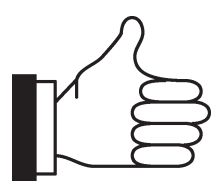
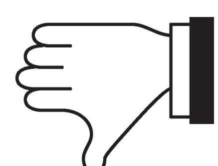
Nominal Techniques in Isabelle/HOL

based on work by Andy Pitts

joint work with Stefan, Markus, Alexander...



up



down

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

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

Proof: By induction on the structure of M.

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

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

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

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

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

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

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

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

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

Proof: By induction on the structure of M.

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

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

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

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

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

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

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

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

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

Proof: By induction on the structure of M.

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

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

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

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

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

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

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

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

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

Proof: By induction on the structure of M.

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

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

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

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

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume

2.1.12. Convention: Terms that are α-congruent are identified. So now we write $\lambda x.x \equiv \lambda y.y$ etcetera.

2.1.13. Variable Convention: If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.

2.1.14. Moral: Using conventions 2.1.12 and 2.1.13 one can work with λ-terms in the naive way.

tion hypothesis.

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

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

Proof: By induction on the structure of M.

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

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

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

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

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

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

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

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

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

Proof: By induction on the structure of M.

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

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

  Case 1.2. $M \equiv$

  implie

  Case 1.3. $M \equiv$

- **Case 2:** $M \equiv$ … e

  that $z \not\equiv x, y$ … s

  $(\lambda z.M_1)$

  $\equiv \lambda z.(M_1[$

  $\equiv \lambda z.(M_1[$

  $\equiv (\lambda z.M_1)$

- **Case 3:** $M \equiv$ … -

  tion hypothesis. … "

Remember: only if $y \neq x$ and $x \notin FV(N)$ then

$$(\lambda y.M)[x := N] = \lambda y.(M[x := N])$$

$$\begin{aligned}
& (\lambda z.M_1)[x := N][y := L] & \\
\equiv\ & (\lambda z.(M_1[x := N]))[y := L] & \overset{1}{\leftarrow} \\
\equiv\ & \lambda z.(M_1[x := N][y := L]) & \overset{2}{\leftarrow} \\
\equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) & \text{IH} \\
\equiv\ & (\lambda z.(M_1[y := L]))[x := N[y := L]]) & \overset{2}{\rightarrow}\ ! \\
\equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]]. & \overset{1}{\rightarrow}
\end{aligned}$$

Existing Formalisation Techniques

- with “bare hands”

  (extremely messy) defining lambda-terms as syntax-trees; work with explicit α-conversions

- de-Bruijn indices

  they are “very formal”; but even if there were no technical problems with dB, they involve often quite different lemmas than “paper proofs”

- HOAS

  ...yes, **but** induction is problematic, no way to define conveniently notions such as simultaneous substitution **etc** ...not my personal preference ;o)

Formal Proof in Isabelle

lemma forget:
assumes a: "$x \# L$"
shows "$L[x ::= N] = L$"
using a by (nominal_induct L avoiding: x N rule: lam.induct)
(auto simp add: abs_fresh fresh_atm)

lemma fresh_fact:
fixes x :: "name"
assumes a: "$x \# M$" and b: "$x \# N$"
shows "$x \# M[y ::= N]$"
using a b by (nominal_induct M avoiding: x y N rule: lam.induct)
(auto simp add: abs_fresh fresh_atm)

lemma subst_lemma:
assumes a: "$x \neq y$" and b: "$x \# L$"
shows "$M[x ::= N][y ::= L] = M[y ::= L][x ::= N[y ::= L]]$"
using a b by (nominal_induct M avoiding: x y N L rule: lam.induct)
(auto simp add: forget fresh_fact)

We Start with Atoms

We introduce **atoms**. Everything that is **bound**, **binding** and **bindable** is an atom (independent from the language at hand).

a countable infinite set — this will be important on later on.

We Start with Atoms

We introduce **atoms**. Everything that is **bound**, **binding** and **bindable** is an atom (independent from the language at hand).

