Nominal Abstractions

- Nominal abstractions need to satisfy:
  - $\pi \bullet ([a].x) = [\pi \bullet a].(\pi \bullet x)$
  - $[a].x = [b].y$ iff

    $(a = b \wedge x = y) \vee$
    $(a \neq b \wedge x = (a\,b) \bullet y \wedge a \# y)$

- These properties imply that for finitely supported x:

$$\mathsf{supp}([a].x) = \mathsf{supp}(x) - \{a\}$$

- Are there any structures that satisfy these properties?

Nominal Abstractions

- Nominal abstractions need to satisfy:
  - $\pi \bullet ([a].x) = [\pi \bullet a].(\pi \bullet x)$
  - $[a].x = [b].y$ iff

    $(a = b \wedge x = y) \vee$
    $(a \neq b \wedge x = (a\,b) \bullet y \wedge a \# y)$

- These properties imply that for finitely sup

From some perspective, this is a completely uninteresting question, because the user will only see this "interface".

- Are there any structures that satisfy these properties?

Yes...

- α-equivalence classes (sets of syntax trees), e.g. $[\lambda a.(a\ c)]_\alpha = [\lambda b.(b\ c)]_\alpha$
- terms with de-Bruijn indices and named free variables, like $\lambda(1\ c)$.
  (you need a function abs which "abstracts" a variable:
  $abs(x, t) \mapsto \lambda(\ldots)$)
- a weak HOAS encoding (lambdas as functions — the function for $\lambda a.(a\ c)$ will be the same as the one for $\lambda b.(b\ c)$)

Yes...

- α-equivalence classes (sets of syntax trees), e.g. $[\lambda a.(a\ c)]_\alpha = [\lambda b.(b\ c)]_\alpha$
- I could now stop here and probably go for α-equivalence classes (Norrish is doing this now with the help of a package by Hohmeier for HOL4).
- a weak HOAS encoding (lambdas as functions — the function for $\lambda a.(a\ c)$ will be the same as the one for $\lambda b.(b\ c)$)

Yes...

- α-equivalence classes (sets of syntax trees), e.g. $[\lambda a.(a\ c)]_\alpha = [\lambda b.(b\ c)]_\alpha$
- terms with de-Bruijn indices and named free variables, like $\lambda(1\ c)$.
  (you need a function abs which "abstracts" a variable:
  $abs(x, t) \mapsto \lambda(\ldots)$)
- a weak HOAS encoding (lambdas as functions — the function for $\lambda a.(a\ c)$ will be the same as the one for $\lambda b.(b\ c)$)

Function $[a].t$ '=' $[\lambda a.t]_\alpha$

$$[a].t \stackrel{\text{def}}{=} (\lambda b.\ \text{if } a = b \text{ then Some}(t) \text{ else if } b \# t \text{ then Some}((b\,a) \bullet t) \text{ else None})$$

type: $\alpha \to \iota$ option

Function $[a].t$ '=' $[\lambda a.t]_\alpha$

$$[a].t \stackrel{\text{def}}{=} (\lambda b.\ \text{if } a = b \text{ then Some}(t) \text{ else if } b \# t \text{ then Some}((b\,a) \bullet t) \text{ else None})$$

This is supposed to stand for the α-equivalence class of $\lambda a.t$.

Function $[a].t$ '=' $[\lambda a.t]_\alpha$

$$[a].\mathsf{App}(a, c) \stackrel{\text{def}}{=}$$
$$(\lambda b.\ \text{if } a = b$$
$$\text{then Some}(\mathsf{App}(a, c))$$
$$\text{else if } b \# \mathsf{App}(a, c)$$
$$\text{then Some}((b\ a) \bullet \mathsf{App}(a, c)) \text{ else None})$$

Let's check this for $[a].\mathsf{App}(a, c)$:

Function $[a].t$ '=' $[\lambda a.t]_\alpha$

$$[a].\mathsf{App}(a,c) \stackrel{\text{def}}{=} (\lambda b.\ \text{if } a = b \text{ then } \mathsf{Some}(\mathsf{App}(a,c)) \text{ else if } b \# \mathsf{App}(a,c) \text{ then } \mathsf{Some}((b\,a) \bullet \mathsf{App}(a,c)) \text{ else None})$$

Let's check this for $[a].\mathsf{App}(a,c)$:

$[a].\mathsf{App}(a,c)$ 'applied to' a 'gvs' $\mathsf{Some}(\mathsf{App}(a,c))$

Function $[a].t$ '=' $[\lambda a.t]_\alpha$

$$[a].\mathsf{App}(a, c) \stackrel{\text{def}}{=}$$
$$(\lambda b.\ \text{if } a = b$$
$$\text{then Some}(\mathsf{App}(a, c))$$
$$\text{else if } b \# \mathsf{App}(a, c)$$
$$\text{then Some}((b\ a) \bullet \mathsf{App}(a, c))\ \text{else None})$$

Let's check this for $[a].\mathsf{App}(a, c)$:

$[a].\mathsf{App}(a, c)$ 'applied to' a 'gvs' $\text{Some}(\mathsf{App}(a, c))$

$[a].\mathsf{App}(a, c)$ 'applied to' b 'gvs' $\text{Some}(\mathsf{App}(b, c))$

Function $[a].t$ '=' $[\lambda a.t]_\alpha$

$$[a].\mathsf{App}(a, c) \stackrel{\text{def}}{=}$$
$$(\lambda b.\ \text{if } a = b$$
$$\text{then Some}(\mathsf{App}(a, c))$$
$$\text{else if } b \# \mathsf{App}(a, c)$$
$$\text{then Some}((b\ a) \bullet \mathsf{App}(a, c)) \text{ else None})$$

Let's check this for $[a].\mathsf{App}(a, c)$:

