Alpha-Equivalence

- How do we get $\lambda x.x = \lambda y.y$?

```ml
datatype lam =
  | Var "name"
  | App "lam" "lam"
  | Lam "name" "lam"
```

This does not work.

- In this case we have to make sure (manually) that everything we do is invariant modulo alpha-equivalence. Curry & Feys need in “Combinatory Logic” 10 pages just for showing that

$$ M \approx_{\alpha} M', \quad N \approx_{\alpha} N' \quad \Rightarrow \quad M[x := N] \approx_{\alpha} M'[x := N'] $$
HOL includes a mechanism for introducing new types:

1. If you can identify a non-empty subset in an existing type, then you can turn this set into a new type.

   ```
   typedef my_silly_new_type = "{0, 1, 2::nat}"
   by auto
   ```
HOL includes a mechanism for introducing new types:

- If you can identify a non-empty subset in an existing type, then you can turn this set into a new type.

```avel
typedef my_silly_new_type = "{0, 1, 2::nat}"
by auto
```

Diagram:
- Small set: \{0, 1, 2\}
- Big set: nats
- New type: `my_silly_new_type`
Types in HOL

HOL includes a mechanism for introducing new types:

- If you can identify a non-empty subset in an existing type, then you can turn this set into a new type.

  `typedef my_silly_new_type = "{0, 1, 2::nat}" by auto`

- As a result, we will be able to introduce the type of named $\alpha$-equivalence classes.

  `nominal_datatype lam =
    Var "name"
    | App "lam" "lam"
    | Lam "\langle name\rangle lam"`
We can define 'raw' lambda-terms (i.e. trees) as

```haskell
datatype raw_lam =
    Var "name"
  |
    App "raw_lam" "raw_lam"
  |
    Lam "name" "raw_lam"
```

and then quotient them modulo $\alpha$.

```haskell
typedef lam = "(UNIV::raw_lam set) // alpha"
```
First Naive Attempt

- We can define 'raw' lambda-terms (i.e. trees) as
  ```
  datatype raw_lam =
      Var "name"
    | App "raw_lam" "raw_lam"
    | Lam "name" "raw_lam"
  ```

- and then quotient them modulo \( \alpha \).
  ```
  typedef lam = "(UNIV::raw_lam set) // alpha"
  ```

- Problem: This is not an inductive definition and we have to provide an induction principle for lam (recall Barendregt's substitution lemma). This is painful.
We like to define

```plaintext
datatype pre_lam =
    Var "name"
| App "pre_lam" "pre_lam"
| Lam "(name × pre_lam) set"
```
We like to define

```
datatype pre_lam =
  Var "name"
| App "pre_lam" "pre_lam"
| Lam "(name × pre_lam) set"
```

and then perform the following construction

```
big set: pre_lam
```
We like to define

```plaintext
datatype pre_lam =
    Var "name"
  | App "pre_lam" "pre_lam"
  | Lam "(name \times pre_lam) set"
```

and then perform the following construction.
Second Naive Attempt

We like to define

\[
\text{datatype } \text{pre\_lam} = \\
\quad \text{Var } "\text{name}" \\
| \quad \text{App } "\text{pre\_lam}" "\text{pre\_lam}" \\
| \quad \text{Lam } "(\text{name} \times \text{pre\_lam}) \text{ set}" \\
\]

and then perform the following construction
Second Naive Attempt (2)

Unfortunately this does **not** work, because datatypes need to be definable as sets.

But a Cantor argument will tell us that pre_lam set will always be bigger than pre_lam.

datatype pre_lam =
  Var "name"
| App "pre_lam" "pre_lam"
| Lam "(name × pre_lam) set"
Unfortunately this does **not** work, because datatypes need to be definable as sets.

But a Cantor argument will tell us that `pre_lam set` will always be bigger than `pre_lam`.

datatype pre_lam =
  Var "name"
  | App "pre_lam" "pre_lam"
  | Lam "(name×pre_lam) set"
Unfortunately this does not work, because datatypes need to be definable as sets. But a Cantor argument will tell us that pre_lam set will always be bigger than pre_lam.

In the following we will make this idea to work by finding an alternative representation for $\alpha$-equivalence classes.
Free Variables

What are the free variables of a lambda-term?
What are the free variables of a lambda-term?

\[ \text{fv}(a) \overset{\text{def}}{=} \{a\} \]
\[ \text{fv}(t_1 \ t_2) \overset{\text{def}}{=} \text{fv}(t_1) \cup \text{fv}(t_2) \]
\[ \text{fv}(\lambda a. t) \overset{\text{def}}{=} \text{fv}(t) - \{a\} \]
What are the free variables of a lambda-term?

\[
\text{fv}(a) \overset{\text{def}}{=} \{a\}
\]
\[
\text{fv}(t_1 \ t_2) \overset{\text{def}}{=} \text{fv}(t_1) \cup \text{fv}(t_2)
\]
\[
\text{fv}(\lambda a. t) \overset{\text{def}}{=} \text{fv}(t) - \{a\}
\]

What are the free variables of a pair?

\[
\text{fv}(t_1, t_2) \overset{\text{def}}{=} \text{fv}(t_1) \cup \text{fv}(t_2)
\]
Free Variables

What are the free variables of a lambda-term?

\[
\begin{align*}
fv(a) & \overset{\text{def}}{=} \{a\} \\
fv(t_1 t_2) & \overset{\text{def}}{=} fv(t_1) \cup fv(t_2) \\
fv(\lambda a. t) & \overset{\text{def}}{=} fv(t) - \{a\}
\end{align*}
\]

What are the free variables of a pair?

\[
fv(t_1, t_2) \overset{\text{def}}{=} fv(t_1) \cup fv(t_2)
\]
Free Variables

What are the free variables of a lambda-term?

\[
\begin{align*}
fv(a) & \overset{\text{def}}{=} \{a\} \\
fv(t_1 t_2) & \overset{\text{def}}{=} fv(t_1) \cup fv(t_2) \\
fv(\lambda a. t) & \overset{\text{def}}{=} fv(t) - \{a\}
\end{align*}
\]

What are the free variables of a pair?

\[
fv(t_1, t_2) \overset{\text{def}}{=} fv(t_1) \cup fv(t_2)
\]

What are the free variables of a list?
Free Variables

What are the free variables of a lambda-term?

\[
\begin{align*}
\text{fv}(a) & \overset{\text{def}}{=} \{a\} \\
\text{fv}(t_1 t_2) & \overset{\text{def}}{=} \text{fv}(t_1) \cup \text{fv}(t_2) \\
\text{fv}(\lambda a.t) & \overset{\text{def}}{=} \text{fv}(t) - \{a\}
\end{align*}
\]

What are the free variables of a pair?

\[
\text{fv}(t_1, t_2) \overset{\text{def}}{=} \text{fv}(t_1) \cup \text{fv}(t_2)
\]

What are the free variables of a list?

\[
\begin{align*}
\text{fv}([]) & \overset{\text{def}}{=} \emptyset \\
\text{fv}(t::ts) & \overset{\text{def}}{=} \text{fv}(t) \cup \text{fv}(ts)
\end{align*}
\]
Free Variables

- What are the free variables of a lambda-term?
  
  \[
  fv(a) \overset{\text{def}}{=} \{a\} \\
  fv(t_1 t_2) \overset{\text{def}}{=} fv(t_1) \cup fv(t_2) \\
  fv(\lambda a.t) \overset{\text{def}}{=} fv(t) - \{a\}
  \]

- What are the free variables of a pair?
  
  \[
  fv(t_1, t_2) \overset{\text{def}}{=} fv(t_1) \cup fv(t_2)
  \]

- What are the free variables of a list?
  
  \[
  fv([]) \overset{\text{def}}{=} \emptyset \\
  fv(t::ts) \overset{\text{def}}{=} fv(t) \cup fv(ts)
  \]

- What are the free variables of a set?
Free Variables

- What are the free variables of a lambda-term?
  
  \[
  \text{fv}(a) \overset{\text{def}}{=} \{a\} \\
  \text{fv}(t_1 \, t_2) \overset{\text{def}}{=} \text{fv}(t_1) \cup \text{fv}(t_2) \\
  \text{fv}(\lambda a.\, t) \overset{\text{def}}{=} \text{fv}(t) \setminus \{a\}
  \]

- What are the free variables of a pair?
  
  \[
  \text{fv}(t_1, t_2) \overset{\text{def}}{=} \text{fv}(t_1) \cup \text{fv}(t_2)
  \]

- What are the free variables of a list?
  
  \[
  \text{fv}([]) \overset{\text{def}}{=} \emptyset \\
  \text{fv}(t :: ts) \overset{\text{def}}{=} \text{fv}(t) \cup \text{fv}(ts)
  \]

- What are the free variables of a set?
  
  \[
  \text{fv}(S) \overset{\text{def}}{=} \bigcup_{t \in S} \text{fv}(t)
  \]
What are the free variables of a function, for example the identity function?
What are the free variables of a function, for example the identity function?

But you just told me what the free variables of pairs and sets are. The identity function can be seen as the set of pairs (inputs and outputs):

\{(x, x), (y, y), (z, z), \ldots, (t_1 t_2, t_1, t_2), \ldots\}

This would imply that the free variables of $\lambda x.x$ is the set of all variables?!
We like to have an (overloaded) definition recursing over the type hierarchy.

Starting with definitions for the base types (such as natural numbers, strings and the object languages we want to study).