We Start with Atoms

We introduce **atoms**. Everything that is **bound**, **binding** and **bindable** is an atom (independent from the language at hand).

example lambda-calculus

$$\lambda a.\lambda b.(a\, b\, c)$$

a and b are atoms—bound and binding

We Start with Atoms

We introduce **atoms**. Everything that is **bound**, **binding** and **bindable** is an atom (independent from the language at hand).

example lambda-calculus

$$\lambda a.\lambda b.(a\, b\, c)$$

c is an atom—bindable

We Start with Atoms

We introduce **atoms**. Everything that is **bound**, **binding** and **bindable** is an atom (independent from the language at hand).

example lambda-calculus

$$\lambda c.\lambda a.\lambda b.(a\,b\,c)$$

now c is bound

We Start with Atoms

We introduce **atoms**. Everything that is **bound**, **binding** and **bindable** is an atom (independent from the language at hand).

example integrals

$$\int_0^1 x^2 + y\, dx$$

x is an atom—bound and binding

We Start with Atoms

We introduce **atoms**. Everything that is **bound**, **binding** and **bindable** is an atom (independent from the language at hand).

example integrals

$$\int_{-\infty}^{\infty} \left(\int_{0}^{1} x^2 + y \, dx \right) dy$$

y is an atom—bindable

We Start with Atoms

We introduce **atoms**. Everything that is **bound**, **binding** and **bindable** is an atom (independent from the language at hand).

example integrals

$$\int_0^1 x^2 + y\, dx$$

0, 1 and 2 are constants

We Start with Atoms

We introduce **atoms**. Everything that is **bound**, **binding** and **bindable** is an atom (independent from the language at hand).

example integrals

$$\int_{-\infty}^{\infty}\left(\int_{0}^{1} x^{2} + y\,dx\right)d2$$

binding 2 does not make sense

We Start with Atoms

We introduce **atoms**. Everything that is **bound**,
bindin
Why atoms? Because an operation we introduce shortly will act on atoms only and leaves everything else alone.
ndent
from t

examp

$$\int_{-\infty}^{\infty}\left(\int_0^1 x^2 + y\, dx\right) d2$$

binding 2 does not make sense

Swappings

In general, renaming substitutions do not respect α-equivalence, e.g.

$$\lambda a.b \qquad\qquad \lambda c.b$$

Swappings

In general, renaming substitutions do not respect α-equivalence, e.g.

$$\begin{aligned}[b := a]\,\lambda a.b &= \lambda a.a\end{aligned} \qquad \begin{aligned}[b := a]\,\lambda c.b &= \lambda c.a\end{aligned}$$

Swappings

In general, renaming substitutions do not respect α-equivalence, e.g.

$$[b := a]\,\boldsymbol{\lambda a.b} = \boldsymbol{\lambda a.a} \qquad [b := a]\,\boldsymbol{\lambda c.b} = \boldsymbol{\lambda c.a}$$

<u>Traditional Solution:</u> replace $[b := a]t$ by a more complicated, 'capture-avoiding' form of substitution.

Swappings

In general, renaming substitutions do not respect α-equivalence, e.g.

$$(b\,a)\bullet \lambda a.b = \lambda b.a \qquad\qquad (b\,a)\bullet \lambda c.b = \lambda c.a$$

<u>Nice Alternative:</u> use a less complicated operation for renaming

$(b\,a)\bullet t \stackrel{\text{def}}{=}$ swap **all** occurrences of b and a in t

Swappings

In general, renaming substitutions do not respect α-equivalence, e.g.

$$(b\ a)\bullet \lambda a.b = \lambda b.a \qquad\qquad (b\ a)\bullet \lambda c.b = \lambda c.a$$

Nice Alternative: use a less complicated operation for renaming

$(b\ a)\bullet t \stackrel{\text{def}}{=}$ swap **all** occurrences of b and a in t

be they bound, binding or bindable

Swappings

In general, renaming substitutions do not respect α-equivalence, e.g.

$$(b\,a)\bullet\lambda a.b = \lambda b.a \qquad\qquad (b\,a)\bullet\lambda c.b = \lambda c.a$$

<u>Nice Alternative:</u> use a less complicated operation for renaming

$(b\,a)\bullet t \stackrel{\text{def}}{=}$ swap **all** occurrences of b and a in t

Unlike for $[b:=a](-)$, for $(b\,a)\bullet(-)$ we do have if $t =_\alpha t'$ then $(b\,a)\bullet t =_\alpha (b\,a)\bullet t'$.