$[a].\mathsf{App}(a, c)$ 'applied to' a 'gvs' $\text{Some}(\mathsf{App}(a, c))$

$[a].\mathsf{App}(a, c)$ 'applied to' b 'gvs' $\text{Some}(\mathsf{App}(b, c))$

$[a].\mathsf{App}(a, c)$ 'applied to' c 'gvs' None

Function $[a].t$ '=' $[\lambda a.t]_\alpha$

$$[a].\mathsf{App}(a,c) \stackrel{\text{def}}{=}$$
$$(\lambda b.\ \text{if } a = b$$
$$\text{then Some}(\mathsf{App}(a,c))$$
$$\text{else if } b \# \mathsf{App}(a,c)$$
$$\text{then Some}((b\,a) \bullet \mathsf{App}(a,c)) \text{ else None})$$

Let's check this for $[a].\mathsf{App}(a,c)$:

$[a].\mathsf{App}(a,c)$ 'applied to' a 'gvs' $\text{Some}(\mathsf{App}(a,c))$

$[a].\mathsf{App}(a,c)$ 'applied to' b 'gvs' $\text{Some}(\mathsf{App}(b,c))$

$[a].\mathsf{App}(a,c)$ 'applied to' c 'gvs' None

$[a].\mathsf{App}(a,c)$ 'applied to' d 'gvs' $\text{Some}(\mathsf{App}(d,c))$

$\vdots$

Function $[a].t$ '=' $[\lambda a.t]_\alpha$

$$[a].\mathsf{App}(a,c) \stackrel{\text{def}}{=} (\lambda b.\ \text{if } a = b \text{ then } \mathsf{Some}(\mathsf{App}(a,c)) \text{ else if } b \# \mathsf{App}(a,c) \text{ then } \mathsf{Some}((b\,a) \bullet \mathsf{App}(a,c)) \text{ else None})$$

Let's check this for $[a].\mathsf{App}(a,c)$:

$[a].\mathsf{App}(a,c)$ 'applied to' a 'gvs' $\mathsf{Some}(\mathsf{App}(a,c))$ $\quad$ '$\lambda a.(a\,c)$'

$[a].\mathsf{App}(a,c)$ 'applied to' b 'gvs' $\mathsf{Some}(\mathsf{App}(b,c))$ $\quad$ '$\lambda b.(b\,c)$'

$[a].\mathsf{App}(a,c)$ 'applied to' c 'gvs' None

$[a].\mathsf{App}(a,c)$ 'applied to' d 'gvs' $\mathsf{Some}(\mathsf{App}(d,c))$ $\quad$ '$\lambda d.(d\,c)$'

$\vdots$

Function $[a].t$ '=' $[\lambda a.t]_\alpha$

$$[a].\mathrm{App}(a,c) \stackrel{\mathrm{def}}{=} (\lambda b.\ \text{if } a = b \text{ then } \mathrm{Some}(\mathrm{App}(a,c)) \text{ else if } b \# \mathrm{App}(a,c) \text{ then } \mathrm{Some}((b\ a) \bullet \mathrm{App}(a,c)) \text{ else None})$$

Let's check this for $[a].\mathrm{App}(a,c)$:

$[a].\mathrm{App}(a,c)$ 'applied to' a 'gvs' $\mathrm{Some}(\mathrm{App}(a,c))$ $\quad$ '$\lambda a.(a\ c)$'

$[a].\mathrm{App}(a,c)$ 'applied to' b 'gvs' $\mathrm{Some}(\mathrm{App}(b,c))$ $\quad$ '$\lambda b.(b\ c)$'

$[a].\mathrm{App}(a,c)$ 'applied to' c 'gvs' None

$[a].\mathrm{App}(a,c)$ 'applied to' d 'gvs' $\mathrm{Some}(\mathrm{App}(d,c))$ $\quad$ '$\lambda d.(d\ c)$'

$\vdots$

$[\lambda a.(a\ c)]_\alpha$:

$\vdots$

$$\pi \bullet [a].t = [\pi \bullet a].(\pi \bullet t)$$

$\pi \bullet [a].t$

$$\pi \bullet [a].t = [\pi \bullet a].(\pi \bullet t)$$

$\pi \bullet [a].t$

$\stackrel{\text{def}}{=}$ $\pi \bullet \lambda b.\, \texttt{if}\ a = b\ \texttt{then}\ \mathsf{Some}(t)$
$\texttt{else if}\ b \# t\ \texttt{then}\ \mathsf{Some}((a\,b) \bullet t)\ \texttt{else}\ \mathsf{None}$

$$\pi \bullet [a].t = [\pi \bullet a].(\pi \bullet t)$$

$$\begin{array}{ll} & \pi \bullet [a].t \\ \stackrel{\text{def}}{=} & \pi \bullet \lambda b.\, \texttt{if } a = b \texttt{ then } \mathsf{Some}(t) \\ & \qquad \texttt{else if } b \# t \texttt{ then } \mathsf{Some}((a\,b) \bullet t) \texttt{ else } \mathsf{None} \\ \stackrel{\text{def}}{=} & \lambda b.\, \pi \bullet \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(t) \\ & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}((a\ \pi^{-1} \bullet b) \bullet t) \texttt{ else } \mathsf{None} \end{array}$$

$\pi \bullet [a].t = [\pi \bullet a].(\pi \bullet t)$

$$
\begin{array}{ll}
 & \pi \bullet [a].t \\
\stackrel{\text{def}}{=} & \pi \bullet \lambda b.\, \texttt{if } a = b \texttt{ then } \mathsf{Some}(t) \\
 & \qquad \texttt{else if } b \# t \texttt{ then } \mathsf{Some}((a\,b) \bullet t) \texttt{ else } \mathsf{None} \\
\stackrel{\text{def}}{=} & \lambda b.\, \pi \bullet \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}((a\ \pi^{-1} \bullet b) \bullet t) \texttt{ else } \mathsf{None} \\
= & \lambda b.\, \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(\pi \bullet t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}(\pi \bullet (a\ \pi^{-1} \bullet b) \bullet t) \texttt{ else } \mathsf{None}
\end{array}
$$

