Nominal Techniques

or, Something Crazy about Free Variables

Christian Urban (TU Munich)

http://isabelle.in.tum.de/nominal/

Free Variables of Lambda-Terms:

$$\begin{aligned} \mathsf{fv}(x) &= \{x\} \\ \mathsf{fv}(t_1\, t_2) &= \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2) \\ \mathsf{fv}(\lambda x.t) &= \mathsf{fv}(t) - \{x\} \end{aligned}$$

What are the free variables of pairs, sets, functions...?

Informal Reasoning

Fluet: “Expressions differing only in names of bound variables are equivalent.”

Harper and Pfenning about contexts: “...when we write Γ,x:A we assume that x is not already declared in Γ. If necessary, we tacitly rename x before adding it to the context Γ.”

Pfenning in Logical Frameworks - A Brief Introduction: “We allow tacit α-conversion (renaming of bound variables) ...”

Plan

- How do we get a type for lambda-terms where we have the equation

$$\lambda x.x = \lambda y.y?$$

- For this we will have a closer look at the notion of free variables and describe abstractly what abstractions are. (Lots of fun!)

A Non-Starter

- If we define

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

  then we do not have $\lambda x.x = \lambda y.y$.

- 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

$$M \approx_\alpha M', N \approx_\alpha N' \Rightarrow M[x:=N] \approx_\alpha M'[x:=N']$$

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.

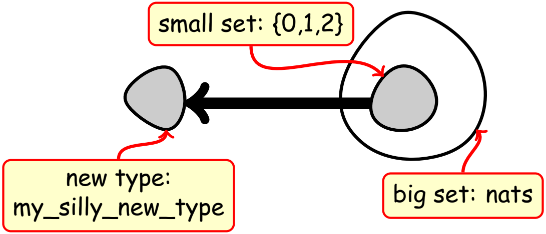
```
typedef my_silly_new_type = "{0, 1, 2::nat}"
  by auto
```

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
  ```



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** α-equivalence classes.

```
nominal_datatype lam =
    Var "name"
  | App "lam" "lam"
  | Lam "«name»lam"
```

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 α.

```
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 α.

```
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.

Second Naive Attempt

- We like to define

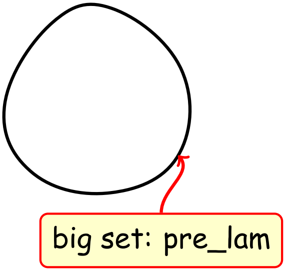
```
datatype pre_lam =
   Var "name"
 | App "pre_lam" "pre_lam"
 | Lam "(name × pre_lam) set"
```

Second Naive Attempt

- 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

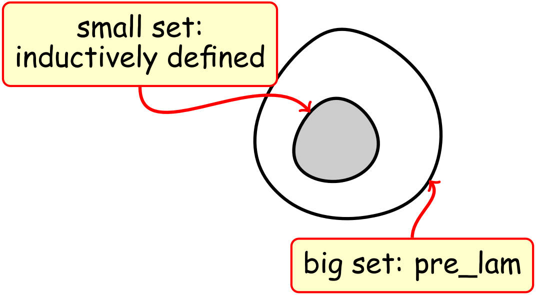


Second Naive Attempt

- 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

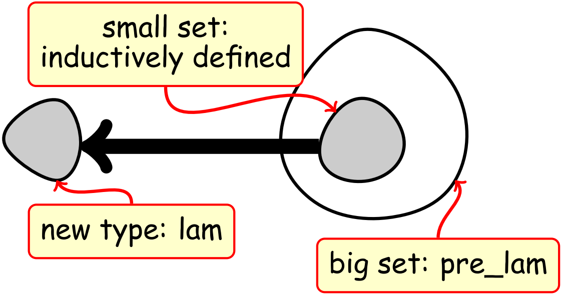


Second Naive Attempt

- 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



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"
```

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"
```

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"
```

- In the following we will make this **idea** to work by finding an alternative representation for α-equivalence classes.

Free Variables

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

Free Variables

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

$$
\begin{aligned}
\mathsf{fv}(a) &\stackrel{\text{def}}{=} \{a\} \\
\mathsf{fv}(t_1\, t_2) &\stackrel{\text{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2) \\
\mathsf{fv}(\lambda a.t) &\stackrel{\text{def}}{=} \mathsf{fv}(t) - \{a\}
\end{aligned}
$$

Free Variables

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

$$\begin{aligned} \mathsf{fv}(a) &\stackrel{\text{def}}{=} \{a\} \\ \mathsf{fv}(t_1\, t_2) &\stackrel{\text{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2) \\ \mathsf{fv}(\lambda a.t) &\stackrel{\text{def}}{=} \mathsf{fv}(t) - \{a\} \end{aligned}$$

- What are the free variables of a pair?

Free Variables

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

$$\begin{aligned}\mathsf{fv}(a) &\stackrel{\text{def}}{=} \{a\}\\ \mathsf{fv}(t_1\,t_2) &\stackrel{\text{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)\\ \mathsf{fv}(\lambda a.t) &\stackrel{\text{def}}{=} \mathsf{fv}(t) - \{a\}\end{aligned}$$

- What are the free variables of a pair?

$$\mathsf{fv}(t_1, t_2) \stackrel{\text{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)$$

Free Variables

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

$$\begin{aligned} \mathsf{fv}(a) &\stackrel{\mathsf{def}}{=} \{a\} \\ \mathsf{fv}(t_1\, t_2) &\stackrel{\mathsf{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2) \\ \mathsf{fv}(\lambda a.t) &\stackrel{\mathsf{def}}{=} \mathsf{fv}(t) - \{a\} \end{aligned}$$

- What are the free variables of a pair?

$$\mathsf{fv}(t_1, t_2) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)$$

- What are the free variables of a list?