Permutations

We shall extend 'swappings' to '(finite) lists of swappings'

$$(a_1\,b_1)\ldots(a_n\,b_n),$$

also called **permutations** (we shall often write π for them). Permutations are **bijective** mappings from atoms to atoms. For example

$$\pi = \begin{pmatrix} a \mapsto b \\ b \mapsto a \\ c \mapsto c \end{pmatrix} = (c\,b)(a\,b)(a\,c)$$

Permutations

We shall extend 'swappings' to '(finite) lists of swappings'

$$(a_1\, b_1) \ldots (a_n\, b_n),$$

also called **permutations** (we shall often write π for them). Permutations are **bijective** mappings from atoms to atoms. For example

$$\pi = \begin{pmatrix} a \mapsto b \\ b \mapsto a \\ c \mapsto c \end{pmatrix} \qquad (c\, b)(a\, b)(a\, c) \bullet a = b$$

Permutations

We shall extend 'swappings' to '(finite) lists of swappings'

$$(a_1\, b_1) \ldots (a_n\, b_n),$$

also called **permutations** (we shall often write π for them). Permutations are **bijective** mappings from atoms to atoms. For example

$$\pi = \begin{pmatrix} a \mapsto b \\ b \mapsto a \\ c \mapsto c \end{pmatrix} \qquad (c\, b)(a\, b)(a\, c) \bullet b = a$$

Permutations

We shall extend 'swappings' to '(finite) lists of swappings'

$$(a_1\,b_1)\ldots(a_n\,b_n),$$

also called **permutations** (we shall often write π for them). Permutations are **bijective** mappings from atoms to atoms. For example

$$\pi = \begin{pmatrix} a \mapsto b \\ b \mapsto a \\ c \mapsto c \end{pmatrix} \qquad (c\,b)(a\,b)(a\,c)\bullet c = c$$

Permutations

We shall extend 'swappings' to '(finite) lists of
swappin

also cal … en write
π for t … **ve**
mapping … xample

Our list-representation is **not** unique, because

$$(c\,b)(a\,b)(a\,c) \text{ and } (a\,b)$$

are the 'same' permutation.

$$\pi = \begin{pmatrix} a \mapsto b \\ b \mapsto a \\ c \mapsto c \end{pmatrix} \qquad (c\,b)(a\,b)(a\,c) \bullet c = c$$

Permutations on Atoms

A permutation **acts** on an atom 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 $(a_1\, a_2)$

Permutations on Atoms (ct.)

- 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 on λ-Terms

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

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

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

We have:

- $\pi^{-1} \bullet (\pi \bullet t) = t$
- $t_1 = t_2$ if and only if $\pi \bullet t_1 = \pi \bullet t_2$
- $\pi \bullet t_1 = t_2$ if and only if $t_1 = \pi^{-1} \bullet t_2$

Permutations on λ-Terms

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

We have:

- $\pi^{-1}\bullet(\pi\bullet$
- $t_1 = t_2$ i $\bullet t_2$
- $\pi\bullet t_1 = t_2$ if and only if $t_1 = \pi^{-1}\bullet t_2$

'we treat lambdas as if there were no binders'

Permutations on λ-Terms

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

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

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

We have:

- $\pi^{-1} \bullet (\pi \bullet t) = t$
- $t_1 = t_2$ if and only if $\pi \bullet t_1 = \pi \bullet t_2$
- $\pi \bullet t_1 = t_2$ if and only if $t_1 = \pi^{-1} \bullet t_2$

Permutations on λ-Terms

What is it about permutations? Well...