$\pi \bullet [a].t = [\pi \bullet a].(\pi \bullet t)$

$$
\begin{array}{ll}
 & \pi \bullet [a].t \\
\stackrel{\text{def}}{=} & \pi \bullet \lambda b.\, \texttt{if } a = b \texttt{ then } \mathsf{Some}(t) \\
 & \qquad \texttt{else if } b \# t \texttt{ then } \mathsf{Some}((a\ b) \bullet t) \texttt{ else } \mathsf{None} \\
\stackrel{\text{def}}{=} & \lambda b.\, \pi \bullet \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}((a\ \pi^{-1} \bullet b) \bullet t) \texttt{ else } \mathsf{None} \\
= & \lambda b.\, \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(\pi \bullet t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}(\pi \bullet (a\ \pi^{-1} \bullet b) \bullet t) \texttt{ else } \mathsf{None} \\
= & \lambda b.\, \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(\pi \bullet t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}((\pi \bullet a\ b) \bullet \pi \bullet t) \texttt{ else } \mathsf{None}
\end{array}
$$

$\pi \bullet [a].t = [\pi \bullet a].(\pi \bullet t)$

$$
\begin{array}{ll}
 & \pi \bullet [a].t \\
\stackrel{\text{def}}{=} & \pi \bullet \lambda b.\, \texttt{if } a = b \texttt{ then } \mathsf{Some}(t) \\
 & \qquad \texttt{else if } b \# t \texttt{ then } \mathsf{Some}((a\ b) \bullet t) \texttt{ else } \mathsf{None} \\
\stackrel{\text{def}}{=} & \lambda b.\, \pi \bullet \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}((a\ \pi^{-1} \bullet b) \bullet t) \texttt{ else } \mathsf{None} \\
= & \lambda b.\, \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(\pi \bullet t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}(\pi \bullet (a\ \pi^{-1} \bullet b) \bullet t) \texttt{ else } \mathsf{None} \\
= & \lambda b.\, \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(\pi \bullet t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}((\pi \bullet a\ b) \bullet \pi \bullet t) \texttt{ else } \mathsf{None} \\
= & \lambda b.\, \texttt{if } \pi \bullet a = b \texttt{ then } \mathsf{Some}(\pi \bullet t) \\
 & \quad \texttt{else if } b \# \pi \bullet t \texttt{ then } \mathsf{Some}((\pi \bullet a\ b) \bullet \pi \bullet t) \texttt{ else } \mathsf{None}
\end{array}
$$

$\pi \bullet [a].t = [\pi \bullet a].(\pi \bullet t)$

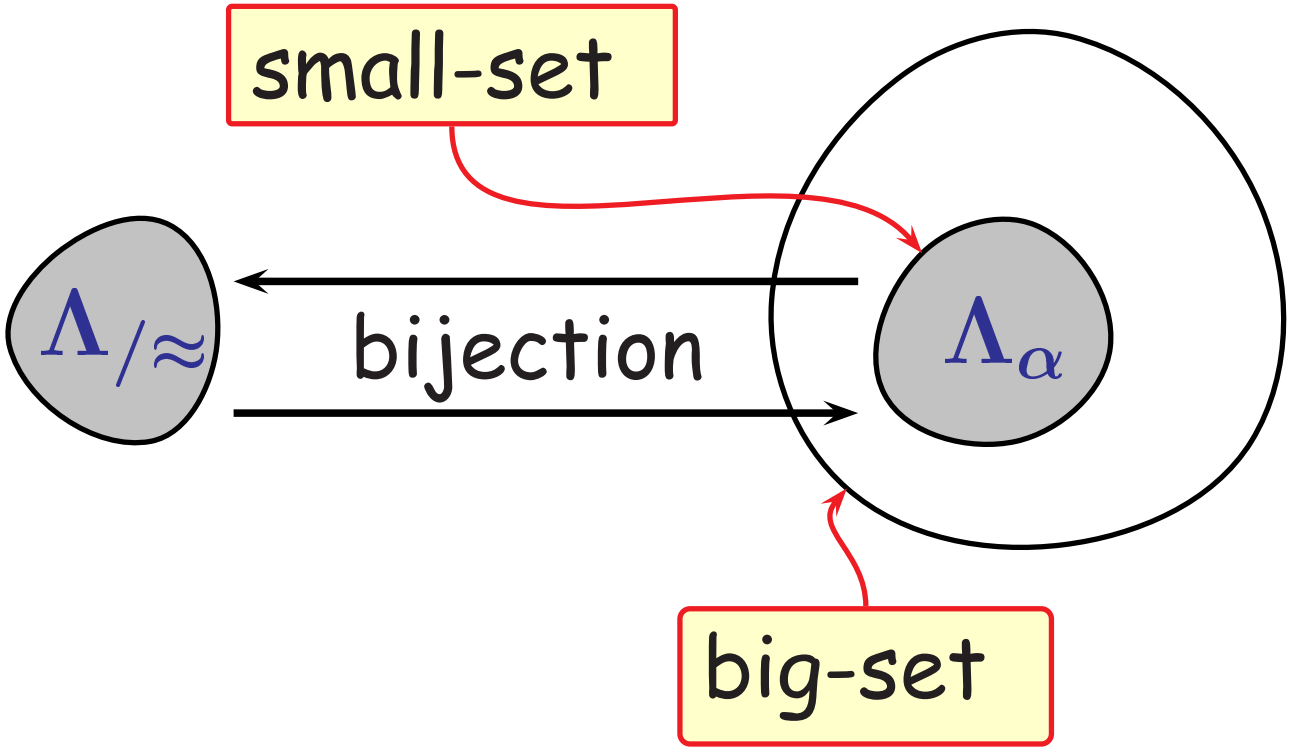
$$
\begin{array}{ll}
 & \pi \bullet [a].t \\
\stackrel{\text{def}}{=} & \pi \bullet \lambda b.\, \texttt{if } a = b \texttt{ then } \mathsf{Some}(t) \\
 & \qquad \texttt{else if } b \# t \texttt{ then } \mathsf{Some}((a\ b) \bullet t) \texttt{ else } \mathsf{None} \\
\stackrel{\text{def}}{=} & \lambda b.\, \pi \bullet \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}((a\ \pi^{-1} \bullet b) \bullet t) \texttt{ else } \mathsf{None} \\
= & \lambda b.\, \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(\pi \bullet t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}(\pi \bullet (a\ \pi^{-1} \bullet b) \bullet t) \texttt{ else } \mathsf{None} \\
= & \lambda b.\, \texttt{if } a = \pi^{-1} \bullet b \texttt{ then } \mathsf{Some}(\pi \bullet t) \\
 & \quad \texttt{else if } \pi^{-1} \bullet b \# t \texttt{ then } \mathsf{Some}((\pi \bullet a\ b) \bullet \pi \bullet t) \texttt{ else } \mathsf{None} \\
= & \lambda b.\, \texttt{if } \pi \bullet a = b \texttt{ then } \mathsf{Some}(\pi \bullet t) \\
 & \quad \texttt{else if } b \# \pi \bullet t \texttt{ then } \mathsf{Some}((\pi \bullet a\ b) \bullet \pi \bullet t) \texttt{ else } \mathsf{None} \\
\stackrel{\text{def}}{=} & [\pi \bullet a].(\pi \bullet t)
\end{array}
$$