Then for type-formers where the definition should depend on earlier defined notions:

\[ \text{fv}(t_1, t_2) \overset{\text{def}}{=} \text{fv}(t_1) \cup \text{fv}(t_2) \]

\[ \text{fv}([]) \overset{\text{def}}{=} \emptyset \]

\[ \text{fv}(t :: ts) \overset{\text{def}}{=} \text{fv}(t) \cup \text{fv}(ts) \]

But what shall we do about functions, \( \tau \Rightarrow \sigma \)?
Atoms

- We start with a countably infinite set of atoms.
  - They will be used for object language variables.
  - They are the ‘things’ that can be bound.
We start with a countably infinite set of atoms. They will be used for object language variables. They are the ‘things’ that can be bound.

We restrict ourselves here to just one kind of atoms.
We start with a countably infinite set of *atoms*. They will be used for object language variables. They are the ‘things’ that can be bound.

We restrict ourselves here to just one kind of atoms.

**Permutations** are lists of pairs of atoms:

\[(a_1, b_1) \ldots (a_n, b_n)\]
Permutations

A permutation acts on atoms as follows:

\[
\begin{align*}
\text{[]} \cdot a & \overset{\text{def}}{=} a \\
((a_1 \, a_2) :: \pi) \cdot a & \overset{\text{def}}{=} \\
& \begin{cases} 
    a_1 & \text{if } \pi \cdot a = a_2 \\
    a_2 & \text{if } \pi \cdot a = a_1 \\
    \pi \cdot a & \text{otherwise}
\end{cases}
\end{align*}
\]

- \text{[]} stands for the empty list (the identity permutation), and
- \((a_1 \, a_2) :: \pi\) stands for the permutation \(\pi\) followed by the swapping \((a_1 \, a_2)\). (We usually drop the ::.)
Permutations (2)

- **Composition**: The composition of two permutations is given by list-concatenation, written as $\pi' \circ \pi$.

- **Inverse**: The inverse of a permutation is given by list reversal, written as $\pi^{-1}$, and

- **Permutation Equality**: Two permutations $\pi$ and $\pi'$ are equal iff
  \[
  \pi \sim \pi' \overset{\text{def}}{=} \forall a. \pi \cdot a = \pi' \cdot a
  \]
Permutations (2)

- **Composition** of two permutations is given by list-concatenation, written as $\pi' @ \pi$.

- **Inverse** of a permutation is given by list reversal, written as $\pi^{-1}$, and

- **Permutation equality**, two permutations $\pi$ and $\pi'$ are equal iff
  \[ \pi \sim \pi' \overset{\text{def}}{=} \forall a. \pi \cdot a = \pi' \cdot a \]

- Example calculations:
  \[(b d)(b c)(a c) \cdot a = d\]
the composition of two permutations is given by list-concatenation, written as $\pi' @ \pi$,

the inverse of a permutation is given by list reversal, written as $\pi^{-1}$, and

permutation equality, two permutations $\pi$ and $\pi'$ are equal iff

$$\pi \sim \pi' \overset{\text{def}}{=} \forall a. \ \pi \cdot a = \pi' \cdot a$$

Example calculations:

$$(b \ d)(b \ c)(a \ c)^{-1} = (a \ c)(b \ c)(b \ d)$$
the composition of two permutations is given by list-concatenation, written as $\pi' @ \pi$,

the inverse of a permutation is given by list reversal, written as $\pi^{-1}$, and

permutation equality, two permutations $\pi$ and $\pi'$ are equal iff

\[ \pi \sim \pi' \overset{\text{def}}{=} \forall a. \pi \cdot a = \pi' \cdot a \]

Example calculations:

\[ (a \ a) \sim [] \]
Three Properties

We require of all permutation operations that:

1. $[] \cdot x = x$
2. $(\pi_1 @ \pi_2) \cdot x = \pi_1 \cdot (\pi_2 \cdot x)$
3. If $\pi_1 \sim \pi_2$ then $\pi_1 \cdot x = \pi_2 \cdot x$. 
Three Properties

We require of all permutation operations that:

- $[] \cdot x = x$
- $(\pi_1 @ \pi_2) \cdot x = \pi_1 \cdot (\pi_2 \cdot x)$
- If $\pi_1 \sim \pi_2$ then $\pi_1 \cdot x = \pi_2 \cdot x$.

From this we have:

- $\pi^{-1} \cdot (\pi \cdot x) = x$
- $\pi \cdot x_1 = x_2$ if and only if $x_1 = \pi^{-1} \cdot x_2$
- $x_1 = x_2$ if and only if $\pi \cdot x_1 = \pi \cdot x_2$
Permutations on $\lambda$-Terms

$\pi \cdot (a)$

$\pi \cdot (t_1 \ t_2) \overset{\text{def}}{=} (\pi \cdot t_1)(\pi \cdot t_2)$

$\pi \cdot (\lambda a.\ t) \overset{\text{def}}{=} \lambda(\pi \cdot a).\ (\pi \cdot t)$

given by the action on atoms
Permutations on $\lambda$-Terms

$\pi \cdot (a)$
$\pi \cdot (t_1 \, t_2) \overset{\text{def}}{=} (\pi \cdot t_1) (\pi \cdot t_2)$
$\pi \cdot (\lambda a. t) \overset{\text{def}}{=} \lambda (\pi \cdot a). (\pi \cdot t)$

given by the action on atoms

we treat lambdas as if there were no binders
Permutations on $\lambda$-Terms

$\pi \cdot (a)$ given by the action on atoms

$\pi \cdot (t_1 \, t_2) \overset{\text{def}}{=} (\pi \cdot t_1)(\pi \cdot t_2)$

$\pi \cdot (\lambda a. \, t) \overset{\text{def}}{=} \lambda (\pi \cdot a). (\pi \cdot t)$

An aside: This definition leads also to a simple definition of $\alpha$-equivalence:

\[
\frac{t_1 \approx t_2}{\lambda a. t_1 \approx \lambda a. t_2}
\]

\[
\begin{align*}
a \neq b & \quad t_1 \approx (a \, b) \cdot t_2 & \quad a \not\# t_2 \\
\frac{}{}{} & \quad \lambda a. t_1 \approx \lambda b. t_2
\end{align*}
\]
Permutations on $\lambda$-Terms

$\pi \cdot (a)$ given by the action on atoms

$\pi \cdot (t_1 \ t_2) \overset{\text{def}}{=} (\pi \cdot t_1)(\pi \cdot t_2)$

$\pi \cdot (\lambda a. \ t) \overset{\text{def}}{=} \lambda (\pi \cdot a). (\pi \cdot t)$

An aside: This definition leads also to a simple definition of $\alpha$-equivalence:

\[
\begin{align*}
\frac{t_1 = t_2}{\lambda a. t_1 = \lambda a. t_2}
\end{align*}
\]

\[
\begin{align*}
a \neq b \quad t_1 = (a \ b) \cdot t_2 \quad a \ # \ t_2
\frac{}{\lambda a. t_1 = \lambda b. t_2}
\end{align*}
\]
Perm’s for Other Types

- \( \pi \cdot (x_1, x_2) \overset{\text{def}}{=} (\pi \cdot x_1, \pi \cdot x_2) \)  
  pairs

- \( \pi \cdot [] \overset{\text{def}}{=} [] \)  
  lists

- \( \pi \cdot (x :: xs) \overset{\text{def}}{=} (\pi \cdot x) :: (\pi \cdot xs) \)

- \( \pi \cdot X \overset{\text{def}}{=} \{ \pi \cdot x \mid x \in X \} \)  
  sets

- \( \pi \cdot[\lambda x.N]_\alpha = [\lambda(\pi \cdot x).(\pi \cdot N)]_\alpha \)

- \( \pi \cdot f \overset{\text{def}}{=} \lambda x.\pi \cdot (f \ (\pi^{-1} \cdot x)) \)  
  functions

- \( \pi \cdot x \overset{\text{def}}{=} x \)  
  integers, strings, bools

Perm’s for Other Types

\[(\pi \cdot f) (\pi \cdot x) \overset{\text{def}}{=} (\lambda x. \pi \cdot (f (\pi^{-1} \cdot x))) (\pi \cdot x) \]
\[= \pi \cdot (f (\pi^{-1} \cdot (\pi \cdot x))) \]
\[= \pi \cdot (f \ x) \]

\[\pi \cdot (x :: xs) = (\pi \cdot x) :: (\pi \cdot xs) \]

\[\pi \cdot X \overset{\text{def}}{=} \{ \pi \cdot x \mid x \in X \} \]

\[\pi \cdot [\lambda x. N]_{\alpha} = [\lambda (\pi \cdot x). (\pi \cdot N)]_{\alpha} \]

\[\pi \cdot f \overset{\text{def}}{=} \lambda x. \pi \cdot (f (\pi^{-1} \cdot x)) \]
\[\pi \cdot f (\pi \cdot x) = (\pi \cdot f) (\pi \cdot x) \]

\[\pi \cdot x \overset{\text{def}}{=} x \]

sets

functions

integers, strings, bools
Perm’s for Other Types

- \( \pi \cdot (x_1, x_2) \overset{\text{def}}{=} (\pi \cdot x_1, \pi \cdot x_2) \)  
  
  *pairs*

- \( \pi \cdot [] \overset{\text{def}}{=} [] \)  
  
  *lists*

- \( \pi \cdot (x :: x::s) \overset{\text{def}}{=} (\pi \cdot x) :: (\pi \cdot x::s) \)  
  