- they have much nicer properties than renaming-substitutions (stemming from the fact that they are bijections on atoms),
- they give rise to a relatively simple definition of α-equivalence on syntax-trees (shown next)
- and more later on

α-Equivalence

Consider the following four rules:

$$\frac{}{a \approx a}\approx\text{-atm}$$

$$\frac{t_1 \approx s_1 \quad t_2 \approx s_2}{t_1\, t_2 \approx s_1\, s_2}\approx\text{-app}$$

$$\frac{t \approx s}{\lambda a.t \approx \lambda a.s}\approx\text{-lam}_1$$

$$\frac{t \approx (a\, b)\bullet s \quad a \not\in \mathsf{fv}(s)}{\lambda a.t \approx \lambda b.s}\approx\text{-lam}_2$$

$\boxed{\text{assuming } a \neq b}$

α-Equivalence

Consider the following four rules:

$$\frac{}{a \approx a}\approx\text{-atm} \qquad \frac{t_1 \approx s_1 \quad t_2 \approx s_2}{t_1\, t_2 \approx s_1\, s_2}\approx\text{-app}$$

$$\frac{t \approx s}{\lambda a.t \approx \lambda a.s}\approx\text{-lam}_1 \qquad \frac{t \approx (a\,b)\bullet s \quad a \notin \mathsf{fv}(s)}{\lambda a.t \approx \lambda b.s}\approx\text{-lam}_2$$

$\boxed{\text{assuming } a \neq b}$

$\lambda a.t \approx \lambda b.s$ iff t is α-equivalent with s in which all occurrences of b have been renamed to a...oops **permuted** to a.

α-Equivalence

But this alone leads to an 'unsound' rule! Consider

$$\lambda a.b \quad \text{and} \quad \lambda b.a$$

which are **not** α-equivalent. However, if we apply the permutation $(a\,b)$ to a we get

$$b \approx b$$

which leads to non-sense.

We need to ensure that there are **no** 'free' occurrences of a in s, i.e. $a \notin \mathrm{fv}(s)$.

to a. . . oops **permuted** to a.

α-Equivalence

Consider the following four rules:

$$\frac{}{a \approx a}\ \approx\text{-atm} \qquad \frac{t_1 \approx s_1 \quad t_2 \approx s_2}{t_1\, t_2 \approx s_1\, s_2}\ \approx\text{-app}$$

$$\frac{t \approx s}{\lambda a.t \approx \lambda a.s}\ \approx\text{-lam}_1 \qquad \frac{t \approx (a\, b)\bullet s \quad a \notin \mathsf{fv}(s)}{\lambda a.t \approx \lambda b.s}\ \approx\text{-lam}_2$$

assuming $a \neq b$

$\lambda a.t \approx \lambda b.s$ iff t is α-equivalent with s in which all occurrences of b have been renamed to a. . . oops **permuted** to a.

Not-Free-In

$$\frac{}{a \notin \mathsf{fv}(b)}\;\mathsf{fv\text{-}atm} \qquad \frac{a \notin \mathsf{fv}(t_1) \quad a \notin \mathsf{fv}(t_2)}{a \notin \mathsf{fv}(t_1\,t_2)}\;\mathsf{fv\text{-}app}$$

$$\frac{}{a \notin \mathsf{fv}(\lambda a.t)}\;\mathsf{fv\text{-}lam}_1 \qquad \frac{a \notin \mathsf{fv}(t)}{a \notin \mathsf{fv}(\lambda b.t)}\;\mathsf{fv\text{-}lam}_2$$

$\boxed{\text{assuming } a \neq b}$

Be careful, we have defined two relations over lambda-terms/syntax-trees. We have **not** defined what 'bound' or 'free' means. That is a feature, not a bug.™

$\approx$ is an Equivalence

You might be an agnostic and notice that

$$\frac{a \neq b \quad t \approx (a\, b) \bullet s \quad a \notin \mathsf{fv}(s)}{\lambda a.t \approx \lambda b.s} \approx\text{-lam}_2$$

is defined rather asymmetrically. Still we have:

Theorem: $\approx$ is an equivalence relation.