Free Variables

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

$$\mathsf{fv}(a) \stackrel{\text{def}}{=} \{a\}$$
$$\mathsf{fv}(t_1\, t_2) \stackrel{\text{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)$$
$$\mathsf{fv}(\lambda a.t) \stackrel{\text{def}}{=} \mathsf{fv}(t) - \{a\}$$

- What are the free variables of a pair?

$$\mathsf{fv}(t_1, t_2) \stackrel{\text{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)$$

- What are the free variables of a list?

$$\mathsf{fv}([]) \stackrel{\text{def}}{=} \varnothing \qquad \mathsf{fv}(t :: ts) \stackrel{\text{def}}{=} \mathsf{fv}(t) \cup \mathsf{fv}(ts)$$

Free Variables

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

$$\mathsf{fv}(a) \stackrel{\mathsf{def}}{=} \{a\}$$
$$\mathsf{fv}(t_1\, t_2) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)$$
$$\mathsf{fv}(\lambda a.t) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t) - \{a\}$$

- What are the free variables of a pair?

$$\mathsf{fv}(t_1, t_2) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)$$

- What are the free variables of a list?

$$\mathsf{fv}([]) \stackrel{\mathsf{def}}{=} \varnothing \qquad \mathsf{fv}(t :: ts) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t) \cup \mathsf{fv}(ts)$$

- What are the free variables of a set?

Free Variables

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

$$\mathsf{fv}(a) \stackrel{\mathsf{def}}{=} \{a\}$$
$$\mathsf{fv}(t_1\, t_2) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)$$
$$\mathsf{fv}(\lambda a.t) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t) - \{a\}$$

- What are the free variables of a pair?

$$\mathsf{fv}(t_1, t_2) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)$$

- What are the free variables of a list?

$$\mathsf{fv}([]) \stackrel{\mathsf{def}}{=} \varnothing \qquad \mathsf{fv}(t :: ts) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t) \cup \mathsf{fv}(ts)$$

- What are the free variables of a set?

$$\mathsf{fv}(S) \stackrel{\mathsf{def}}{=} \bigcup_{t \in S} \mathsf{fv}(t)$$

Free Variables (2)

- What are the free variables of a function, for example the identity function?

Free Variables (2)

- 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?!

Free Variables (3)

- 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:

$$\mathsf{fv}(t_1, t_2) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)$$

$$\mathsf{fv}([]) \stackrel{\mathsf{def}}{=} \varnothing$$

$$\mathsf{fv}(t :: ts) \stackrel{\mathsf{def}}{=} \mathsf{fv}(t) \cup \mathsf{fv}(ts)$$

  - 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.

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.

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:

$$[]\bullet a \stackrel{\text{def}}{=} a$$

$$((a_1\, a_2) :: \pi)\bullet a \stackrel{\text{def}}{=} \begin{cases} a_1 & \text{if } \pi\bullet a = a_2 \\ a_2 & \text{if } \pi\bullet a = a_1 \\ \pi\bullet a & \text{otherwise} \end{cases}$$

- $[]$ stands for the empty list (the identity permutation), and
- $(a_1\, a_2) :: \pi$ stands for the permutation π followed by the swapping $(a_1\, a_2)$. (We usually drop the $::$.)

Permutations (2)

- 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 π^{-1}, and
- **permutation equality**, two permutations π and π' are equal iff

$$\pi \sim \pi' \stackrel{\text{def}}{=} \forall a.\ \pi \bullet a = \pi' \bullet a$$

Permutations (2)

- 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 π^{-1}, and
- **permutation equality**, two permutations π and π' are equal iff

$$\pi \sim \pi' \stackrel{\text{def}}{=} \forall a.\ \pi \bullet a = \pi' \bullet a$$

- Example calculations:

$$(b\,d)(b\,c)(a\,c) \bullet a = d$$

Permutations (2)

- 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 π^{-1}, and
- **permutation equality**, two permutations π and π' are equal iff

$$\pi \sim \pi' \stackrel{\text{def}}{=} \forall a.\ \pi \bullet a = \pi' \bullet a$$

- Example calculations:

$$(b\,d)(b\,c)(a\,c)^{-1} = (a\,c)(b\,c)(b\,d)$$

Permutations (2)

- 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 π^{-1}, and
- **permutation equality**, two permutations π and π' are equal iff

$$\pi \sim \pi' \stackrel{\text{def}}{=} \forall a.\ \pi \bullet a = \pi' \bullet a$$

- Example calculations:

$$(a\,a) \sim []$$

Three Properties

We require of all permutation operations that:

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

Three Properties

We require of all permutation operations that:

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

From this we have:

- $\pi^{-1}\bullet(\pi\bullet x) = x$
- $\pi\bullet x_1 = x_2$ if and only if $x_1 = \pi^{-1}\bullet x_2$
- $x_1 = x_2$ if and only if $\pi\bullet x_1 = \pi\bullet x_2$

Permutations on λ-Terms

$$\begin{array}{rcl} \pi\bullet(a) & & \text{given by the action on atoms} \\ \pi\bullet(t_1\,t_2) & \stackrel{\text{def}}{=} & (\pi\bullet t_1)(\pi\bullet t_2) \\ \pi\bullet(\lambda a.t) & \stackrel{\text{def}}{=} & \lambda(\pi\bullet a).(\pi\bullet t) \end{array}$$

Permutations on λ-Terms

$$\pi\bullet(a) \quad \text{given by the action on atoms}$$

$$\pi\bullet(t_1\,t_2) \stackrel{\text{def}}{=} (\pi\bullet t_1)(\pi\bullet t_2)$$

$$\pi\bullet(\lambda a.t) \stackrel{\text{def}}{=} \lambda(\pi\bullet a).(\pi\bullet t)$$

we treat lambdas as if there were no binders

Permutations on λ-Terms

$$\pi \bullet (a) \quad \text{given by the action on atoms}$$

$$\pi \bullet (t_1\, t_2) \stackrel{\text{def}}{=} (\pi \bullet t_1)(\pi \bullet t_2)$$

$$\pi \bullet (\lambda a.t) \stackrel{\text{def}}{=} \lambda(\pi \bullet a).(\pi \bullet t)$$

An aside: This definition leads also to a simple definition of α-equivalence:

$$\frac{t_1 \approx t_2}{\lambda a.t_1 \approx \lambda a.t_2}$$

$$\frac{a \neq b \quad t_1 \approx (a\, b) \bullet t_2 \quad a \# t_2}{\lambda a.t_1 \approx \lambda b.t_2}$$

Permutations on λ-Terms

$$\pi\bullet(a) \quad \text{given by the action on atoms}$$

$$\pi\bullet(t_1\,t_2) \stackrel{\text{def}}{=} (\pi\bullet t_1)(\pi\bullet t_2)$$

$$\pi\bullet(\lambda a.t) \stackrel{\text{def}}{=} \lambda(\pi\bullet a).(\pi\bullet t)$$