Weak HOAS Encoding

I define **inductively** the α-equivalence classes of lambda-terms—but they still have **names**.

Weak HOAS Encoding

I define **inductively** the α-equivalence classes of lambda-terms—but they still have **names**.



Big-Set / $Ftrms$

One step back: naïve attempt for big-set

$$
\begin{array}{rcl}
Ftrm & ::= & \mathsf{Var} : Atom \\
 & | & \mathsf{App} : Ftrm \times Ftrm \\
 & | & \mathsf{Lam} : Ftrm\ Set
\end{array}
$$

Big-Set / *Ftrms*

One step back: naïve attempt for big-set

$$\begin{array}{rcl} Ftrm & ::= & \mathsf{Var} : Atom \\ & | & \mathsf{App} : Ftrm \times Ftrm \\ & | & \mathsf{Lam} : Ftrm\ Set \end{array}$$

we try to encode the α-equivalence class as the set of lambda-terms

$$[t]_\alpha \stackrel{\text{def}}{=} \{t' \mid t \approx t'\}$$

Big-Set / $Ftrms$

Better attempt for big-set

$$\begin{array}{rcl} Ftrm & ::= & \text{Var} : Atom \\ & | & \text{App} : Ftrm \times Ftrm \\ & | & \text{Lam} : Atom \rightarrow Ftrm \text{ option} \end{array}$$

same idea, but encoding with (partial) functions, along the lines:

"if $t' \in [\lambda a.t]_\alpha$ then 'yes' else None"