(Reflexivity)	$t \approx t$
(Symmetry)	if $t_1 \approx t_2$ then $t_2 \approx t_1$
(Transitivity)	if $t_1 \approx t_2$ and $t_2 \approx t_3$ then $t_1 \approx t_3$

$\Rightarrow$ is rather tricky to prove

Comparison with $=_\alpha$

Traditionally $=_\alpha$ is defined as

least congruence which identifies $\lambda a.t$ with $\lambda b.[a := b]t$ provided b is not free in t

where $[a := b]t$ replaces all free occurrences of a by b in t.

- with $(-) \approx (-)$ and $(-) \not\in \mathrm{fv}(-)$ we never need to choose a 'fresh' atom (good for implementations)
- permutation respects both relations, whilst renaming-substitution does not

General Permutations

So far we have only considered permutations acting on atoms and lambda-terms. We are now going to overload $\_\bullet\_ : \alpha\ prm \Rightarrow \iota \Rightarrow \iota$ to act on other types as well.

- $\pi \bullet a$ $\quad a$ being an atom (of type α)

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

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

General Permutations

So far we have only considered permutations acting on atoms and lambda-terms. We are now going to overload $\_\bullet\_ : \alpha\ prm \Rightarrow \iota \Rightarrow \iota$ to act on other types as well.

For sake of simplicity, let us assume we only have **one** type of atoms.

- π ...)

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

General Permutations

So far we have only considered permutations acting on atoms and lambda-terms. We are now going to overload $\_\bullet\_ : \alpha\ prm \Rightarrow \iota \Rightarrow \iota$ to act on other types as well.

- $\pi\bullet a \qquad a$ being an atom (of type α)

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

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

Overloading of _ • _

- $\pi \bullet [] \stackrel{\text{def}}{=} []$ 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
- $\pi \bullet (x_1, x_2) \stackrel{\text{def}}{=} (\pi \bullet x_1, \pi \bullet x_2)$ products
- $\pi \bullet \mathsf{None} \stackrel{\text{def}}{=} \mathsf{None}$ options

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

Permutation Properties

Whenever we deal with a type, we have to make sure that it has a sensible permutation operation... axiomatic type-classes are just(?) the thing we need:

- $[] \bullet x = x$
- $(\pi_1 @ \pi_2) \bullet x = \pi_1 \bullet (\pi_2 \bullet x)$
- $\pi_1 \sim \pi_2$ implies $\pi_1 \bullet x = \pi_2 \bullet x$

We refer to these properties as $pt_{\alpha,\iota}$ and refer to the type ι as **permutation type** (provided they are satisfied for ι).

Permutation Types

The property of being a permutation type is in some sense hereditary:

- $pt_{\alpha,\alpha}$
- $pt_{\alpha,\iota\ list}$ provided $pt_{\alpha,\iota}$

  similar for sets, products and options
- $pt_{\alpha,nat}$, $pt_{\alpha,string}$, $pt_{\alpha,bool}$

The nominal datatype-package needs to make sure that every type the implementors deem important is a permutation type (with axiomatic type-classes no problem).

Permutations on Functions

Interesting: Given $f : \iota_1 \Rightarrow \iota_2$ and

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

then pt_{α,ι_1} and pt_{α,ι_2} imply $pt_{\alpha,\iota_1 \Rightarrow \iota_2}$.

The definition on functions implies that

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

holds for permutation types.

Support and Freshness

Even more interesting: The **support** of an object $x : \iota$ is a set of atoms α:

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

An atom is **fresh** for an x, if it is not in the support of x:

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

I will often drop the α in supp_α.

Support and Freshness

Even more interesting: The **support** of an object $x : \iota$ is a set of atoms α:

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

An a… …he
supp…

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

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

I will often drop the α in supp_α.

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:

Support of an Atom

What is the support of the atom c?

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

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

$$a\text{:} \quad (a\,?)\bullet c \neq c$$

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?