- \( \pi \cdot X \overset{\text{def}}{=} \{ \pi \cdot x \mid x \in X \} \)  
  
  *sets*

- \( \pi \cdot \lambda x . N \overset{\alpha}{=} \lambda (\pi \cdot x) . (\pi \cdot N) \)  
  
  \( \pi \cdot [\lambda x . N] \overset{\alpha}{=} [\lambda (\pi \cdot x) . (\pi \cdot N)] \)

- \( \pi \cdot f \overset{\text{def}}{=} \lambda x . \pi \cdot (f \ (\pi^{-1} \cdot x)) \)  
  
  *functions*

- \( \pi \cdot x \overset{\text{def}}{=} x \)  
  
  *integers, strings, bools*
Support and Freshness

The support of an object $x$ is a set of atoms:

$$\text{supp}(x) \overset{\text{def}}{=} \{ a \mid \text{infinite } \{ b \mid (a b) \cdot x \neq x \} \}$$

$$a \not\in x \overset{\text{def}}{=} a \not\in \text{supp}(x)$$

In words: all atoms $a$ where the set

$$\{ b \mid (a b) \cdot x \neq x \}$$

is infinite (each swapping $(a b)$ needs to change something in $x$).
Support and Freshness

The support of an object \( x \) is a set of atoms:

\[
\text{supp}(x) \overset{\text{def}}{=} \{ a \mid \text{infinite} \{ b \mid (a \ b) \cdot x \neq x \} \}
\]

\[
a \neq x \overset{\text{def}}{=} a \not\in \text{supp}(x)
\]

OK, this definition is a tiny bit complicated, so let's go slowly...

In words:

\[
\{ b \mid (a \ b) \cdot x \neq x \}
\]

is infinite (each swapping \((a \ b)\) needs to change something in \(x\)).
Support of an Atom

What is the support of the atom \( c \)?

\[
\text{supp}(c) \overset{\text{def}}{=} \{ a \mid \text{infinite} \{ b \mid (a \cdot b) \cdot c \neq c \} \}
\]

Let’s check the (infinitely many) atoms one by one:
Support of an Atom

What is the support of the atom \( c \)?

\[
\text{supp}(c) \overset{\text{def}}{=} \{ a \mid \text{infinite } \{ b \mid (a \, b) \cdot c \neq c \} \}
\]

Let's check the (infinitely many) atoms one by one:

\[
a: (a \, ?) \cdot c \neq c
\]
Support of an Atom

What is the support of the atom $c$?

$$\text{supp}(c) \overset{\text{def}}{=} \{ a \mid \text{infinite} \{ b \mid (a \ b) \cdot c \neq c \} \}$$

Let’s check the (infinitely many) atoms one by one:

- $a$: $(a \ ?) \cdot c \neq c$ no
- $b$: $(b \ ?) \cdot c \neq c$
What is the support of the atom $c$?

$$\text{supp}(c) \overset{\text{def}}{=} \{ a \mid \text{infinite } \{ b \mid (a b) \cdot c \neq c \} \}$$

Let’s check the (infinitely many) atoms one by one:

- $a$: $(a ?) \cdot c \neq c$ no
- $b$: $(b ?) \cdot c \neq c$ no
- $c$: $(c ?) \cdot c \neq c$ yes
Support of an Atom

What is the support of the atom $c$?

$$\text{supp}(c) \overset{\text{def}}{=} \{ a \mid \text{infinite } \{ b \mid (ab)c \neq c \} \}$$

Let's check the (infinitely many) atoms one by one:

- $a$: $(a?)c \neq c$ no
- $b$: $(b?)c \neq c$ no
- $c$: $(c?)c \neq c$ yes
- $d$: $(d?)c \neq c$
Support of an Atom

What is the support of the atom $c$?

$$\text{supp}(c) \overset{\text{def}}{=} \{ a \mid \text{infinite} \{ b \mid (ab) \cdot c \neq c \} \}$$

Let's check the (infinitely many) atoms one by one:

- $a$: $(a?) \cdot c \neq c$ no
- $b$: $(b?) \cdot c \neq c$ no
- $c$: $(c?) \cdot c \neq c$ yes
- $d$: $(d?) \cdot c \neq c$ no
- $\vdots$ no
Support of an Atom

What is the support of the atom $c$?

$$\text{supp}(c) \overset{\text{def}}{=} \{ a \mid \text{infinite} \{ b \mid (a \ b) \cdot c \neq c \} \}$$

Let's check the (infinitely many) atoms one by one:

$$\text{supp}(c) = \{ c \}$$

- $a$: $(a \ ?) \cdot c \neq c$ no
- $b$: $(b \ ?) \cdot c \neq c$ no
- $c$: $(c \ ?) \cdot c \neq c$ yes
- $d$: $(d \ ?) \cdot c \neq c$ no
  
  ... no

Support of a Pair

\[
\text{supp}(t_1, t_2) \overset{\text{def}}{=} \{ a \mid \inf \{ b \mid (a \cdot b) \cdot (t_1, t_2) \neq (t_1, t_2) \} \}
\]
Support of a Pair

\[ \text{supp}(t_1, t_2) \overset{\text{def}}{=} \{ a \mid \inf \{ b \mid (a b) \cdot (t_1, t_2) \neq (t_1, t_2) \} \} \]

\[ \{ a \mid \inf \{ b \mid ((a b) \cdot t_1, (a b) \cdot t_2) \neq (t_1, t_2) \} \} \]
Support of a Pair

\[
\text{supp}(t_1, t_2) \stackrel{\text{def}}{=} \{ a | \inf \{ b | (a b) \cdot (t_1, t_2) \neq (t_1, t_2) \} \}
\]

\[
\{ a | \inf \{ b | ((a b) \cdot t_1, (a b) \cdot t_2) \neq (t_1, t_2) \} \}
\]

We know

\[(t_1, t_2) = (s_1, s_2) \iff t_1 = s_1 \land t_2 = s_2\]

hence

\[(t_1, t_2) \neq (s_1, s_2) \iff t_1 \neq s_1 \lor t_2 \neq s_2\]
Support of a Pair

\[ \text{supp}(t_1, t_2) \overset{\text{def}}{=} \{ a \mid \inf \{ b \mid (a b) \cdot (t_1, t_2) \neq (t_1, t_2) \} \} \]

\[ \{ a \mid \inf \{ b \mid ((a b) \cdot t_1, (a b) \cdot t_2) \neq (t_1, t_2) \} \} \]

\[ \{ a \mid \inf \{ b \mid (a b) \cdot t_1 \neq t_1 \lor (a b) \cdot t_2 \neq t_2 \} \} \]
Support of a Pair

\[ \text{supp}(t_1, t_2) \overset{\text{def}}{=} \{ a \mid \inf \{ b \mid (ab) \cdot (t_1, t_2) \neq (t_1, t_2) \} \} \]

\[ \{ a \mid \inf \{ b \mid ((ab) \cdot t_1, (ab) \cdot t_2) \neq (t_1, t_2) \} \} \]
\[ \{ a \mid \inf \{ b \mid (ab) \cdot t_1 \neq t_1 \lor (ab) \cdot t_2 \neq t_2 \} \} \]
\[ \{ a \mid \inf(\{ b \mid (ab) \cdot t_1 \neq t_1 \} \cup \{ b \mid (ab) \cdot t_2 \neq t_2 \}) \} \]
Support of a Pair

\[ \text{supp}(t_1,t_2) \overset{\text{def}}{=} \{ a \mid \inf \{ b \mid (a \ b) \cdot (t_1,t_2) \neq (t_1,t_2) \} \} \]

\[ \{ a \mid \inf \{ b \mid ((a \ b) \cdot t_1, (a \ b) \cdot t_2) \neq (t_1,t_2) \} \} \]

\[ \{ a \mid \inf \{ b \mid (a \ b) \cdot t_1 \neq t_1 \lor (a \ b) \cdot t_2 \neq t_2 \} \} \]

\[ \{ a \mid \inf(\{ b \mid (a \ b) \cdot t_1 \neq t_1 \} \cup \{ b \mid (a \ b) \cdot t_2 \neq t_2 \} ) \} \]

\[ \{ a \mid \inf \{ b \mid (a \ b) \cdot t_1 \neq t_1 \} \lor \inf \{ b \mid (a \ b) \cdot t_2 \neq t_2 \} \} \]
Support of a Pair

\[ \text{supp}(t_1, t_2) \overset{\text{def}}{=} \{ a \mid \inf \{ b \mid (a \ b) \cdot (t_1, t_2) \neq (t_1, t_2) \} \} \]

\[
\{ a \mid \inf \{ b \mid ((a \ b) \cdot t_1, (a \ b) \cdot t_2) \neq (t_1, t_2) \} \}
\]

\[
\{ a \mid \inf \{ b \mid (a \ b) \cdot t_1 \neq t_1 \lor (a \ b) \cdot t_2 \neq t_2 \} \}
\]

\[
\{ a \mid \inf (\{ b \mid (a \ b) \cdot t_1 \neq t_1 \} \cup \{ b \mid (a \ b) \cdot t_2 \neq t_2 \}) \}
\]

\[
\{ a \mid \inf \{ b \mid (a \ b) \cdot t_1 \neq t_1 \} \lor \inf \{ b \mid (a \ b) \cdot t_2 \neq t_2 \} \}
\]

\[
\{ a \mid \inf \{ b \mid (a \ b) \cdot t_1 \neq t_1 \} \} \cup \{ a \mid \inf \{ b \mid (a \ b) \cdot t_2 \neq t_2 \} \}
\]
Support of a Pair