An aside: This definition leads also to a simple definition of α-equivalence:

$$\frac{t_1 = t_2}{\lambda a.t_1 = \lambda a.t_2}$$

$$\frac{a \neq b \quad t_1 = (a\,b)\bullet t_2 \quad a \# t_2}{\lambda a.t_1 = \lambda b.t_2}$$

Perm's for Other Types

- $\pi \bullet (x_1, x_2) \stackrel{\text{def}}{=} (\pi \bullet x_1, \pi \bullet x_2)$ pairs
- $\pi \bullet [] \stackrel{\text{def}}{=} []$ lists

  $\pi \bullet (x :: xs) \stackrel{\text{def}}{=} (\pi \bullet x) :: (\pi \bullet xs)$
- $\pi \bullet X \stackrel{\text{def}}{=} \{\pi \bullet x \mid x \in X\}$ sets

  $\pi \bullet [\lambda x.N]_\alpha = [\lambda(\pi \bullet x).(\pi \bullet N)]_\alpha$
- $\pi \bullet f \stackrel{\text{def}}{=} \lambda x.\pi \bullet (f\ (\pi^{-1} \bullet x))$ functions

  $\pi \bullet (f\ x) = (\pi \bullet f)\ (\pi \bullet x)$
- $\pi \bullet x \stackrel{\text{def}}{=} x$ integers, strings, bools

Perm's for Other Types

$$
\begin{aligned}
(\pi \bullet f)\ (\pi \bullet x) &\stackrel{\text{def}}{=} (\lambda x.\pi \bullet (f\ (\pi^{-1} \bullet x)))\ (\pi \bullet x) \\
&= \pi \bullet (f\ (\pi^{-1} \bullet (\pi \bullet x))) \\
&= \pi \bullet (f\ x)
\end{aligned}
$$

…s

…ts

$\pi \bullet (x :: xs) = (\pi \bullet x) :: (\pi \bullet xs)$

- $\pi \bullet X \stackrel{\text{def}}{=} \{\pi \bullet x \mid x \in X\}$ — sets

  $\pi \bullet [\lambda x.N]_\alpha = [\lambda(\pi \bullet x).(\pi \bullet N)]_\alpha$

- $\pi \bullet f \stackrel{\text{def}}{=} \lambda x.\pi \bullet (f\ (\pi^{-1} \bullet x))$ — functions

  $\pi \bullet (f\ x) = (\pi \bullet f)\ (\pi \bullet x)$

- $\pi \bullet x \stackrel{\text{def}}{=} x$ — integers, strings, bools

Perm's for Other Types

- $\pi \bullet (x_1, x_2) \stackrel{\text{def}}{=} (\pi \bullet x_1, \pi \bullet x_2)$ pairs
- $\pi \bullet [] \stackrel{\text{def}}{=} []$ lists

  $\pi \bullet (x :: xs) \stackrel{\text{def}}{=} (\pi \bullet x) :: (\pi \bullet xs)$
- $\pi \bullet X \stackrel{\text{def}}{=} \{\pi \bullet x \mid x \in X\}$ sets

  $\pi \bullet [\lambda x.N]_\alpha = [\lambda(\pi \bullet x).(\pi \bullet N)]_\alpha$
- $\pi \bullet f \stackrel{\text{def}}{=} \lambda x.\pi \bullet (f\ (\pi^{-1} \bullet x))$ functions

  $\pi \bullet (f\ x) = (\pi \bullet f)\ (\pi \bullet x)$
- $\pi \bullet x \stackrel{\text{def}}{=} x$ integers, strings, bools

Support and Freshness

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

$$\mathsf{supp}(x) \stackrel{\mathsf{def}}{=} \{a \mid \mathsf{infinite}\,\{b \mid (a\,b)\bullet x \neq x\}\}$$

$$a \# x \stackrel{\mathsf{def}}{=} a \not\in \mathsf{supp}(x)$$

In words: all atoms a where the set

$$\{b \mid (a\,b)\bullet x \neq x\}$$

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:

$$\mathsf{supp}(x) \stackrel{\text{def}}{=} \{a \mid \mathsf{infinite}\,\{b \mid (a\,b)\bullet x \neq x\}\}$$

$$a \# x \stackrel{\text{def}}{=} a \notin \mathsf{supp}(x)$$

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

In wor

$$\{b \mid (a\,b)\bullet x \neq x\}$$

is infinite (each swapping $(a\,b)$ needs to change something in x).

Support of an Atom

What is the support of the atom c?

$$\mathsf{supp}(c) \stackrel{\text{def}}{=} \{a \mid \mathsf{infinite}\,\{b \mid (a\,b)\bullet c \neq c\}\}$$

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

Support of an Atom

What is the support of the atom c?

$$\mathrm{supp}(c) \stackrel{\text{def}}{=} \{a \mid \text{infinite}\,\{b \mid (a\,b)\bullet c \neq c\}\}$$

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

a: $(a\,?)\bullet c \neq c$

Support of an Atom

What is the support of the atom c?

$$\mathsf{supp}(c) \stackrel{\mathsf{def}}{=} \{a \mid \mathsf{infinite}\,\{b \mid (a\,b)\bullet c \neq c\}\}$$

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

a: $(a\,?)\bullet c \neq c$ no

b: $(b\,?)\bullet c \neq c$

Support of an Atom

What is the support of the atom c?

$$\mathrm{supp}(c) \stackrel{\mathrm{def}}{=} \{a \mid \text{infinite}\,\{b \mid (a\,b)\bullet c \neq c\}\}$$

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

a: $(a\,?)\bullet c \neq c$ no
b: $(b\,?)\bullet c \neq c$ no
c: $(c\,?)\bullet c \neq c$

Support of an Atom

What is the support of the atom c?

$$\mathsf{supp}(c) \stackrel{\text{def}}{=} \{a \mid \mathsf{infinite}\,\{b \mid (a\,b)\bullet c \neq c\}\}$$

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

a:	$(a\,?)\bullet c \neq c$	no
b:	$(b\,?)\bullet c \neq c$	no
c:	$(c\,?)\bullet c \neq c$	yes
d:	$(d\,?)\bullet c \neq c$	

Support of an Atom

What is the support of the atom c?

