framework"?

(Cf. Schöpp & Stark [CSL 2004]—category of nominal sets supports a rich model of dependent types.)

Assessment

- α-Structural recursion & induction principles apply directly to standard notions of AST & α-equivalence within ordinary HOL

  —like Gordon & Melham's "5 Axioms" work [TPHOLs '96], except closer to informal practice regarding freshness of bound names (more applicable).

- Crucial finite support property is automatically preserved by constructions in HOL

  (if we avoid choice principles).

- Mathematical treatment of "fresh names" afforded by nominal sets is proving useful in other contexts

  (e.g. Abramsky et al [LICS '04], Winskel & Turner [200?]).

Conclusion

Claim: dealing with issues of bound names and α-equivalence on ASTs is made easier through use of permutations (rather than traditional use of non-bijective renamings).

Is the use of name-permutations & support simple enough to become part of standard practice?

(It's now part of mine!)