$$\text{supp}(t_1, t_2) \overset{\text{def}}{=} \{ a \mid \inf \{ b \mid (a \ b) \cdot (t_1, t_2) \neq (t_1, t_2) \} \}$$

$$\{ a \mid \inf \{ b \mid ((a \ b) \cdot t_1, (a \ b) \cdot t_2) \neq (t_1, t_2) \} \}$$

$$\{ a \mid \inf \{ b \mid (a \ b) \cdot t_1 \neq t_1 \lor (a \ b) \cdot t_2 \neq t_2 \} \}$$

$$\{ a \mid \inf (\{ b \mid (a \ b) \cdot t_1 \neq t_1 \} \cup \{ b \mid (a \ b) \cdot t_2 \neq t_2 \} ) \}$$

$$\{ a \mid \inf \{ b \mid (a \ b) \cdot t_1 \neq t_1 \} \lor \inf \{ b \mid (a \ b) \cdot t_2 \neq t_2 \} \}$$

$$\{ a \mid \inf \{ b \mid (a \ b) \cdot t_1 \neq t_1 \} \} \cup \{ a \mid \inf \{ b \mid (a \ b) \cdot t_2 \neq t_2 \} \}$$

$$\text{supp}(t_1) \quad \cup \quad \text{supp}(t_2)$$
Support of a Pair

\[ \text{supp}(t_1, t_2) \overset{\text{def}}{=} \{ a \mid \inf \{ b \mid (a b) \cdot (t_1, t_2) \neq (t_1, t_2) \} \} \]

\[
\{ a \mid \inf \{ b \mid (a b) \cdot t_1 \neq t_1 \} \}
\]
\[
\{ a \mid \inf \{ b \mid (a b) \cdot t_2 \neq t_2 \} \}
\]

So \( \text{supp}(t_1, t_2) = \text{supp}(t_1) \cup \text{supp}(t_2) \).

However, such things are proved for you: the user does not have to bother with them.
lemma
shows "supp \((t_1, t_2)\) = supp \(t_1 \cup ((supp \ t_2)::atom \ set)\)"
proof -
have "supp \((t_1, t_2)\) = \{a. inf \{b. [(a,b)]\cdot(t_1,t_2) \neq (t_1,t_2)\}\}"
  by (simp add: supp_def)
also have "... = \{a. inf \{b. [(a,b)]\cdot t_1,[(a,b)]\cdot t_2 \neq (t_1,t_2)\}\}" by simp
also have "... = \{a. inf \{b. [(a,b)]\cdot t_1 \neq t_1 \lor [(a,b)]\cdot t_2 \neq t_2\}\}" by simp
also have "... = \{a. inf \{b. [(a,b)]\cdot t_1 \neq t_1\} \cup \{b. [(a,b)]\cdot t_2 \neq t_2\}\}" 
  by (simp only: Collect_disj_eq)
also have "... = \{a. (inf \{b. [(a,b)]\cdot t_1 \neq t_1\}) \lor (inf \{b. [(a,b)]\cdot t_2 \neq t_2\})\}" 
  by simp
also have "... = \{a. inf \{b. [(a,b)]\cdot t_1 \neq t_1\}\} \cup \{a. inf \{b. [(a,b)]\cdot t_2 \neq t_2\}\}" 
  by (simp only: Collect_disj_eq)
also have "... = supp \(t_1 \cup supp t_2\)" by (simp add: supp_def)
finally show "supp \((t_1,t_2)\) = supp \(t_1 \cup ((supp t_2)::atom \ set)\)" by simp
qed
lemma shows \[ \text{supp} (t_1,t_2) = \text{supp} t_1 \cup ((\text{supp} t_2)::\text{atom set}) \]

proof -

have \[ \text{supp} (t_1,t_2) = \{a. \inf \{b. [(a,b)]\cdot(t_1,t_2) \neq (t_1,t_2)\}\} \]
  by (simp add: supp_def)

also have \[ ... = \{a. \inf \{b. [(a,b)]\cdot t_1,[(a,b)]\cdot t_2 \neq (t_1,t_2)\}\} \]
  by simp

also have \[ ... = \{a. \inf \{b. [(a,b)]\cdot t_1 \neq t_1 \lor [(a,b)]\cdot t_2 \neq t_2\}\} \]
also have \[ ... = \{a. \inf \{b. [(a,b)]\cdot t_1 \neq t_1 \} \cup \{b. [(a,b)]\cdot t_2 \neq t_2\}\}\]
  by (simp only: Collect_disj_eq)

also have \[ ... = \{a. \inf \{b. [(a,b)]\cdot t_1 \neq t_1\}\lor (\inf \{b. [(a,b)]\cdot t_2 \neq t_2\})\} \]
  by simp

also have \[ ... = \{a. \inf \{b. [(a,b)]\cdot t_1 \neq t_1\}\}\cup \{a. \inf \{b. [(a,b)]\cdot t_2 \neq t_2\}\} \]
  by (simp only: Collect_disj_eq)

also have \[ ... = \text{supp} t_1 \cup \text{supp} t_2 \] by (simp add: supp_def)

finally show \[ \text{supp} (t_1,t_2) = \text{supp} t_1 \cup ((\text{supp} t_2)::\text{atom set}) \] by simp

qed
lemma
  shows "\( \text{supp } (t_1, t_2) = \text{supp } t_1 \cup ((\text{supp } t_2)\cdot\text{atom set}) \)"
proof -
  have "\( \text{supp } (t_1, t_2) = \{ a. \inf \{ b. [(a,b)]\cdot(t_1, t_2) \neq (t_1, t_2) \} \} \)"
    by (simp add: supp_def)
  also have "\( ... = \{ a. \inf \{ b. ([(a,b)]\cdot t_1, [(a,b)]\cdot t_2) \neq (t_1, t_2) \} \)" by simp
  also have "\( ... = \{ a. \inf \{ b. [(a,b)]\cdot t_1 \neq t_1 \lor [(a,b)]\cdot t_2 \neq t_2 \} \)" by simp
  also have "\( ... = \{ a. \inf \{ b. [(a,b)]\cdot t_1 \neq t_1 \} \cup \{ b. [(a,b)]\cdot t_2 \neq t_2 \} \} \)"
    by (simp only: Collect_disj_eq)
  also have "\( ... = \{ a. (\inf \{ b. [(a,b)]\cdot t_1 \neq t_1 \}) \lor (\inf \{ b. [(a,b)]\cdot t_2 \neq t_2 \}) \} \)"
    by simp
  also have "\( ... = \{ a. \inf \{ b. [(a,b)]\cdot t_1 \neq t_1 \} \}\cup \{ a. \inf \{ b. [(a,b)]\cdot t_2 \neq t_2 \} \} \)"
    by (simp only: Collect_disj_eq)
  also have "\( ... = \text{supp } t_1 \cup \text{supp } t_2 \)" by (simp add: supp_def)
  finally show "\( \text{supp } (t_1, t_2) = \text{supp } t_1 \cup ((\text{supp } t_2)\cdot\text{atom set}) \)" by simp
qed
lemma shows "\( \text{supp} (t_1,t_2) = \text{supp} t_1 \cup ((\text{supp} t_2)::\text{atom set}) \)"

proof -

have "\( \text{supp} (t_1,t_2) = \{a. \inf \{b. [(a,b)]\cdot(t_1,t_2) \neq (t_1,t_2)\}\} \)"
  by (simp add: \text{supp}\_def)
also have "\( \ldots = \{a. \inf \{b. [((a,b)]\cdot t_1,[(a,b)]\cdot t_2) \neq (t_1,t_2)\}\} \)" by simp
also have "\( \ldots = \{a. \inf \{b. [((a,b)]\cdot t_1 \neq t_1 \lor [(a,b)]\cdot t_2 \neq t_2)\}\} \)" by simp
also have "\( \ldots = \{a. \inf (\{b. [(a,b)]\cdot t_1 \neq t_1\} \cup \{b. [(a,b)]\cdot t_2 \neq t_2\})\}\)"
  by (simp only: \text{Collect}\_disj\_eq)
also have "\( \ldots = \{a. (\inf \{b. [(a,b)]\cdot t_1 \neq t_1\}) \lor (\inf \{b. [(a,b)]\cdot t_2 \neq t_2\})\} \)"
  by simp
also have "\( \ldots = \{a. \inf \{b. [(a,b)]\cdot t_1 \neq t_1\} \cup a. \inf \{b. [(a,b)]\cdot t_2 \neq t_2\}\} \)"
  by (simp only: \text{Collect}\_disj\_eq)
also have "\( \ldots = \text{supp} t_1 \cup \text{supp} t_2 \)" by (simp add: \text{supp}\_def)

finally show "\( \text{supp} (t_1,t_2) = \text{supp} t_1 \cup ((\text{supp} t_2)::\text{atom set}) \)" by simp

qed
lemma shows "\(\text{supp} (t_1,t_2) = \text{supp} t_1 \cup ((\text{supp} t_2)\::\text{atom set})\)"