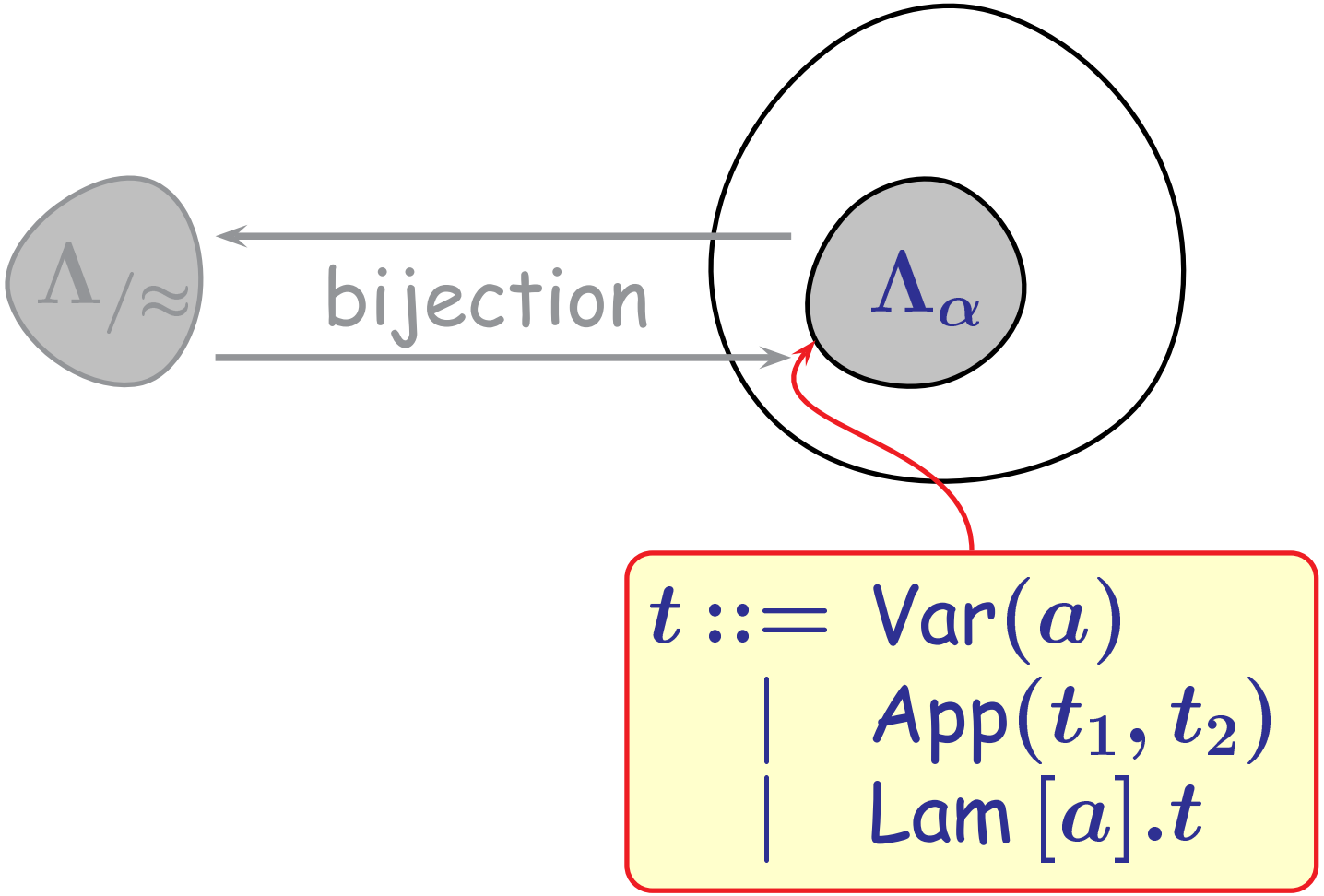
Permutations for Big-Set

Given the permutation action for atoms, we can permute all "**free**" atoms in $Ftrms$:

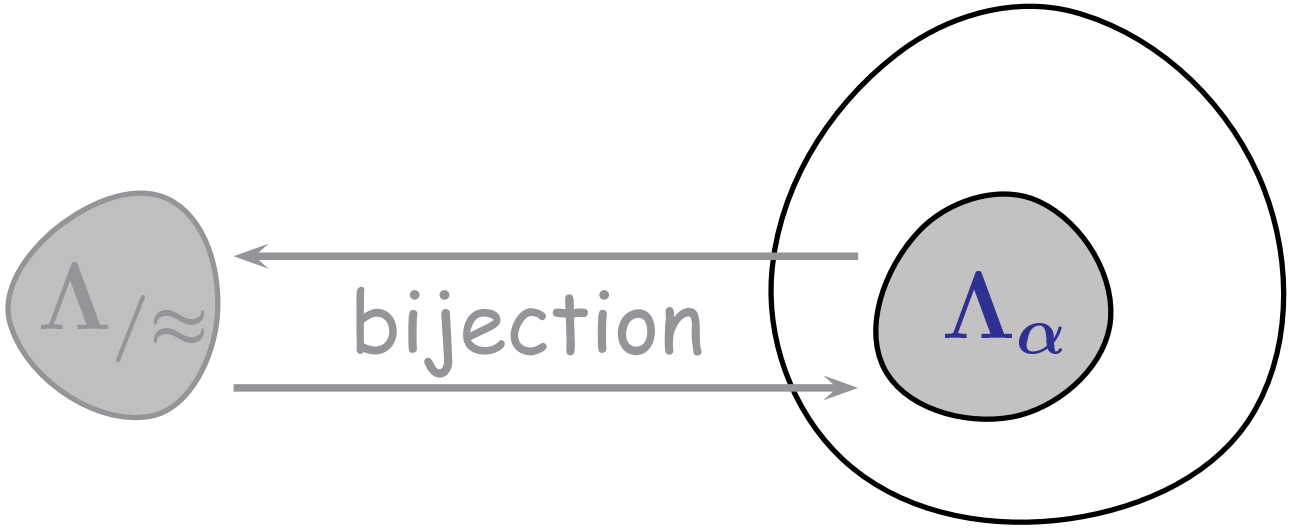
$$\begin{array}{lcl} \pi \bullet \mathsf{Var}(a) & \stackrel{\text{def}}{=} & \mathsf{Var}(\pi \bullet a) \\ \pi \bullet \mathsf{App}(t_1, t_2) & \stackrel{\text{def}}{=} & \mathsf{App}(\pi \bullet t_1, \pi \bullet t_2) \\ \pi \bullet \mathsf{Lam}(fn) & \stackrel{\text{def}}{=} & \mathsf{Lam}(\lambda a.\pi \bullet (fn(\pi^{-1} \bullet a))) \end{array}$$

The $pt_{\alpha,ftrm}$-properties are satisfied.

Definition of Small-Set



Definition of Small-Set



$$\frac{}{\mathsf{Var}(a) \in \Lambda_\alpha} \qquad \frac{t_1 \in \Lambda_\alpha \quad t_2 \in \Lambda_\alpha}{\mathsf{App}(t_1, t_2) \in \Lambda_\alpha}$$

$$\frac{t \in \Lambda_\alpha}{\mathsf{Lam}\,[a].t \in \Lambda_\alpha}$$

Definition of Small-Set

Which also means that we have a familiar induction principle in place for Λ_α (in a moment). And all terms in Λ_α have finite support.

$$\frac{}{\mathsf{Var}(a) \in \Lambda_\alpha} \qquad \frac{t_1 \in \Lambda_\alpha \quad t_2 \in \Lambda_\alpha}{\mathsf{App}(t_1, t_2) \in \Lambda_\alpha}$$

$$\frac{t \in \Lambda_\alpha}{\mathsf{Lam}\,[a].t \in \Lambda_\alpha}$$

Definition of Small-Set

Which also means that we have a familiar induction principle in place for Λ_α (in a moment). And all terms in Λ_α have finite support.