$$\mathsf{supp}(c) \stackrel{\mathsf{def}}{=} \{a \mid \mathsf{infinite}\,\{b \mid (a\,b)\bullet c \neq c\}\}$$

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

a:	$(a\,?)\bullet c \neq c$	no
b:	$(b\,?)\bullet c \neq c$	no
c:	$(c\,?)\bullet c \neq c$	yes
d:	$(d\,?)\bullet c \neq c$	no
	$\vdots$	no

Support of an Atom

What is the support of the atom c?

$$\mathrm{supp}(c) \stackrel{\text{def}}{=} \{a \mid \text{infinite}\,\{b \mid (a\,b)\bullet c \neq c\}\}$$

Let's check th[…]y) atoms one by one:

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

a:	$(a\,?)\bullet c \neq c$	no
b:	$(b\,?)\bullet c \neq c$	no
c:	$(c\,?)\bullet c \neq c$	yes
d:	$(d\,?)\bullet c \neq c$	no
	$\vdots$	no

Support of a Pair

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

Support of a Pair

$$\mathsf{supp}(t_1,t_2) \stackrel{\mathsf{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\,b)\bullet(t_1,t_2) \neq (t_1,t_2)\}\}$$

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

Support of a Pair

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

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

We know

$$(t_1, t_2) = (s_1, s_2) \ \text{ iff } \ t_1 = s_1 \wedge t_2 = s_2$$

hence

$$(t_1, t_2) \neq (s_1, s_2) \ \text{ iff } \ t_1 \neq s_1 \vee t_2 \neq s_2$$

Support of a Pair

$$\mathsf{supp}(t_1,t_2) \stackrel{\text{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\,b)\bullet(t_1,t_2) \neq (t_1,t_2)\}\}$$

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

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1 \vee (a\,b)\bullet t_2 \neq t_2\}\}$$

Support of a Pair

$$\mathrm{supp}(t_1,t_2) \stackrel{\text{def}}{=} \{a \mid \mathrm{inf}\ \{b \mid (a\,b)\bullet(t_1,t_2) \neq (t_1,t_2)\}\}$$

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

$$\{a \mid \mathrm{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1 \vee (a\,b)\bullet t_2 \neq t_2\}\}$$

$$\{a \mid \mathrm{inf}(\{b \mid (a\,b)\bullet t_1 \neq t_1\} \cup \{b \mid (a\,b)\bullet t_2 \neq t_2\})\}$$

Support of a Pair

$$\mathsf{supp}(t_1,t_2) \stackrel{\text{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\,b)\bullet(t_1,t_2) \neq (t_1,t_2)\}\}$$

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

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1 \vee (a\,b)\bullet t_2 \neq t_2\}\}$$

$$\{a \mid \mathsf{inf}(\{b \mid (a\,b)\bullet t_1 \neq t_1\} \cup \{b \mid (a\,b)\bullet t_2 \neq t_2\})\}$$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1\} \vee \mathsf{inf}\{b \mid (a\,b)\bullet t_2 \neq t_2\}\}$$

Support of a Pair

$$\mathsf{supp}(t_1,t_2) \stackrel{\mathsf{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\,b)\bullet(t_1,t_2) \neq (t_1,t_2)\}\}$$

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

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1 \vee (a\,b)\bullet t_2 \neq t_2\}\}$$

$$\{a \mid \mathsf{inf}(\{b \mid (a\,b)\bullet t_1 \neq t_1\} \cup \{b \mid (a\,b)\bullet t_2 \neq t_2\})\}$$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1\} \vee \mathsf{inf}\{b \mid (a\,b)\bullet t_2 \neq t_2\}\}$$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1\}\} \cup \{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_2 \neq t_2\}\}$$

Support of a Pair

$$\mathsf{supp}(t_1,t_2) \stackrel{\mathsf{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\,b)\bullet(t_1,t_2) \neq (t_1,t_2)\}\}$$

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

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1 \vee (a\,b)\bullet t_2 \neq t_2\}\}$$

$$\{a \mid \mathsf{inf}(\{b \mid (a\,b)\bullet t_1 \neq t_1\} \cup \{b \mid (a\,b)\bullet t_2 \neq t_2\})\}$$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1\} \vee \mathsf{inf}\{b \mid (a\,b)\bullet t_2 \neq t_2\}\}$$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1\}\} \cup \{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_2 \neq t_2\}\}$$

$$\mathsf{supp}(t_1) \qquad \cup \qquad \mathsf{supp}(t_2)$$

Support of a Pair

$$\mathsf{supp}(t_1,t_2) \stackrel{\text{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\,b)\bullet(t_1,t_2) \neq (t_1,t_2)\}\}$$

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

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1 \vee (a\,b)\bullet t_2 \neq t_2\}\}$$

$\{a \mid$

$\{a \mid$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_1 \neq t_1\}\} \cup \{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet t_2 \neq t_2\}\}$$