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

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

Support of an Atom

What is the support of the atom c?

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

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?

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

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

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

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 Product

$$\mathsf{supp}(x_1,x_2) \stackrel{\mathsf{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\, b)\bullet(x_1,x_2) \neq (x_1,x_2)\}\}$$

Support of a Product

$$\mathsf{supp}(x_1,x_2) \stackrel{\mathsf{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\,b)\bullet(x_1,x_2) \neq (x_1,x_2)\}\}$$

$$\{a \mid \mathsf{inf}\{b \mid ((a\,b)\bullet x_1, (a\,b)\bullet x_2) \neq (x_1, x_2)\}\}$$

Support of a Product

$$\mathsf{supp}(x_1,x_2) \stackrel{\mathsf{def}}{=} \{a \,|\, \mathsf{inf}\ \{b \,|\, (a\,b)\bullet(x_1,x_2) \neq (x_1,x_2)\}\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, ((a\,b)\bullet x_1, (a\,b)\bullet x_2) \neq (x_1, x_2)\}\}$$

We know

$$(x_1, x_2) = (y_1, y_2) \;\text{ iff }\; x_1 = y_1 \wedge x_2 = y_2$$

hence

$$(x_1, x_2) \neq (y_1, y_2) \;\text{ iff }\; x_1 \neq y_1 \vee x_2 \neq y_2$$

Support of a Product

$$\mathsf{supp}(x_1,x_2) \stackrel{\mathsf{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\,b)\bullet(x_1,x_2) \neq (x_1,x_2)\}\}$$

$$\{a \mid \mathsf{inf}\{b \mid ((a\,b)\bullet x_1, (a\,b)\bullet x_2) \neq (x_1, x_2)\}\}$$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet x_1 \neq x_1 \vee (a\,b)\bullet x_2 \neq x_2\}\}$$

Support of a Product

$$\mathsf{supp}(x_1,x_2) \stackrel{\mathsf{def}}{=} \{a \,|\, \mathsf{inf}\ \{b \,|\, (a\,b)\bullet(x_1,x_2) \neq (x_1,x_2)\}\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, ((a\,b)\bullet x_1, (a\,b)\bullet x_2) \neq (x_1, x_2)\}\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_1 \neq x_1 \vee (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\{a \,|\, \mathsf{inf}(\{b \,|\, (a\,b)\bullet x_1 \neq x_1\} \cup \{b \,|\, (a\,b)\bullet x_2 \neq x_2\})\}$$

Support of a Product

$$\mathsf{supp}(x_1,x_2) \stackrel{\mathsf{def}}{=} \{a \,|\, \mathsf{inf}\ \{b \,|\, (a\,b)\bullet(x_1,x_2) \neq (x_1,x_2)\}\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, ((a\,b)\bullet x_1, (a\,b)\bullet x_2) \neq (x_1, x_2)\}\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_1 \neq x_1 \vee (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\{a \,|\, \mathsf{inf}(\{b \,|\, (a\,b)\bullet x_1 \neq x_1\} \cup \{b \,|\, (a\,b)\bullet x_2 \neq x_2\})\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_1 \neq x_1\} \vee \mathsf{inf}\{b \,|\, (a\,b)\bullet x_2 \neq x_2\}\}$$

Support of a Product

$$\mathsf{supp}(x_1,x_2) \stackrel{\mathsf{def}}{=} \{a \,|\, \mathsf{inf}\ \{b \,|\, (a\,b)\bullet(x_1,x_2) \neq (x_1,x_2)\}\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, ((a\,b)\bullet x_1, (a\,b)\bullet x_2) \neq (x_1, x_2)\}\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_1 \neq x_1 \vee (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\{a \,|\, \mathsf{inf}(\{b \,|\, (a\,b)\bullet x_1 \neq x_1\} \cup \{b \,|\, (a\,b)\bullet x_2 \neq x_2\})\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_1 \neq x_1\} \vee \mathsf{inf}\{b \,|\, (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_1 \neq x_1\}\} \cup \{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_2 \neq x_2\}\}$$