$$\begin{array}{ll} \mathsf{supp}(\mathsf{Var}(a)) & = \{a\} \\ \mathsf{supp}(\mathsf{App}(t_1, t_2)) & = \mathsf{supp}(t1, t2) \\ \mathsf{supp}(\mathsf{Lam}\,[a].t) & = \mathsf{supp}([a].t) = \mathsf{supp}(t) - \{a\} \end{array}$$

$$\frac{t \in \Lambda_\alpha}{\mathsf{Lam}\,[a].t \in \Lambda_\alpha}$$

Bijection

In order to show that $\Lambda_{/\approx}$ and Λ_α are bijective we define a function q from Λ to Λ_α:

$$\begin{array}{lcl} q(a) & \stackrel{\text{def}}{=} & \mathsf{Var}(a) \\ q(t_1\, t_2) & \stackrel{\text{def}}{=} & \mathsf{App}(q(t_1), q(t_2)) \\ q(\lambda a.t) & \stackrel{\text{def}}{=} & \mathsf{Lam}\,[a].q(t) \end{array}$$

with the property

$$t_1 \approx t_2 \;\Leftrightarrow\; q(t_1) = q(t_2)$$

Struct. Induction on Λ_α

$$\frac{}{\mathsf{Var}(a) \in \Lambda_\alpha} \qquad \frac{t_1 \in \Lambda_\alpha \quad t_2 \in \Lambda_\alpha}{\mathsf{App}(t_1, t_2) \in \Lambda_\alpha}$$

$$\frac{t \in \Lambda_\alpha}{\mathsf{Lam}\,[a].t \in \Lambda_\alpha}$$

Structural Induction Principle:

$$\frac{(\forall a)\ P\,(\mathsf{Var}(a)) \qquad (\forall t_1, t_2)\ P\,t_1 \wedge P\,t_2 \Rightarrow P\,(\mathsf{App}(t_1, t_2)) \qquad (\forall a, t)\ P\,t \Rightarrow P\,(\mathsf{Lam}\,[a].t)}{(\forall t)\ P\,t}$$

Substitution Lemma: If $x \not\equiv y$ and $x \notin FV(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Proof: By induction on the structure of M.

- **Case 1:** M is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal L, for $x \notin FV(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal z.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and z is not free in N, L. Then by induction hypothesis

$$\begin{aligned}
&(\lambda z.M_1)[x := N][y := L] \\
\equiv\ &\lambda z.(M_1[x := N][y := L]) \\
\equiv\ &\lambda z.(M_1[y := L][x := N[y := L]]) \\
\equiv\ &(\lambda z.M_1)[y := L][x := N[y := L]].
\end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis.

" □ "

Can We Do Better?

We consider induction hypotheses of the form $P\ t\ x$ with x **finitely supported**. In the case of the substitution lemma:

$$P = \lambda M.\ \lambda(x, y, N, L).$$
$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$$
$$x = (x, y, N, L)$$

$$\frac{\forall a.\ P\,(\mathsf{Var}(a))\,x \qquad \forall t_1, t_2.\ (\forall z.\ P\,t_1\,z) \wedge (\forall z.\ P\,t_2\,z) \Rightarrow P\,(\mathsf{App}(t_1, t_2))\,x \qquad \forall a.\ a \# x \Rightarrow (\forall t.\ (\forall z.\ P\,t\,z) \Rightarrow P\,(\mathsf{Lam}\,[a].t)\,x)}{P\,t\,x}$$

Proof

By an induction over Λ_α. We need to strengthen the induction hypothesis to

$$\forall t, \pi.\ P\,(\pi \bullet t)\,x$$

In the lambda-case we need to show:

$$P\,(\pi \bullet \mathsf{Lam}\,[a].t)\,x$$

We have:

$$\forall t, \pi.\ P\,(\pi \bullet t)\,x$$

$$\forall a.\, a \# x \Rightarrow (\forall t.\ (\forall z.\, P\,t\,z) \Rightarrow P\,(\mathsf{Lam}\,[a].t)\,x)$$

Proof (II)

We have:

$$\forall t, \pi.\ P\,(\pi \bullet t)\,x$$

$$\forall a.\, a \# x \Rightarrow (\forall t.\ (\forall z.\, P\, t\, z) \Rightarrow P\,(\mathsf{Lam}\,[a].t)\,x)$$

We choose a c so that $c \# (x, \pi \bullet a, \pi \bullet t)$.
Hence $\forall t.\ (\forall z.\, P\, t\, z) \Rightarrow P\,(\mathsf{Lam}\,[c].t)\,x$.
Moreover $P\,((c\ \ \pi \bullet a) :: \pi \bullet t)\,x$. Therefore
$P\,(\mathsf{Lam}\,[c].((c\ \ \pi \bullet a) :: \pi \bullet t))\,x$. Now

$$\mathsf{Lam}\,[c].((c\,\pi \bullet a) :: \pi \bullet t) = \mathsf{Lam}\,[\pi \bullet a].(\pi \bullet t).$$

So we have $P\,(\pi \bullet \mathsf{Lam}\,[a].t)\,x$. Done.