$$\mathsf{supp}(t_1) \qquad \cup \qquad \mathsf{supp}(t_2)$$

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

However, such things are proved for you:
the user does **not** have to bother with them.

```
lemma
  shows "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::atom set)"
proof -
 have "supp (t₁,t₂) = {a. inf {b. [(a,b)]•(t₁,t₂) ≠ (t₁,t₂)}}"
   by (simp add: supp_def)
 also have "... = {a. inf {b. ([(a,b)]•t₁,[(a,b)]•t₂) ≠ (t₁,t₂)}}" by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁ ∨ [(a,b)]•t₂ ≠ t₂}}" by simp
 also have "... = {a. inf ({b. [(a,b)]•t₁ ≠ t₁} ∪ {b. [(a,b)]•t₂ ≠ t₂})}"
   by (simp only: Collect_disj_eq)
 also have "... = {a. (inf {b. [(a,b)]•t₁ ≠ t₁})∨(inf {b. [(a,b)]•t₂ ≠ t₂})}"
   by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁}}∪{a. inf {b. [(a,b)]•t₂ ≠ t₂}}"
   by (simp only: Collect_disj_eq)
 also have "... = supp t₁ ∪ supp t₂" by (simp add: supp_def)
 finally show "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::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) ≠ (t₁,t₂)}}"
   by (simp add: supp_def)
 also have "... = {a. inf {b. ([(a,b)]•t₁,[(a,b)]•t₂) ≠ (t₁,t₂)}}" by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁ ∨ [(a,b)]•t₂ ≠ t₂}}" by simp
 also have "... = {a. inf ({b. [(a,b)]•t₁ ≠ t₁} ∪ {b. [(a,b)]•t₂ ≠ t₂})}"
   by (simp only: Collect_disj_eq)
 also have "... = {a. (inf {b. [(a,b)]•t₁ ≠ t₁})∨(inf {b. [(a,b)]•t₂ ≠ t₂})}"
   by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁}}∪{a. inf {b. [(a,b)]•t₂ ≠ t₂}}"
   by (simp only: Collect_disj_eq)
 also have "... = supp t₁ ∪ supp t₂" by (simp add: supp_def)
 finally show "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::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) ≠ (t₁,t₂)}}"
   by (simp add: supp_def)
 also have "... = {a. inf {b. ([(a,b)]•t₁,[(a,b)]•t₂) ≠ (t₁,t₂)}}" by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁ ∨ [(a,b)]•t₂ ≠ t₂}}" by simp
 also have "... = {a. inf ({b. [(a,b)]•t₁ ≠ t₁} ∪ {b. [(a,b)]•t₂ ≠ t₂})}"
   by (simp only: Collect_disj_eq)
 also have "... = {a. (inf {b. [(a,b)]•t₁ ≠ t₁})∨(inf {b. [(a,b)]•t₂ ≠ t₂})}"
   by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁}}∪{a. inf {b. [(a,b)]•t₂ ≠ t₂}}"
   by (simp only: Collect_disj_eq)
 also have "... = supp t₁ ∪ supp t₂" by (simp add: supp_def)
 finally show "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::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) ≠ (t₁,t₂)}}"
   by (simp add: supp_def)
 also have "... = {a. inf {b. ([(a,b)]•t₁,[(a,b)]•t₂) ≠ (t₁,t₂)}}" by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁ ∨ [(a,b)]•t₂ ≠ t₂}}" by simp
 also have "... = {a. inf ({b. [(a,b)]•t₁ ≠ t₁} ∪ {b. [(a,b)]•t₂ ≠ t₂})}"
   by (simp only: Collect_disj_eq)
 also have "... = {a. (inf {b. [(a,b)]•t₁ ≠ t₁})∨(inf {b. [(a,b)]•t₂ ≠ t₂})}"
   by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁}}∪{a. inf {b. [(a,b)]•t₂ ≠ t₂}}"
   by (simp only: Collect_disj_eq)
 also have "... = supp t₁ ∪ supp t₂" by (simp add: supp_def)
 finally show "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::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) ≠ (t₁,t₂)}}"
   by (simp add: supp_def)
 also have "... = {a. inf {b. ([(a,b)]•t₁,[(a,b)]•t₂) ≠ (t₁,t₂)}}" by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁ ∨ [(a,b)]•t₂ ≠ t₂}}" by simp
 also have "... = {a. inf ({b. [(a,b)]•t₁ ≠ t₁} ∪ {b. [(a,b)]•t₂ ≠ t₂})}"
   by (simp only: Collect_disj_eq)
 also have "... = {a. (inf {b. [(a,b)]•t₁ ≠ t₁})∨(inf {b. [(a,b)]•t₂ ≠ t₂})}"
   by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁}}∪{a. inf {b. [(a,b)]•t₂ ≠ t₂}}"
   by (simp only: Collect_disj_eq)
 also have "... = supp t₁ ∪ supp t₂" by (simp add: supp_def)
 finally show "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::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) ≠ (t₁,t₂)}}"
   by (simp add: supp_def)
 also have "... = {a. inf {b. ([(a,b)]•t₁,[(a,b)]•t₂) ≠ (t₁,t₂)}}" by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁ ∨ [(a,b)]•t₂ ≠ t₂}}" by simp
 also have "... = {a. inf ({b. [(a,b)]•t₁ ≠ t₁} ∪ {b. [(a,b)]•t₂ ≠ t₂})}"
   by (simp only: Collect_disj_eq)
 also have "... = {a. (inf {b. [(a,b)]•t₁ ≠ t₁})∨(inf {b. [(a,b)]•t₂ ≠ t₂})}"
   by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁}}∪{a. inf {b. [(a,b)]•t₂ ≠ t₂}}"
   by (simp only: Collect_disj_eq)
 also have "... = supp t₁ ∪ supp t₂" by (simp add: supp_def)
 finally show "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::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) ≠ (t₁,t₂)}}"
   by (simp add: supp_def)
 also have "... = {a. inf {b. ([(a,b)]•t₁,[(a,b)]•t₂) ≠ (t₁,t₂)}}" by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁ ∨ [(a,b)]•t₂ ≠ t₂}}" by simp
 also have "... = {a. inf ({b. [(a,b)]•t₁ ≠ t₁} ∪ {b. [(a,b)]•t₂ ≠ t₂})}"
   by (simp only: Collect_disj_eq)
 also have "... = {a. (inf {b. [(a,b)]•t₁ ≠ t₁})∨(inf {b. [(a,b)]•t₂ ≠ t₂})}"
   by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁}}∪{a. inf {b. [(a,b)]•t₂ ≠ t₂}}"
   by (simp only: Collect_disj_eq)
 also have "... = supp t₁ ∪ supp t₂" by (simp add: supp_def)
 finally show "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::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) ≠ (t₁,t₂)}}"
   by (simp add: supp_def)
 also have "... = {a. inf {b. ([(a,b)]•t₁,[(a,b)]•t₂) ≠ (t₁,t₂)}}" by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁ ∨ [(a,b)]•t₂ ≠ t₂}}" by simp
 also have "... = {a. inf ({b. [(a,b)]•t₁ ≠ t₁} ∪ {b. [(a,b)]•t₂ ≠ t₂})}"
   by (simp only: Collect_disj_eq)
 also have "... = {a. (inf {b. [(a,b)]•t₁ ≠ t₁})∨(inf {b. [(a,b)]•t₂ ≠ t₂})}"
   by simp
 also have "... = {a. inf {b. [(a,b)]•t₁ ≠ t₁}}∪{a. inf {b. [(a,b)]•t₂ ≠ t₂}}"
   by (simp only: Collect_disj_eq)
 also have "... = supp t₁ ∪ supp t₂" by (simp add: supp_def)
 finally show "supp (t₁,t₂) = supp t₁ ∪ ((supp t₂)::atom set)" by simp
qed
```

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a = b$ clear.