Support of a Product

$$\mathsf{supp}(x_1,x_2) \stackrel{\text{def}}{=} \{a \mid \mathsf{inf}\ \{b \mid (a\,b)\bullet(x_1,x_2) \neq (x_1,x_2)\}\}$$

$$\{a \mid \mathsf{inf}\{b \mid ((a\,b)\bullet x_1, (a\,b)\bullet x_2) \neq (x_1, x_2)\}\}$$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet x_1 \neq x_1 \vee (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\{a \mid \mathsf{inf}(\{b \mid (a\,b)\bullet x_1 \neq x_1\} \cup \{b \mid (a\,b)\bullet x_2 \neq x_2\})\}$$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet x_1 \neq x_1\} \vee \mathsf{inf}\{b \mid (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet x_1 \neq x_1\}\} \cup \{a \mid \mathsf{inf}\{b \mid (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\mathsf{supp}(x_1) \qquad \cup \qquad \mathsf{supp}(x_2)$$

Support of a Product

$$\mathsf{supp}(x_1,x_2) \stackrel{\mathsf{def}}{=} \{a \,|\, \mathsf{inf}\ \{b \,|\, (a\,b)\bullet(x_1,x_2) \neq (x_1,x_2)\}\}$$

$\{a \,|\, \mathsf{inf}\{b$

So $\mathsf{supp}(x_1, x_2) = \mathsf{supp}(x_1) \cup \mathsf{supp}(x_2)$

$$\{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_1 \neq x_1 \vee (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\{a \,|\, \mathsf{inf}(\{b \,|\, (a\,b)\bullet x_1 \neq x_1\} \cup \{b \,|\, (a\,b)\bullet x_2 \neq x_2\})\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_1 \neq x_1\} \vee \mathsf{inf}\{b \,|\, (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_1 \neq x_1\}\} \cup \{a \,|\, \mathsf{inf}\{b \,|\, (a\,b)\bullet x_2 \neq x_2\}\}$$

$$\mathsf{supp}(x_1) \qquad\qquad \cup \qquad\qquad \mathsf{supp}(x_2)$$

Some Simple Properties

- $\text{supp}\ (x_1, x_2) = (\text{supp}\ x_1) \cup (\text{supp}\ x_2)$

  $a \# (x_1, x_2)$ iff $a \# x_1 \wedge a \# x_2$
- $\text{supp}_\alpha\ (a : \alpha) = \{a\}$
- $\text{supp}\ [] = \varnothing$,
  $\text{supp}(x :: xs) = \text{supp}(x) \cup \text{supp}(xs)$
- $\text{supp}(\text{None}) = \varnothing$,
  $\text{supp}(\text{Some}(x)) = \text{supp}(x)$
- $\text{supp}(1) = \text{supp}(''s'') = \text{supp}(\mathit{True}) = \varnothing$

Some Simple Properties

- su
  $_2$
- su
- su
  su
- su
  sup
- supp(1) = supp$(\text{''}s\text{''})$ = supp$(\mathit{True}) = \varnothing$

The support of "finitary" structures is usually quite simple: for example the support of a lambda-term t is the set of atoms occuring in t.

$$
\begin{array}{lcl}
\pi \bullet a & \stackrel{\text{def}}{=} & \ldots \\
\pi \bullet (t_1\, t_2) & \stackrel{\text{def}}{=} & (\pi \bullet t_1)\, (\pi \bullet t_2) \\
\pi \bullet \lambda a.t & \stackrel{\text{def}}{=} & \lambda(\pi \bullet a)\, (\pi \bullet t)
\end{array}
$$

FYI: Infinitary Structures

- supp $\quad = \varnothing$ set of all atoms in α

  since $\forall a, b.\ (a\ b)\bullet \quad =$

- supp $F = \{a_1, \ldots, a_n\}$ assuming F is a finite set of atoms $a_1, \ldots, a_n$
- not every set of atoms has finite support: e.g. "$atoms/2$"
- the support of functions is even more interesting (one instance later on)

Existence of a Fresh Atom

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

A: For any finitely supported x:

$$\boxed{\exists c.\ \ c \# x}$$

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

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b) \bullet x = x$

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b) \bullet x = x$

Proof: case $a = b$ clear

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

Proof: case $a \neq b$:

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

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

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

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

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

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

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

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

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

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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′) $\mathsf{fin}\{c \mid (a\,c)\bullet x \neq x \vee (b\,c)\bullet x \neq x\}$ f. (1)

(3′) $\mathsf{inf}\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x)\}$ f. (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 so that $c \neq a, b$.