Substitution

We do not know yet how to define functions, but assume we have the following **total** definition for substitution

$$\begin{array}{rcl} \mathsf{Var}(a)[b := t'] & \stackrel{\text{def}}{=} & \texttt{if } a = b \texttt{ then } t' \texttt{ else } \mathsf{Var}(a) \\ \mathsf{App}(t_1, t_2)[b := t'] & \stackrel{\text{def}}{=} & \mathsf{App}(t_1[b := t'], t_2[b := t']) \\ \mathsf{Lam}\,[a].t[b := t'] & \stackrel{\text{def}}{=} & \mathsf{Lam}\,[a].(t[b := t']) \\ & & \text{provided } a \# (b, t') \end{array}$$

Total, "because":

$$\forall\, t, b, t'.\ \exists\, r.\ \mathsf{subst_rel}(t, b, t', r)\ .$$

"Forget" Lemma

This lemma is part of the substitution lemma.

$$b \# t \longrightarrow t[b := t'] = t$$

Induction on t.

$$P = \lambda t.\ \lambda(b, t').\ \ a \# t \longrightarrow t[b := t'] = t$$
$$x = (b, t')$$

We want to prove

$$P\ t\ x\ .$$

"Forget" Lemma

This lemma is part of the substitution lemma.

$$b \# t \longrightarrow t[b := t'] = t$$

Induction on t.

$$P = \lambda t.\ \lambda(b, t').\ \ a \# t \longrightarrow t[b := t'] = t$$
$$x = (b, t')$$

We want to prove

$$P\, t\, (b, t')\ .$$

Proof (I)

We have to show that:

$$\forall\, a.\ b \# \mathsf{Var}(a) \longrightarrow \mathsf{Var}(a)[b := t'] = \mathsf{Var}(a)$$

$$\frac{\begin{array}{l}\forall a.\ P\,(\mathsf{Var}(a))\,x \\ \forall t_1, t_2.\ (\forall z.\ P\,t_1\,z) \wedge (\forall z.\ P\,t_2\,z) \Rightarrow P\,(\mathsf{App}(t_1, t_2))\,x \\ \forall a.\ a \# x \Rightarrow (\forall t.\ (\forall z.\ P\,t\,z) \Rightarrow P\,(\mathsf{Lam}\,[a].t)\,x)\end{array}}{(\forall t)\ P\,t\,x}$$

Proof (I)

We have to show that:

$$\forall\, a.\ b \# \mathrm{Var}(a) \longrightarrow \mathrm{Var}(a)[b := t'] = \mathrm{Var}(a)$$

This amounts to showing:

$$\frac{b \# \mathrm{Var}(a)}{\mathrm{Var}(a)[b := t'] = \mathrm{Var}(a)}$$

Proof (I)

We have to show that:

$$\forall\, a.\ b \# \mathrm{Var}(a) \longrightarrow \mathrm{Var}(a)[b := t'] = \mathrm{Var}(a)$$

This amounts to showing:

$$\frac{b \neq a}{\mathrm{Var}(a)[b := t'] = \mathrm{Var}(a)}$$

Proof (I)

We have to show that:

$$\forall\, a.\ b \# \mathrm{Var}(a) \longrightarrow \mathrm{Var}(a)[b ::= t'] = \mathrm{Var}(a)$$

This amounts to showing:

$$\frac{b \neq a}{\mathrm{Var}(a) = \mathrm{Var}(a)}$$

Proof (II)

We have:

$$\forall(b, t').\ b \# t_1 \longrightarrow t_1[b := t'] = t_1$$
$$\forall(b, t').\ b \# t_2 \longrightarrow t_2[b := t'] = t_2$$

We have to show:

$$b \# \mathsf{App}(t_1, t_2) \longrightarrow \mathsf{App}(t_1, t_2)[b := t'] = \mathsf{App}(t_1, t_2)$$

$$\frac{\forall a.\ P\,(\mathsf{Var}(a))\,x \qquad \forall t_1, t_2.\ (\forall z.\ P\,t_1\,z) \wedge (\forall z.\ P\,t_2\,z) \Rightarrow P\,(\mathsf{App}(t_1, t_2))\,x \qquad \forall a.\ a \# x \Rightarrow (\forall t.\ (\forall z.\ P\,t\,z) \Rightarrow P\,(\mathsf{Lam}\,[a].t)\,x)}{(\forall t)\ P\,t\,x}$$

Proof (II)

We have:

$$\forall(b, t').\ b \# t_1 \longrightarrow t_1[b := t'] = t_1$$
$$\forall(b, t').\ b \# t_2 \longrightarrow t_2[b := t'] = t_2$$

We have to show:

$$\frac{b \# \mathsf{App}(t_1, t_2)}{\mathsf{App}(t_1, t_2)[b := t'] = \mathsf{App}(t_1, t_2)}$$

Proof (II)

We have:

$$\forall(b, t').\ b \# t_1 \longrightarrow t_1[b := t'] = t_1$$
$$\forall(b, t').\ b \# t_2 \longrightarrow t_2[b := t'] = t_2$$

We have to show:

$$\frac{b \# t_1 \quad b \# t_2}{\mathsf{App}(t_1, t_2)[b := t'] = \mathsf{App}(t_1, t_2)}$$

Proof (II)

We have:

$$\forall(b, t').\ b \# t_1 \longrightarrow t_1[b := t'] = t_1$$
$$\forall(b, t').\ b \# t_2 \longrightarrow t_2[b := t'] = t_2$$

We have to show:

$$\frac{t_1[b := t'] = t_1 \quad t_2[b := t'] = t_2}{\mathsf{App}(t_1, t_2)[b := t'] = \mathsf{App}(t_1, t_2)}$$

Proof (II)

We have:

$$\forall(b, t').\ b \# t_1 \longrightarrow t_1[b := t'] = t_1$$
$$\forall(b, t').\ b \# t_2 \longrightarrow t_2[b := t'] = t_2$$

We have to show:

$$\frac{t_1[b := t'] = t_1 \quad t_2[b := t'] = t_2}{\mathsf{App}(t_1[b := t'], t_2[b := t']) = \mathsf{App}(t_1, t_2)}$$

Proof (II)

We have:

$$\forall(b, t').\ b \# t_1 \longrightarrow t_1[b := t'] = t_1$$
$$\forall(b, t').\ b \# t_2 \longrightarrow t_2[b := t'] = t_2$$