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

(1) $\mathsf{fin}\{c \mid (a\,c)\bullet x \neq x\}$ from Ass. +Def. of $\#$

$\mathsf{fin}\{c \mid (b\,c)\bullet x \neq x\}$

$$a \# x \stackrel{\text{def}}{=} a \notin \mathsf{supp}(x)$$

$$\mathsf{supp}(x) \stackrel{\text{def}}{=} \{a \mid \mathsf{inf}\{c \mid (a\,c)\bullet x \neq x\}\}$$

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

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

$\text{fin}\{c \mid (b\,c)\bullet x \neq x\}$

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

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

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

(2) $\text{fin}\{c \mid (a\,c)\bullet x \neq x \vee (b\,c)\bullet x \neq x\}$ f'rm (1)

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b){\bullet}x = x$

Proof: case $a \neq b$:

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

(2) $\text{fin}\{c \mid (a\,c){\bullet}x \neq x \vee (b\,c){\bullet}x \neq x\}$ f'rm (1)

(3) $\text{inf}\{c \mid \neg((a\,c){\bullet}x \neq x \vee (b\,c){\bullet}x \neq x)\}$ f'rm (2)

Given a finite set of atoms, its 'co-set' must be infinite.

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

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

$\quad \text{fin}\{c \mid (b\,c)\bullet x \neq x\}$

(2) $\text{fin}\{c \mid (a\,c)\bullet x \neq x \vee (b\,c)\bullet x \neq x\}$ f'rm (1)

(3) $\text{inf}\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x\}$ f'rm (2)

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b){\bullet}x = x$

Proof: case $a \neq b$:

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

$\text{fin}\{c \mid (b\,c){\bullet}x \neq x\}$

(2) $\text{fin}\{c \mid (a\,c){\bullet}x \neq x \vee (b\,c){\bullet}x \neq x\}$ f'rm (1)

(3) $\text{inf}\{c \mid (a\,c){\bullet}x = x \wedge (b\,c){\bullet}x = x\}$ f'rm (2)

(4) (i) $(a\,c){\bullet}x = x$ (ii) $(b\,c){\bullet}x = x$ for a $c \in$ (3)

If a set is infinite, it must contain a few elements. Let's pick c.

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b){\bullet}x = x$

Proof: case $a \neq b$:

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

(2) $\text{fin}\{c \mid (a\,c){\bullet}x \neq x \vee (b\,c){\bullet}x \neq x\}$ f'rm (1)

(3) $\text{inf}\{c \mid (a\,c){\bullet}x = x \wedge (b\,c){\bullet}x = x\}$ f'rm (2)

(4) (i) $(a\,c){\bullet}x = x$ (ii) $(b\,c){\bullet}x = x$ for a $c \in$ (3)

(5) $(a\,c){\bullet}x = x$ by (4i)

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

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

$\text{fin}\{c \mid (b\,c)\bullet x \neq x\}$

(2) $\text{fin}\{c \mid (a\,c)\bullet x \neq x \vee (b\,c)\bullet x \neq x\}$ f'rm (1)

(3) $\text{inf}\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x\}$ f'rm (2)

(4) (i) $(a\,c)\bullet x = x$ (ii) $(b\,c)\bullet x = x$ for a $c \in$ (3)

(5) $(a\,c)\bullet x = x$ by (4i)

(6) $(b\,c)\bullet(a\,c)\bullet x = (b\,c)\bullet x$ by bij.

bij.: $x = y$ iff $\pi\bullet x = \pi\bullet y$

It's as Simple as This

Lemma: $a \mathrel{\#} x \wedge b \mathrel{\#} x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

(1) fin$\{c \mid (a\,c)\bullet x \neq x\}$ — from Ass. +Def. of $\#$

fin$\{c \mid (b\,c)\bullet x \neq x\}$

(2) fin$\{c \mid (a\,c)\bullet x \neq x \vee (b\,c)\bullet x \neq x\}$ — f'rm (1)

(3) inf$\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x\}$ — f'rm (2)

(4) (i) $(a\,c)\bullet x = x$ (ii) $(b\,c)\bullet x = x$ — for a $c \in$ (3)

(5) $(a\,c)\bullet x = x$ — by (4i)

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

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b) \bullet x = x$

Proof: case $a \neq b$:

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

(2) $\text{fin}\{c \mid (a\,c) \bullet x \neq x \vee (b\,c) \bullet x \neq x\}$ f'rm (1)

(3) $\text{inf}\{c \mid (a\,c) \bullet x = x \wedge (b\,c) \bullet x = x\}$ f'rm (2)

(4) (i) $(a\,c) \bullet x = x$ (ii) $(b\,c) \bullet x = x$ for a $c \in (3)$

(5) $(a\,c) \bullet x = x$ by (4i)

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

(7) $(a\,c) \bullet (b\,c) \bullet (a\,c) \bullet x = (a\,c) \bullet x$ by bij.

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\, b) \bullet x = x$

Proof: case $a \neq b$:

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

(2) $\text{fin}\{c \mid (a\, c) \bullet x \neq x \vee (b\, c) \bullet x \neq x\}$ f'rm (1)

(3) $\text{inf}\{c \mid (a\, c) \bullet x = x \wedge (b\, c) \bullet x = x\}$ f'rm (2)

(4) (i) $(a\, c) \bullet x = x$ (ii) $(b\, c) \bullet x = x$ for a $c \in$ (3)

(5) $(a\, c) \bullet x = x$ by (4i)

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

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

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

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

(2) $\text{fin}\{c \mid (a\,c)\bullet x \neq x \vee (b\,c)\bullet x \neq x\}$ f'rm (1)

(3) $\text{inf}\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x\}$ f'rm (2)

(4) (i) $(a\,c)\bullet x = x$ (ii) $(b\,c)\bullet x = x$ for a $c \in$ (3)

(5) $(a\,c)\bullet x = x$ by (4i)

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

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

$$(a\,c)(b\,c)(a\,c)\bullet a = b$$
$$(a\,c)(b\,c)(a\,c)\bullet b = a$$
$$(a\,c)(b\,c)(a\,c)\bullet c = c$$