proof -

have "\(\text{supp} (t_1,t_2) = \{a. \inf \{b. [(a,b)]\cdot (t_1,t_2) \neq (t_1,t_2)\}\}\)"
  by (simp add: supp_def)
also have "... = \{a. \inf \{b. [(a,b)]\cdot t_1, [(a,b)]\cdot t_2 \neq (t_1,t_2)\}\}" by simp
also have "... = \{a. \inf \{b. [(a,b)]\cdot t_1 \neq t_1 \lor [(a,b)]\cdot t_2 \neq t_2\}\}" by simp
also have "... = \{a. \inf (\{b. [(a,b)]\cdot t_1 \neq t_1\} \cup \{b. [(a,b)]\cdot t_2 \neq t_2\})\}"
  by (simp only: Collect_disj_eq)
also have "... = \{a. (\inf \{b. [(a,b)]\cdot t_1 \neq t_1\}) \lor (\inf \{b. [(a,b)]\cdot t_2 \neq t_2\})\}" by simp
also have "... = \{a. \inf \{b. [(a,b)]\cdot t_1 \neq t_1\}\} \cup \{a. \inf \{b. [(a,b)]\cdot t_2 \neq t_2\}\}"
  by (simp only: Collect_disj_eq)
also have "... = \text{supp} t_1 \cup \text{supp} t_2" by (simp add: supp_def)
finally show "\(\text{supp} (t_1,t_2) = \text{supp} t_1 \cup ((\text{supp} t_2)\::\text{atom set})\)" by simp
qed
lemma shows "supp \( (t_1, t_2) \) = supp \( t_1 \) \( \cup \) ((supp \( t_2 \)::atom set)"

proof -

have "supp \( (t_1, t_2) \) = \{a. \inf \{b. [(a,b)][t_1, [(a,b)][t_2) \neq (t_1, t_2)]\}\}"
  by (simp add: supp_def)
also have "... = \{a. \inf \{b. [(a,b)][t_1 \neq t_1 \lor [(a,b)][t_2 \neq t_2])\}" by simp
also have "... = \{a. \inf (\{b. [(a,b)][t_1 \neq t_1 \} \cup \{b. [(a,b)][t_2 \neq t_2])\}"
  by (simp only: Collect_disj_eq)
also have "... = \{a. (\inf \{b. [(a,b)][t_1 \neq t_1])\lor (\inf \{b. [(a,b)][t_2 \neq t_2])\}" by simp
also have "... = \{a. \inf \{b. [(a,b)][t_1 \neq t_1])\} \cup \{a. \inf \{b. [(a,b)][t_2 \neq t_2])\}"
  by (simp only: Collect_disj_eq)
also have "... = supp \( t_1 \) \( \cup \) supp \( t_2 \)" by (simp add: supp_def)
finally show "supp \( (t_1, t_2) \) = supp \( t_1 \) \( \cup \) ((supp \( t_2 \)::atom set)" by simp

qed
lemma
  shows "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::atom set)"
proof -
  have "supp (t₁,t₂) = \{a. inf \{b. [(a,b)]•(t₁,t₂) \neq (t₁,t₂)\}\}"
    by (simp add: supp_def)
  also have "... = \{a. inf \{b. ([(a,b)]•t₁,[(a,b)]•t₂) \neq (t₁,t₂)\}\}" by simp
  also have "... = \{a. inf \{b. [(a,b)]•t₁ \neq t₁ \lor [(a,b)]•t₂ \neq t₂\}\}" by simp
  also have "... = \{a. inf ((\{b. [(a,b)]•t₁ \neq t₁\} \cup \{b. [(a,b)]•t₂ \neq t₂\}))\}"
     by (simp only: Collect_disj_eq)
  also have "... = \{a. (inf \{b. [(a,b)]•t₁ \neq t₁\}) \lor (inf \{b. [(a,b)]•t₂ \neq t₂\})\}" by simp
  also have "... = \{a. inf \{b. [(a,b)]•t₁ \neq t₁\}\} \cup \{a. inf \{b. [(a,b)]•t₂ \neq t₂\}\}" by (simp only: Collect_disj_eq)
  also have "... = supp t₁ \cup supp t₂" by (simp add: supp_def)
finally show "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::atom set)" by simp
qed
Lemma shows \( \text{supp} (t_1, t_2) = \text{supp} t_1 \cup ((\text{supp} t_2) :: \text{atom set}) \)

Proof -

Have \( \text{supp} (t_1, t_2) = \{a. \inf \{b. [(a,b)] \cdot (t_1, t_2) \neq (t_1, t_2)\}\} \)

by (simp add: \text{supp_def})

Also have \( \ldots = \{a. \inf \{b. [(a,b)] \cdot t_1, [(a,b)] \cdot t_2 \neq (t_1, t_2)\}\} \) by simp

Also have \( \ldots = \{a. \inf \{b. [(a,b)] \cdot t_1 \neq t_1 \lor [(a,b)] \cdot t_2 \neq t_2\}\} \) by simp

Also have \( \ldots = \{a. \inf \{b. [(a,b)] \cdot t_1 \neq t_1\} \cup \{b. [(a,b)] \cdot t_2 \neq t_2\}\} \)

by (simp only: Collect_disj_eq)

Also have \( \ldots = \{a. (\inf \{b. [(a,b)] \cdot t_1 \neq t_1\}) \lor (\inf \{b. [(a,b)] \cdot t_2 \neq t_2\})\} \)

by simp

Also have \( \ldots = \{a. \inf \{b. [(a,b)] \cdot t_1 \neq t_1\}\} \cup \{a. \inf \{b. [(a,b)] \cdot t_2 \neq t_2\}\} \)

by (simp only: Collect_disj_eq)

Also have \( \ldots = \text{supp} t_1 \cup \text{supp} t_2 \) by (simp add: \text{supp_def})

Finally show \( \text{supp} (t_1, t_2) = \text{supp} t_1 \cup ((\text{supp} t_2) :: \text{atom set}) \) by simp

QED
Lemma: $a \not\equiv x \land b \not\equiv x \Rightarrow (a \cdot b) \cdot x = x$
Lemma: \( a \not\!\!\!\# x \land b \not\!\!\!\# x \Rightarrow (a \ b) \cdot x = x \)

Proof: case \( a = b \) clear.
Lemma: \( a \not\# x \land b \not\# x \Rightarrow (a \cdot b) \cdot x = x \)

Proof: case \( a \neq b \):

(1) \( \text{fin}\{c \mid (a \cdot c) \cdot x \neq x\} \)  
(2) \( \text{fin}\{c \mid (b \cdot c) \cdot x \neq x\} \)  

from Ass. + Def. of \( \# \)

\[
\begin{align*}
\text{supp}(x) & \overset{\text{def}}{=} \{a \mid \inf\{c \mid (a \cdot c) \cdot x \neq x\}\} \\
\text{supp}(x) & \overset{\text{def}}{=} \{a \mid \inf\{c \mid (a \cdot c) \cdot x \neq x\}\} \\
\end{align*}
\]
Lemma: \( a \not= x \land b \not= x \Rightarrow (a b) \cdot x = x \)

Proof: case \( a \neq b \):

(1) \( \text{fin}\{ c \mid (a c) \cdot x \neq x \} \)
\( \text{fin}\{ c \mid (b c) \cdot x \neq x \} \)

from Ass. + Def. of \( \not= \)

(2) \( \text{fin}(\{ c \mid (a c) \cdot x \neq x \} \cup \{ c \mid (b c) \cdot x \neq x \}) \) \text{ f'rm (1)}

(3) \( (a c) \cdot x = x \) by (4i)

(4) by bij.

(5) by bij.

(6) \( (a b) \cdot x = x \) by prop. of perms

Done.
Lemma: $a \# x \land b \# x \Rightarrow (a \ b) \cdot x = x$

Proof: case $a \neq b$:

(1) $\text{fin}\{c \mid (a \ c) \cdot x \neq x\}$  \hspace{1cm} \text{from Ass. + Def. of } \# \\
    \hspace{1cm} \text{fin}\{c \mid (b \ c) \cdot x \neq x\}$

(2) $\text{fin}\{c \mid (a \ c) \cdot x \neq x \lor (b \ c) \cdot x \neq x\}$  \hspace{1cm} \text{f'rm (1)}
Lemma: \( a \# x \land b \# x \Rightarrow (a b) \cdot x = x \)

Proof: case \( a \neq b \):

1. \( \text{fin}\{c \mid (a c) \cdot x \neq x\} \)  
2. \( \text{fin}\{c \mid (b c) \cdot x \neq x\} \)  
3. \( \text{fin}\{c \mid (a c) \cdot x \neq x \lor (b c) \cdot x \neq x\} \)  
4. \( \text{fin}\{c \mid \neg((a c) \cdot x \neq x \lor (b c) \cdot x \neq x)\} \)

Given a finite set of atoms, its 'co-set' must be infinite.
Lemma: \( a \# x \land b \# x \Rightarrow (a b) \cdot x = x \)

Proof: case \( a \neq b \):

(1) \( \text{fin}\{c \mid (a c) \cdot x \neq x\} \quad \text{fin}\{c \mid (b c) \cdot x \neq x\} \)

from Ass. + Def. of \# 

(2) \( \text{fin}\{c \mid (a c) \cdot x \neq x \lor (b c) \cdot x \neq x\} \)

f'rm (1)

(3) \( \text{inf}\{c \mid (a c) \cdot x = x \land (b c) \cdot x = x\} \)

f'rm (2)
Lemma: \( a \# x \land b \# x \Rightarrow (a b) \cdot x = x \)

Proof: case \( a \neq b \):

1. \( \text{fin}\{ c \mid (a c) \cdot x \neq x \} \)
   \( \text{fin}\{ c \mid (b c) \cdot x \neq x \} \) from Ass. + def. of \#  

2. \( \text{fin}\{ c \mid (a c) \cdot x \neq x \lor (b c) \cdot x \neq x \} \) from (1)  