We have to show:

$$\frac{t_1[b := t'] = t_1 \quad t_2[b := t'] = t_2}{\mathsf{App}(t_1, t_2) = \mathsf{App}(t_1, t_2)}$$

Proof (III)

We have:

$$a \# (b, t')$$

$$\forall (b, t'). \; b \# t \longrightarrow t[b := t'] = t$$

We have to show:

$$b \# \mathsf{Lam}\,[a].t \longrightarrow (\mathsf{Lam}\,[a].t)[b := t'] = \mathsf{Lam}\,[a].t$$

$$\frac{\forall a. \; P\,(\mathsf{Var}(a))\,x \qquad \forall t_1, t_2. \; (\forall z. \; P\,t_1\,z) \wedge (\forall z. \; P\,t_2\,z) \Rightarrow P\,(\mathsf{App}(t_1, t_2))\,x \qquad \forall a. \; a \# x \Rightarrow (\forall t. \; (\forall z. \; P\,t\,z) \Rightarrow P\,(\mathsf{Lam}\,[a].t)\,x)}{(\forall t) \; P\,t\,x}$$

Proof (III)

We have:

$$a \# (b, t')$$

$$\forall (b, t'). \; b \# t \longrightarrow t[b := t'] = t$$

We have to show:

$$\frac{b \# \mathsf{Lam}\,[a].t}{(\mathsf{Lam}\,[a].t)[b := t'] = \mathsf{Lam}\,[a].t}$$

Proof (III)

We have:

$$a \# (b, t') \Rightarrow a \# b \wedge a \# t'$$

$$\forall (b, t'). \; b \# t \longrightarrow t[b := t'] = t$$

We have to show:

$$\frac{b \# \text{Lam}\,[a].t}{(\text{Lam}\,[a].t)[b := t'] = \text{Lam}\,[a].t}$$

Proof (III)

We have:

$$a \# (b, t') \Rightarrow a \# b \wedge a \# t'$$

$$\forall (b, t'). \; b \# t \longrightarrow t[b := t'] = t$$

We have to show:

$$\frac{b \# t}{(\mathsf{Lam}\,[a].t)[b := t'] = \mathsf{Lam}\,[a].t}$$

Proof (III)

We have:

$$a \# (b, t') \Rightarrow a \# b \wedge a \# t'$$

$$\forall (b, t').\ b \# t \longrightarrow t[b := t'] = t$$

We have to show:

$$\frac{t[b := t'] = t}{(\mathsf{Lam}\,[a].t)[b := t'] = \mathsf{Lam}\,[a].t}$$

Proof (III)

We have:

$$a \# (b, t') \Rightarrow a \# b \wedge a \# t'$$

$$\forall (b, t').\ b \# t \longrightarrow t[b := t'] = t$$

We have to show:

$$\frac{t[b := t'] = t}{(\mathsf{Lam}\,[a].t)[b := t'] = \mathsf{Lam}\,[a].t}$$

$$\begin{array}{rcl} \mathsf{Var}(a)[b := t'] & \stackrel{\text{def}}{=} & \texttt{if } a = b \texttt{ then } t' \texttt{ else } \mathsf{Var}(a) \\ \mathsf{App}(t_1, t_2)[b := t'] & \stackrel{\text{def}}{=} & \mathsf{App}(t_1[b := t'], t_2[b := t']) \\ \mathsf{Lam}\,[a].t[b := t'] & \stackrel{\text{def}}{=} & \mathsf{Lam}\,[a].(t[b := t']) \\ & & \text{provided } a \# (b, t') \end{array}$$

Proof (III)

We have:

$$a \# (b, t') \Rightarrow a \# b \wedge a \# t'$$

$$\forall (b, t').\ b \# t \longrightarrow t[b := t'] = t$$

We have to show:

$$\frac{t[b := t'] = t}{\mathsf{Lam}\,[a].(t[b := t']) = \mathsf{Lam}\,[a].t}$$

Proof (III)

We have:

$$a \# (b, t') \Rightarrow a \# b \wedge a \# t'$$

$$\forall (b, t'). \; b \# t \longrightarrow t[b := t'] = t$$

We have to show:

$$\frac{t[b := t'] = t}{\mathsf{Lam}\,[a].t = \mathsf{Lam}\,[a].t}$$

```
lemma forget:
shows "b # t ⟶ t[b := t'] = t"
proof (nominal_induct t rule: lam_induct)
  case (Var b t' a) (* case 1: variables *)
  show "b # Var a ⟶ (Var a)[b := t'] = Var a" by (simp add: fresh_atm)
next
  case (App b t' t1 t2) (* case 3: applications *)
  thus "b # App t1 t2 ⟶ (App t1 t2)[b := t'] = App t1 t2" by simp
next
  case (Lam b t' a t) (* case 2: lambdas *)
  have ih: "∀ b t'. b # t ⟶ t[b := t'] = t" by fact
  have f1: "a#(b, t')" by fact
  hence f2: "a ≠ b" by (simp add: fresh_prod fresh_atm)
  show "b # Lam [a].t ⟶ (Lam [a].t)[b := t'] = Lam [a].t"
  proof
    assume "b # Lam [a].t"
    hence "b # t" using f2 by (simp add: abs_fresh)
    hence "t[b := t'] = t" using ih by simp
    thus "(Lam [a].t)[b := t'] = Lam [a].t" using f1 by simp
  qed
qed
```

Authoritative Version

```
lemma forget:
shows "b # t ⟶ t[b := t'] = t"
apply (nominal_induct t rule: lam_induct)
apply (auto simp add: abs_fresh fresh_prod fresh_atm)
done
```

What is Coming Next

- a bit more about inductions
- how can functions be defined over α-equivalence classes (this is a sore point in the current version of the nominal datatype package)
- ...