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

(1) $\mathsf{fin}\{c \mid (a\,c)\bullet x \neq x\}$ — from Ass. +Def. of $\#$

$\mathsf{fin}\{c \mid (b\,c)\bullet x \neq x\}$

(2) fin … $\neq x\}$ — f'rm (1)

(3) inf … $= x\}$ — f'rm (2)

> property of permutation:
>
> $\pi_1 \sim \pi_2 \Rightarrow \pi_1\bullet x = \pi_2\bullet x$

(4) (i) $(a\,c)\bullet x = x$ (ii) $(b\,c)\bullet x = x$ — for a $c \in$ (3)

(5) $(a\,c)\bullet x = x$ — by (4i)

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

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

(8) $(a\,b)\bullet x = x$ — by prop. of perms

It's as Simple as This

Lemma: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

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

(2) $\text{fin}\{c \mid (a\,c)\bullet x \neq x \vee (b\,c)\bullet x \neq x\}$ f'rm (1)

(3) $\text{inf}\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x\}$ f'rm (2)

(4) (i) $(a\,c)\bullet x = x$ (ii) $(b\,c)\bullet x = x$ for a $c \in$ (3)

(5) $(a\,c)\bullet x = x$ by (4i)

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

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

(8) $(a\,b)\bullet x = x$ 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:

$$\exists a.\ \ a \# x$$

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 α-equivalence classes and because of a trick.

Nominal Abstractions

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

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

and has to satisfy two properties:

- $\pi\bullet([a].x) = [\pi\bullet a].(\pi\bullet x)$
- $[a].x \;=\; [b].y$ iff

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

- These two properties imply for finitely supported x

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

Nominal Abstractions

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

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

and has to satisfy two properties:

- $\pi \bullet ([a].x) = [\pi \bullet a].(\pi \bullet x)$
- $[a].x = [b].y$ iff

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

- These two properties imply for finitely supported x

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

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:

- $\pi \bullet ([a].x) = [\pi \bullet a].(\pi \bullet x)$
- $[a].x$

  $(a =$

  $$\frac{}{a \,\#\, [a].x} \qquad \frac{b \neq a \quad b \,\#\, x}{b \,\#\, [a].x}$$

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

- These two properties imply for finitely supported x

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

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: atom $\to$ trm 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 $[a].t$ '=' $[\lambda a.t]_\alpha$

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

$(\lambda b.$if $a = b$
then Some(a,c)
else if $b \# (a,c)$
then Some$((b\,a)\bullet(a,c))$ else None$)$

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

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

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

$(\lambda b.\text{if } a = b$

$\quad\text{then Some}(a, c)$

$\quad\text{else if } b \# (a, c)$

$\quad\quad\text{then Some}((b\, a)\bullet(a, c)) \text{ else None})$

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

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

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

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

$(\lambda b.$if $a = b$

then Some(a,c)

else if $b \;\#\; (a,c)$

then Some$((b\;a)\bullet(a,c))$ 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 $[a].t$ '=' $[\lambda a.t]_\alpha$

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

$(\lambda b.\text{if } a = b$
  $\text{then Some}(a,c)$
  $\text{else if } b \# (a,c)$
    $\text{then Some}((b\,a)\bullet(a,c)) \text{ else None})$

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' None

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

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

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)
c 'applied to' $[a].(a,c)$ 'gives' None
d 'applied to' $[a].(a,c)$ 'gives' Some(d,c)
⋮

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

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

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

a 'applied to' $[a].(a,c)$ 'gives' Some(a,c)		'$\lambda a.(a\,c)$'
b 'applied to' $[a].(a,c)$ 'gives' Some(b,c)		'$\lambda b.(b\,c)$'
c 'applied to' $[a].(a,c)$ 'gives' None		
d 'applied to' $[a].(a,c)$ 'gives' Some(d,c)		'$\lambda d.(d\,c)$'
⋮		⋮

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

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

$(\lambda b.$if $a = b$
then Some(a,c)
else if $b \# (a,c)$
then Some$((b\,a)\bullet(a,c))$ 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)

c ‘applied to’ $[a].(a,c)$ ‘gives’ None

d ‘applied to’ $[a].(a,c)$ ‘gives’ Some(d,c)

$\vdots$

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

‘$\lambda a.(a\,c)$’

‘$\lambda b.(b\,c)$’

‘$\lambda d.(d\,c)$’

$\vdots$

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

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

This function 'takes' a lambda-abstraction and an atom, and tries to rename the abstraction according to the given atom.

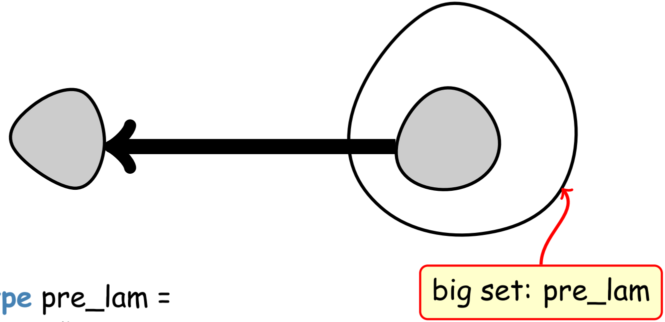
Function $[a].t$ '=' $[\lambda a.t]_\alpha$

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

This function 'takes' a lambda-abstraction and an atom, and tries to rename the abstraction according to the given atom.

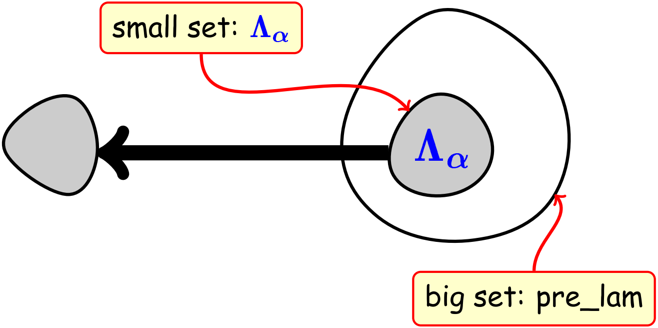
α-Equivalence Classes

We can now define **inductively** **named** α-equivalence classes of lambda-terms:




```
datatype pre_lam =
  Var "atom"
| App "pre_lam" "pre_lam"
| Lam "atom ⇒ pre_lam option"
```