3. \( \text{inf}\{ c \mid (a c) \cdot x = x \land (b c) \cdot x = x \} \) from (2)  

4. (i) \( (a c) \cdot x = x \) (ii) \( (b c) \cdot x = x \) for a \( c \in (3) \)

If a set is infinite, it must contain a few elements. Let’s pick \( c \).
Lemma: $a \# x \land b \# x \Rightarrow (a b) \cdot x = x$

Proof: case $a \neq b$:

1. $\text{fin}\{c \mid (a c) \cdot x \neq x\}$
2. $\text{fin}\{c \mid (b c) \cdot x \neq x\}$

from Ass. + Def. of $\#$

3. $\text{fin}\{c \mid (a c) \cdot x \neq x \lor (b c) \cdot x \neq x\}$ f'rm (1)

4. (i) $(a c) \cdot x = x$ (ii) $(b c) \cdot x = x$ for a $c \in (3)$ by (4i)

5. $(a c) \cdot x = x$
Lemma: $a \# x \land b \# x \Rightarrow (a\; b) \cdot x = x$

Proof: case $a \neq b$:

1. $\text{fin}\{ c \mid (a\; c) \cdot x \neq x \}$
   $\text{fin}\{ c \mid (b\; c) \cdot x \neq x \}$

2. $\text{fin}\{ c \mid (a\; c) \cdot x \neq x \lor (b\; c) \cdot x \neq x \}$

3. $\text{inf}\{ c \mid (a\; c) \cdot x = x \land (b\; c) \cdot x = x \}$

4. (i) $(a\; c) \cdot x = x$  (ii) $(b\; c) \cdot x = x$  for a $c \in (3)$

5. $(a\; c) \cdot x = x$

6. $(b\; c) \cdot (a\; c) \cdot x = (b\; c) \cdot x$

bij.: $x = y$ iff $\pi \cdot x = \pi \cdot y$
Lemma: \( a \not\equiv x \land b \not\equiv x \Rightarrow (ab) \cdot x = x \)

Proof: case \( a \neq b \):

(1) \( \text{fin} \{ c | (ac) \cdot x \neq x \} \)
\( \text{fin} \{ c | (bc) \cdot x \neq x \} \)

from Ass. + Def. of \( \not\equiv \)

(2) \( \text{fin} \{ c | (ac) \cdot x \neq x \lor (bc) \cdot x \neq x \} \)

f'rm (1)

(3) \( \text{inf} \{ c | (ac) \cdot x = x \land (bc) \cdot x = x \} \)

f'rm (2)

(4) (i) \( (ac) \cdot x = x \) (ii) \( (bc) \cdot x = x \)

for a \( c \in (3) \)

(5) \( (ac) \cdot x = x \)

by (4i)

(6) \( (bc) \cdot (ac) \cdot x = x \)

by bij.,(4ii)
Lemma: \( a \not\# x \land b \not\# x \Rightarrow (a \ b) \cdot x = x \)

Proof: case \( a \neq b \):

1. \( \text{fin}\{c \mid (a \ c) \cdot x \neq x\} \) \quad \text{from Ass. +Def. of \#}
2. \( \text{fin}\{c \mid (a \ c) \cdot x \neq x \lor (b \ c) \cdot x \neq x\} \) \quad \text{f'rm (1)}
3. \( \text{inf}\{c \mid (a \ c) \cdot x = x \land (b \ c) \cdot x = x\} \) \quad \text{f'rm (2)}
4. (i) \( (a \ c) \cdot x = x \) \quad \text{for a } c \in (3) \quad \text{by (4i)}
   (ii) \( (b \ c) \cdot x = x \)
5. \( (a \ c) \cdot x = x \) \quad \text{by bij. (4ii)}
6. \( (b \ c) \cdot (a \ c) \cdot x = x \)
7. \( (a \ c) \cdot (b \ c) \cdot (a \ c) \cdot x = (a \ c) \cdot x \) \quad \text{by bij.}
Lemma: $a \not\approx x \land b \not\approx x \Rightarrow (a \cdot b) \cdot x = x$

Proof: case $a \neq b$:

1. $\text{fin}\{c \mid (a \cdot c) \cdot x \neq x\}$
   $\text{fin}\{c \mid (b \cdot c) \cdot x \neq x\}$
   from Ass. + Def. of $\#$

2. $\text{fin}\{c \mid (a \cdot c) \cdot x \neq x \lor (b \cdot c) \cdot x \neq x\}$
   from (1)

3. $\text{inf}\{c \mid (a \cdot c) \cdot x = x \land (b \cdot c) \cdot x = x\}$
   from (2)

4. (i) $(a \cdot c) \cdot x = x$ (ii) $(b \cdot c) \cdot x = x$
   for a $c \in (3)$

5. $(a \cdot c) \cdot x = x$
   by (4i)

6. $(b \cdot c) \cdot (a \cdot c) \cdot x = x$
   by bij., (4ii)

7. $(a \cdot c) \cdot (b \cdot c) \cdot (a \cdot c) \cdot x = x$
   by bij., (4i)
Lemma: \( a \# x \land b \# x \Rightarrow (ab) \cdot x = x \)

Proof: case \( a \neq b \):

(1) \( \text{fin}\{c \mid (ac) \cdot x \neq x\} \)  
\( \text{fin}\{c \mid (bc) \cdot x \neq x\} \)  
from Ass. + Def. of \# 

(2) \( \text{fin}\{c \mid (ac) \cdot x \neq x \lor (bc) \cdot x \neq x\} \)  
f'rm (1)

(3) \( \text{inf}\{c \mid (ac) \cdot x = x \land (bc) \cdot x = x\} \)  
f'rm (2) 

(4) (i) \( (ac) \cdot x = x \) (ii) \( (bc) \cdot x = x \)  
for a \( c \in (3) \) 

(5) \( (ac) \cdot x = x \)  
by (4i)

(6) \( (bc) \cdot (ac) \cdot x = x \)  
by bij.,(4ii)

(7) \( (ac) \cdot (bc) \cdot (ac) \cdot x = x \)  
by bij.,(4i)

\( (ac)(bc)(ac) \cdot a = b \)  
\( (ac)(bc)(ac) \cdot b = a \)  
\( (ac)(bc)(ac) \cdot c = c \)
Lemma: \( a \not= x \land b \not= x \Rightarrow (a \, b) \cdot x = x \)

Proof: case \( a \neq b\):

(1) \( \text{fin}\{ c \mid (a \, c) \cdot x \neq x \} \) \quad \text{from Ass. +Def. of \( \# \)}

(2) \( \text{fin}\{ c \mid (b \, c) \cdot x \neq x \} \)

(2) \( \text{property of permutation:} \)

(3) \( \text{inf } \pi_1 \sim \pi_2 \Rightarrow \pi_1 \cdot x = \pi_2 \cdot x = x \) \quad \text{f'rm (2)}

(4) (i) \( (a \, c) \cdot x = x \) \quad (ii) \( (b \, c) \cdot x = x \) \quad \text{for a \( c \in (3) \)}

(5) \( (a \, c) \cdot x = x \) \quad \text{by (4i)}

(6) \( (b \, c) \cdot (a \, c) \cdot x = x \) \quad \text{by bij. (4ii)}

(7) \( (a \, c) \cdot (b \, c) \cdot (a \, c) \cdot x = x \) \quad \text{by bij. (4i)}

(8) \( (a \, b) \cdot x = x \) \quad \text{by prop. of perms}
Lemma: $a \neq x \land b \neq x \Rightarrow (a \ b) \cdot x = x$

Proof: case $a \neq b$:

1. $\text{fin}\{c \mid (a \ c) \cdot x \neq x\}$
   $\text{fin}\{c \mid (b \ c) \cdot x \neq x\}$ from Ass. + Def. of $\neq$

2. $\text{fin}\{c \mid (a \ c) \cdot x \neq x \lor (b \ c) \cdot x \neq x\}$ f'rm (1)

3. $\text{inf}\{c \mid (a \ c) \cdot x = x \land (b \ c) \cdot x = x\}$ f'rm (2)

4. (i) $(a \ c) \cdot x = x$ (ii) $(b \ c) \cdot x = x$ for a $c \in (3)$

5. $(a \ c) \cdot x = x$ by (4i)

6. $(b \ c) \cdot (a \ c) \cdot x = x$ by bij.,(4ii)

7. $(a \ c) \cdot (b \ c) \cdot (a \ c) \cdot x = x$ by bij.,(4i)

8. $(a \ b) \cdot x = x$ by prop. of perms

Done.
Existence of a Fresh Atom

Q: Why do we assume that there are countably infinitely many atoms?

A: For any finitely supported $x$:

$$\exists a. \ a \not= x$$

If something is finitely supported, then we can always choose a fresh atom (also for finitely supported functions).
Exercises about Support

- Given a finite set of atoms. What is the support of this set?
- What is the support of the set of all atoms?
- From the set of all atoms take one atom out. What is the support of the resulting set?
- Are there any sets of atoms that have infinite support?

“Support by Andrew Pitts”
In Daily Use there Is Nothing Scary about Support

- We usually restrict ourselves to finitary structures (lists, lambda-terms, etc). In those structures, the notion of support coincides with the usual notion of what the free variables are.

- We just have to be careful with sets and functions (we treat them on a case-by-case basis and they usually turn out to have empty support).
In Daily Use there Is Nothing Scary about Support

- We usually restrict ourselves to finitary structures (lists, lambda-terms, etc). In those structures, the notion of support coincides with the usual notion of what the free variables are.