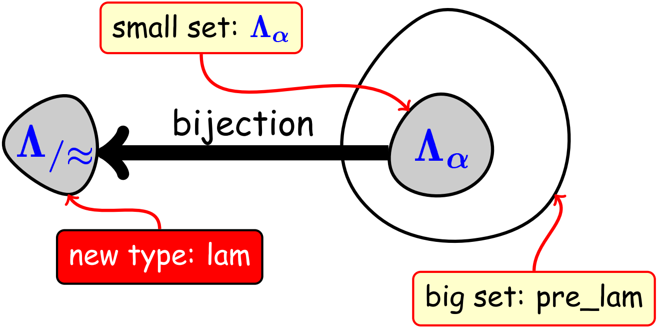
Definition of Small Set



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

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

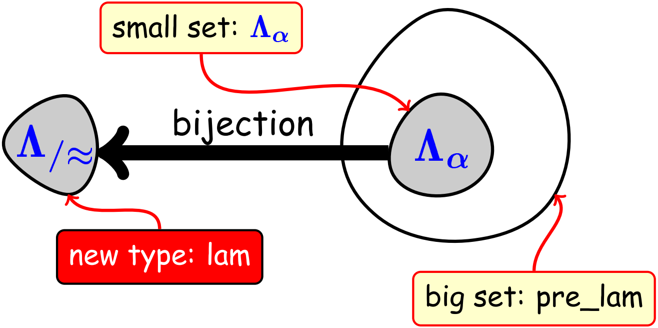
Definition of Small Set



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

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

Definition of Small Set



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

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

This means we have the familiar induction principle for Λ_α and so also for $\Lambda_{/\approx}$.

Structural Induction

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

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

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

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

Better Structural Induction

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

implies (as seen yesterday)

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

provided c is finitely supported

“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)
```

"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)
```

"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 alpha[no_vars]

$([a].x = [b].y) =$
$\quad (a = b \wedge x = y \vee a \neq b \wedge x = [(a, b)] \bullet y \wedge a \# y)$

“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 swap_simps[no_vars]
[(a, b)] • a = b
[(a, b)] • b = a
```

"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)
```

In LF

```
nominal_datatype
    kind =   Type
           | KPi "ty" "«name»kind"
and ty =   TConst "id"
           | TApp "ty" "trm"
           | TPi "ty" "«name»ty"
and trm = Const "id"
           | Var "name"
           | App "trm" "trm"
           | Lam "ty" "«name»trm"

abbreviation KPi_syn :: "name ⇒ ty ⇒ kind ⇒ kind" ("Π[_:_]._")
where "Π[x:A].K ≡ KPi A x K"

abbreviation TPi_syn :: "name ⇒ ty ⇒ ty ⇒ ty" ("Π[_:_]._")
where "Π[x:A1].A2 ≡ TPi A1 x A2"

abbreviation Lam_syn :: "name ⇒ ty ⇒ trm ⇒ trm" ("Lam [_:_]._")
where  "Lam [x:A].M ≡ Lam A x M"
```

In My PhD

```
nominal_datatype trm =
    Ax    "name" "coname"
  | Cut   "«coname»trm" "«name»trm"                      ("Cut ⟨_⟩._ (_)._")
  | NotR  "«name»trm" "coname"                           ("NotR (_)._ _")
  | NotL  "«coname»trm" "name"                           ("NotL ⟨_⟩._ _")
  | AndR  "«coname»trm" "«coname»trm" "coname"           ("AndR ⟨_⟩._ ⟨_⟩._ _")
  | AndL1 "«name»trm" "name"                             ("AndL1 (_)._ _")
  | AndL2 "«name»trm" "name"                             ("AndL2 (_)._ _")
  | OrR1  "«coname»trm" "coname"                         ("OrR1 ⟨_⟩._ _")
  | OrR2  "«coname»trm" "coname"                         ("OrR2 ⟨_⟩._ _")
  | OrL   "«name»trm" "«name»trm" "name"                 ("OrL (_)._ (_)._ _")
  | ImpR  "«name»(«coname»trm)" "coname"                 ("ImpR (_).⟨_⟩._ _")
  | ImpL  "«coname»trm" "«name»trm" "name"               ("ImpL ⟨_⟩._ (_)._ _")
```

- A SN-result for cut-elimination in CL: reviewed by Henk Barendregt and Andy Pitts, and reviewers of conference and journal paper. Still, I found errors in central lemmas; fortunately the main claim was correct :o)

Conclusions

- The support of $\lambda x.x$:

$$
\begin{aligned}
\pi \bullet \lambda x.x &\stackrel{\text{def}}{=} \lambda x.\pi \bullet ((\lambda x.x)\ (\pi^{-1} \bullet x)) \\
&= \lambda x.\pi \bullet \pi^{-1} \bullet x \\
&= \lambda x.x
\end{aligned}
$$

Conclusions

- The support of $\lambda x.x$:

$$\begin{aligned} \pi \bullet \lambda x.x &\stackrel{\text{def}}{=} \lambda x.\pi \bullet ((\lambda x.x)\,(\pi^{-1} \bullet x)) \\ &= \lambda x.\pi \bullet \pi^{-1} \bullet x \\ &= \lambda x.x \end{aligned}$$

- Therefore

$$\begin{aligned} \mathsf{supp}(\lambda x.x) &\stackrel{\text{def}}{=} \{a \,|\, \mathsf{infinite}\{b \,|\, (a\,b) \bullet \lambda x.x \neq \lambda x.x\}\} \\ &= \{a \,|\, \mathsf{infinite}\{b \,|\, \lambda x.x \neq \lambda x.x\}\} \\ &= \varnothing \end{aligned}$$

Conclusions

- To represent α-equivalence classes we used a trick:
  - The same α-equivalence class can be written in many ways ($\lambda x.x$, $\lambda y.y$).
  - Similarly, one and the same function can be written in many ways ($[x]$.Var x, $[y]$.Var y).

Conclusions

- To represent α-equivalence classes we used a trick:
  - The same α-equivalence class can be written in many ways ($\lambda x.x$, $\lambda y.y$).
  - Similarly, one and the same function can be written in many ways ($[x]$.Var x, $[y]$.Var y).

- **Next:** This all might look complicated, but my claim is that nearly all complications can be hidden away. I will show you tomorrow how to formalise a simple CK machine.

Exercises

- 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?