- We just have to be careful with sets and functions (we treat them on a case-by-case basis and they usually turn out to have empty support).

- There are two reasons for wanting to find out what the free variables of functions are: when we define functions over the “structure” of $\alpha$-equivalence classes and because of a trick.
Nominal Abstractions

We are now going to specify what abstraction ‘abstractly’ means: it is an operation

\[
\_ \cdot (\_) : \text{atom} \Rightarrow \text{trm} \Rightarrow \text{trm}
\]

and has to satisfy two properties:

- \( \pi \cdot ([a].x) = [\pi \cdot a].(\pi \cdot x) \)
- \([a].x = [b].y \) iff
  \[
  (a = b \land x = y) \lor \\
  (a \neq b \land x = (a \cdot b) \cdot y \land a \not\# y)
  \]

These two properties imply for finitely supported \( x \)

\[
\text{supp}([a].x) = \text{supp}(x) - \{a\}
\]
We are now going to specify what abstraction ‘abstractly’ means: it is an operation

\[ \mathsf{[\_].(\_)} : \text{atom} \Rightarrow \text{trm} \Rightarrow \text{trm} \]

and has to satisfy two properties:

1. \( \pi \cdot ([a].x) = [\pi \cdot a].(\pi \cdot x) \)
2. \([a].x = [b].y \) iff
   \[(a = b \land x = y) \lor (a \neq b \land x = (a b) \cdot y \land a \# y) \]

These two properties imply for finitely supported \( x \)

\[ \text{supp}([a].x) = \text{supp}(x) - \{a\} \]
Nominal Abstractions

We are now going to specify what abstraction ‘abstractly’ means: it is an operation

\[
[\_].(\_) : \text{atom} \Rightarrow \text{trm} \Rightarrow \text{trm}
\]

and has to satisfy two properties:

1. \(\pi \cdot ([a].x) = [\pi \cdot a].(\pi \cdot x)\)
2. \([a].x = [b].y \iff (a = b \land x = y) \lor (a \neq b \land x = (ab) \cdot y \land a \# y)\)

These two properties imply for finitely supported \(x\)

\[\text{supp}([a].x) = \text{supp}(x) - \{a\}\]
Function \( [a].t \equiv [\lambda a.t]_\alpha \)

\[
[a].t \overset{\text{def}}{=} (\lambda b. \text{if } a = b \\
\text{then Some}(t) \\
\text{else if } b \neq t \text{ then Some}((b \circ a) \cdot t) \text{ else None})
\]

type: \text{atom} \rightarrow \text{trm option}
Function $[a].t \mathrel{\equiv} \[\lambda a. t\]_\alpha$

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

This is supposed to stand for the $\alpha$-equivalence class of $\lambda a. t$. 
Function \([a].t \equiv [\lambda a.t]_\alpha\)

\[ [a].(a, c) \overset{\text{def}}{=} \]

\[
(\lambda b. \text{if } a = b \\
\quad \text{then Some}(a, c) \\
\quad \text{else if } b \not= (a, c) \\
\quad \quad \text{then Some}((b a) \cdot (a, c)) \text{ else None})
\]

Let's check this for \([a].(a, c)\):
Function \([a] . t \equiv \lambda a . t\)\(_\alpha\)

\[ [a] . (a, c) \overset{\text{def}}{=} \]

\((\lambda b . \text{if } a = b \text{ then Some}(a, c) \text{ else if } b \neq (a, c) \text{ then Some}((b a) \cdot (a, c)) \text{ else None})\)

Let's check this for \([a] . (a, c)\):

\(a \text{ 'applied to' } [a] . (a, c) \text{ 'gives' } \text{Some}(a, c)\)
Function $[a].t \equiv [\lambda a.t]_\alpha$

$$[a].(a, c) \equiv \lambda b. \text{if } a = b \text{ then Some}(a, c) \text{ else if } b \neq (a, c) \text{ then Some}((b a) \cdot (a, c)) \text{ else None}$$

Let's check this for $[a].(a, c)$:

- $a$ applied to $[a].(a, c)$ gives Some$(a, c)$
- $b$ applied to $[a].(a, c)$ gives Some$(b, c)$
Function \([a].t \equiv [\lambda a.t]_\alpha\)

\([a].(a, c) \overset{\text{def}}{=} (\lambda b. \text{if } a = b \text{ then } \text{Some}(a, c) \text{ else if } b \neq (a, c) \text{ then } \text{Some}((b \cdot a) \cdot (a, c))) \text{ else None})\)

Let's check this for \([a].(a, c)\):

- **a** 'applied to' \([a].(a, c)\) 'gives' \text{Some}(a, c)
- **b** 'applied to' \([a].(a, c)\) 'gives' \text{Some}(b, c)
- **c** 'applied to' \([a].(a, c)\) 'gives' \text{None}
Function \([a].t \equiv [\lambda a.t]_\alpha\)

\([a].(a, c) \overset{\text{def}}{=} \lambda b. \begin{cases} \text{if } a = b & \text{then Some}(a, c) \\ \text{else if } b \not= (a, c) & \text{then Some}((b \cdot a) \cdot (a, c)) \text{ else None} \end{cases}\)

Let’s check this for \([a].(a, c)\):

- **a** ’applied to’ \([a].(a, c)\) ’gives’ Some\((a, c)\)
- **b** ’applied to’ \([a].(a, c)\) ’gives’ Some\((b, c)\)
- **c** ’applied to’ \([a].(a, c)\) ’gives’ None
- **d** ’applied to’ \([a].(a, c)\) ’gives’ Some\((d, c)\)

;
Let's check this for \([a].(a, c)\):

\[ a \text{ 'applied to' } [a].(a, c) \text{ 'gives' } \text{Some}(a, c) \quad \lambda a.(a \ c) \]
\[ b \text{ 'applied to' } [a].(a, c) \text{ 'gives' } \text{Some}(b, c) \quad \lambda b.(b \ c) \]
\[ c \text{ 'applied to' } [a].(a, c) \text{ 'gives' } \text{None} \]
\[ d \text{ 'applied to' } [a].(a, c) \text{ 'gives' } \text{Some}(d, c) \quad \lambda d.(d \ c) \]
Function \([a].t \equiv [\lambda a.t]_\alpha\)

\([a].(a, c) \equiv \)

\[(\lambda b. \text{if } a = b \)
  \text{ then } \text{Some}(a, c)
  \text{ else if } b \neq (a, c)
  \text{ then } \text{Some}((b a) \cdot (a, c)) \text{ else None}\]

Let's check this for \([a].(a, c)\):

- \(a\) 'applied to' \([a].(a, c)\) 'gives' \(\text{Some}(a, c)\)
- \(b\) 'applied to' \([a].(a, c)\) 'gives' \(\text{Some}(b, c)\)
- \(c\) 'applied to' \([a].(a, c)\) 'gives' \(\text{None}\)
- \(d\) 'applied to' \([a].(a, c)\) 'gives' \(\text{Some}(d, c)\)
  
  ...
Function \([a].t \equiv [\lambda a.t]_\alpha\)

\([a].t \overset{\text{def}}{=} (\lambda b.\text{if } a = b \text{ then Some}(t) \text{ else if } b \neq t \text{ then Some}((b a) \cdot t) \text{ else None})\)

This function ‘takes’ a lambda-abstraction and an atom, and tries to rename the abstraction according to the given atom.
Function $[a].t \equiv [\lambda a.t]_\alpha$

$[a].t \overset{\text{def}}{=} (\lambda b. \begin{cases} 
                     \text{if } a = b 
                     & \text{then Some}(t) \\
                     \text{else if } b \# t 
                     & \text{then Some}((b\ a)\cdot t) \\
                     \text{else None}
                 \end{cases})$

This function ‘takes’ a lambda-abstraction and an atom, and tries to rename the abstraction according to the given atom.
We can now define inductively named $\alpha$-equivalence classes of lambda-terms:

```plaintext
datatype pre_lam =
    Var "atom"
  | App "pre_lam" "pre_lam"
  | Lam "atom $\Rightarrow$ pre_lam option"
```

big set: pre_lam
Definition of Small Set

small set: $\Lambda_\alpha$

big set: pre_lam

\[
\begin{align*}
\text{Var } a & \in \Lambda_\alpha \\
\text{App } t_1 \, t_2 & \in \Lambda_\alpha \\
\text{Lam } [a].t & \in \Lambda_\alpha
\end{align*}
\]
Definition of Small Set

small set: $\Lambda_\alpha$

new type: lam

big set: pre_lam

\[ \text{Var } a \in \Lambda_\alpha \]
\[ t_1 \in \Lambda_\alpha \quad t_2 \in \Lambda_\alpha \]
\[ \text{App } t_1 \quad t_2 \in \Lambda_\alpha \]
\[ t \in \Lambda_\alpha \]
\[ \text{Lam } [a] \cdot t \in \Lambda_\alpha \]
Definition of Small Set

small set: $\Lambda_\alpha$

bijection

big set: pre_lam

t new type: lam

Var $a \in \Lambda_\alpha$

$t_1 \in \Lambda_\alpha$  $t_2 \in \Lambda_\alpha$

$\text{App } t_1 t_2 \in \Lambda_\alpha$

$t \in \Lambda_\alpha$

$Lam[a].t \in \Lambda_\alpha$

This means we have the familiar induction principle for $\Lambda_\alpha$ and so also for $\Lambda/\approx$. 
Structural Induction

\[
\begin{align*}
\text{Var } a &\in \Lambda_{\alpha} & t_1 &\in \Lambda_{\alpha} & t_2 &\in \Lambda_{\alpha} \\
\text{App } t_1 & t_2 &\in \Lambda_{\alpha}
\end{align*}
\]

\[
\begin{align*}
t &\in \Lambda_{\alpha} \\
\text{Lam } [a].t &\in \Lambda_{\alpha}
\end{align*}
\]

...implies the structural induction principle over the type lam:

\[
\begin{align*}
\forall a. \ P \ (\text{Var } a) \\
\forall t_1 \ t_2. \ P \ t_1 \land P \ t_2 &\Rightarrow P \ (\text{App } t_1 \ t_2) \\
\forall a \ t. \ P \ t &\Rightarrow P \ (\text{Lam } [a].t)
\end{align*}
\]

\[P \ t\]
Better Structural Induction

\[ \forall a. \, P (\text{Var} \, a) \]
\[ \forall t_1 \, t_2. \, P \, t_1 \land P \, t_2 \Rightarrow P (\text{App} \, t_1 \, t_2) \]
\[ \forall a \, t. \, P \, t \Rightarrow P (\text{Lam} \, [a].t) \]

\[ \forall a \, t. \, P \, t \]

implies (as seen yesterday)

\[ \forall a \, c. \, P \, c (\text{Var} \, a) \]
\[ \forall t_1 \, t_2 \, c. \, (\forall d. P \, d \, t_1) \land (\forall d. P \, d \, t_2) \Rightarrow P \, c (\text{App} \, t_1 \, t_2) \]
\[ \forall a \, t \, c. \, a \neq c \land (\forall d. P \, d \, t) \Rightarrow P \, c (\text{Lam} \, [a].t) \]

\[ P \, c \, t \]

provided \( c \) is finitely supported
We define permutation operations for “everything” (the notion of support is then fixed)
Big Picture

- We define permutation operations for “everything” (the notion of support is then fixed)
- Construct the “model” for alpha-equivalence classes (this gives us a weak induction principle)
Big Picture

- We define permutation operations for "everything" (the notion of support is then fixed)
- Construct the "model" for alpha-equivalence classes (this gives us a weak induction principle)
- Derive a strong induction principle.
We define permutation operations for "everything" (the notion of support is then fixed)

Construct the "model" for alpha-equivalence classes (this gives us a weak induction principle)

Derive a strong induction principle.

Derive a recursion combinator.
nominal_datatype lam =
  Var "name"
| App "lam" "lam"
| Lam "«name»lam" ("Lam [__].__")

lemma alpha_test:
  shows "Lam [x].Var x = Lam [y].Var y"
  by (simp add: lam.inject alpha swap_simps fresh_atm)
"All" for Free

nominal_datatype lam =
  Var "name"
| App "lam" "lam"
| Lam "«name»lam" ("Lam [__].__")

lemma alpha_test:
  shows "Lam [x].Var x = Lam [y].Var y"
  by (simp add: lam.inject alpha swap_simps fresh_atm)

thm lam.inject[no_vars]
  (Var x1 = Var y1) = (x1 = y1)
  (App x2 x1 = App y2 y1) = (x2 = y2 ∧ x1 = y1)
  (Lam x1 x2 = Lam y1 y2) = ([x1].x2 = [y1].y2)
nominal_datatype lam =
   Var "name"
| App "lam" "lam"
| Lam "«name›lam" ("Lam [__].__")

lemma alpha_test:
   shows "Lam [x].Var x = Lam [y].Var y"
by (simp add: lam.inject alpha swap_simps fresh_atm)

thm alpha[no_vars]
([a].x = [b].y) =
   (a = b ∧ x = y ∨ a ≠ b ∧ x = [(a, b)] · y ∧ a ≠ y)
nominal_datatype lam =
  Var "name"
| App "lam" "lam"
| Lam "«name»lam" ("Lam [__].__")

lemma alpha_test:
  shows "Lam [x].Var x = Lam [y].Var y"
by (simp add: lam.inject alpha swap_simps fresh_atm)

thm swap_simps[no_vars]
[(a, b)] :: a = b
[a ≠ c; b ≠ c] ⇒ [(a, b)] :: c = c
"All" for Free

nominal_datatype lam =
  Var "name"
| App "lam" "lam"
| Lam "«name»lam" ("Lam [__].__")

lemma alpha_test:
  shows "Lam [x].Var x = Lam [y].Var y"
by (simp add: lam.inject alpha swap_simps fresh_atm)

thm fresh_atm[no_vars]
  a ≠ b = (a ≠ b)
The support of an object \( x \) is a set of atoms:

\[
\text{supp}(x) \overset{\text{def}}{=} \{ a \mid \text{infinite} \{ b \mid (a b) \cdot x \neq x \} \}
\]

\( a \neq x \overset{\text{def}}{=} a \not\in \text{supp}(x) \)
Support and Supports

The support of an object \( x \) is a set of atoms:

\[
\text{supp}(x) \overset{\text{def}}{=} \{ a \mid \text{infinite} \{ b \mid (a \ b) \cdot x \neq x \} \}
\]

\[
a \# x \overset{\text{def}}{=} a \not\in \text{supp}(x)
\]

\[
S \text{ supports } x \overset{\text{def}}{=}
\forall ab. a \not\in S \land b \not\in S \Rightarrow (a \ b) \cdot x = x
\]

if \( S \) is finite and \( S \text{ supports } x \) then \( \text{supp}(x) \subseteq S \)
Some Exercises

Given a finite set of atoms. What is the support of this set?
Given a finite set of atoms. What is the support of this set? If $S$ is finite, then $\text{supp}(S) = S$. 

To find the support of a set, you look for the elements that are contained within it. If a set is finite, its support is the set itself. If you remove an element from a set, the support of the resulting set is the set of all elements that were in the original set except for the one you removed.
Some Exercises

- Given a finite set of atoms. What is the support of this set? If \( S \) is finite, then \( \text{supp}(S) = S \).

- What is the support of the set of all atoms?
Some Exercises

Given a finite set of atoms. What is the support of this set? If $S$ is finite, then $\text{supp}(S) = S$.

What is the support of the set of all atoms? Let $A = \{a_0, a_1 \ldots\}$, then $\text{supp}(A) = \emptyset$. 
Some Exercises

- Given a finite set of atoms. What is the support of this set? If $S$ is finite, then $\text{supp}(S) = S$.

- What is the support of the set of all atoms? Let $A = \{a_0, a_1, \ldots\}$, then $\text{supp}(A) = \emptyset$.

- From the set of all atoms take one atom out. What is the support of the resulting set?
Some Exercises

Given a finite set of atoms. What is the support of this set? If \( S \) is finite, then \( \text{supp}(S) = S \).

What is the support of the set of all atoms?
Let \( A = \{a_0, a_1 \ldots\} \), then \( \text{supp}(A) = \emptyset \).

From the set of all atoms take one atom out. What is the support of the resulting set?
\( \text{supp}(A - \{a\}) = \{a\} \).
Some Exercises

- Given a finite set of atoms. What is the support of this set? If $S$ is finite, then $\text{supp}(S) = S$.

- What is the support of the set of all atoms? Let $A = \{a_0, a_1 \ldots\}$, then $\text{supp}(A) = \emptyset$.

- From the set of all atoms take one atom out. What is the support of the resulting set? $\text{supp}(A - \{a\}) = \{a\}$.

- Are there any sets of atoms that have infinite support?
Some Exercises

- Given a finite set of atoms. What is the support of this set? If $S$ is finite, then $\text{supp}(S) = S$.

- What is the support of the set of all atoms? Let $A = \{a_0, a_1 \ldots\}$, then $\text{supp}(A) = \emptyset$.

- From the set of all atoms take one atom out. What is the support of the resulting set? $\text{supp}(A - \{a\}) = \{a\}$.

- Are there any sets of atoms that have infinite support? If both $S$ and $A - S$ are infinite then $\text{supp}(S) = A$. 
The support of $\lambda x. x$:

\[
\pi \cdot \lambda x. x \overset{\text{def}}{=} \lambda x. \pi \cdot ((\lambda x. x) (\pi^{-1} \cdot x)) \\
= \lambda x. \pi \cdot \pi^{-1} \cdot x \\
= \lambda x. x
\]
The support of $\lambda x.x$:

\[ \pi \cdot \lambda x.x \overset{\text{def}}{=} \lambda x.\pi \cdot ((\lambda x.x) (\pi^{-1} \cdot x)) \]
\[ = \lambda x.\pi \cdot \pi^{-1} \cdot x \]
\[ = \lambda x.x \]

Therefore

\[ \text{supp}(\lambda x.x) \overset{\text{def}}{=} \{ a \mid \text{infinite}\{ b \mid (a b) \cdot \lambda x.x \neq \lambda x.x \} \} \]
\[ = \{ a \mid \text{infinite}\{ b \mid \lambda x.x \neq \lambda x.x \} \} \]
\[ = \emptyset \]
Existence of a Fresh Atom

For any finitely supported $x$:

$$\exists a. \ a \neq x$$

If something is finitely supported, then we can always choose a fresh atom (also for finitely supported functions).
Existence of a Fresh Atom

For any finitely supported $x$:

$$\exists a. \, a \neq x$$

If something is finitely supported, then we can always choose a fresh atom (also for finitely supported functions).

Andy in Nominal Logic assumed everything is finitely supported. This clashes with the choice principle, or Hilbert’s $\epsilon$. 