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

<u>Proof:</u> 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. (1)

(3') inf$\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x)\}$ f. (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 is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

(3') inf$\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x)\}$ f. (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 is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

(3') inf$\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x)\}$ f. (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 is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

(3′) inf$\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x)\}$ f. (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 is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b) \bullet x = x$

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. (1)

(3′) inf$\{c \mid (a\,c) \bullet x = x \wedge (b\,c) \bullet x = x)\}$ f. (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 is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

(3′) inf$\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x)\}$ f. (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)

$$(a\,c)(b\,c)(a\,c) \sim (a\,b)$$

It is as Simple as That...

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

(3′) inf$\{c \mid (a$ ... f. (2′)

(4) (i) $(a\,c)\bullet$ ... $c \in$ (3′)

3rd property of $pt_{\alpha,\iota}$:

$\pi_1 \sim \pi_2 \Rightarrow \pi_1\bullet x = \pi_2\bullet x$

(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 3rd. prop.

It is as Simple as That…

Assuming $pt_{\alpha,\iota}$: $a \# x \wedge b \# x \Rightarrow (a\,b)\bullet x = x$

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. (1)

(3') inf$\{c \mid (a\,c)\bullet x = x \wedge (b\,c)\bullet x = x)\}$ f. (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 3rd. prop.

Done.

Last Lem. in the SN-Proof

lemma all_Red:
assumes a: "$\Gamma \vdash t : \tau$"
    and b: "$\forall(x,\sigma) \in \mathsf{set}\,\Gamma.\ x \in \mathsf{dom}(\theta) \wedge \theta\langle x\rangle \in Red_{\sigma}$"
shows "$\theta[t] \in Red_{\tau}$"

Girard in Proofs-and-Types:

Let t be any term (not assumed to be reducible), and suppose all free variables of t are among $x_1 \ldots x_n$ of types $\sigma_1 \ldots \sigma_n$. If $t_1 \ldots t_n$ are reducible terms of type $\sigma_1 \ldots \sigma_n$ then $t[x_1 := t_1, \ldots, x_n := t_n]$ is reducible.

Last Lem. in the SN-Proof

lemma all_Red:
assumes a: "$\Gamma \vdash t : \tau$"
and b: "$\forall (x,\sigma) \in$ set $\Gamma.\ x \in$ dom$(\theta) \wedge \theta\langle x\rangle \in Red_\sigma$"
shows "$\theta[t] \in Red_\tau$"
using a b proof (nominal_induct t avoiding: $\Gamma\ \tau\ \theta$ rule: lam.induct)
case (Lam a t)
have ih: "$\bigwedge \Gamma\tau\theta.\ [\Gamma \vdash t : \tau; \forall (x,\sigma) \in$ set $\Gamma.\ x \in$ dom$(\theta) \wedge \theta\langle x\rangle \in Red_\sigma]$
$\Longrightarrow \theta[t] \in Red_\tau$"
and θ_cond: "$\forall (x,\sigma) \in$ set $\Gamma.\ x \in$ dom$(\theta) \wedge \theta\langle x\rangle \in Red_\sigma$"
and fresh: "$a \# \Gamma$" "$a \# \theta$"
and "$\Gamma \vdash$ Lam $[a].t : \tau$" by fact
hence "$\exists \tau_1\tau_2.\ \tau = \tau_1 \to \tau_2 \wedge ((a,\tau_1)\#\Gamma) \vdash t : \tau_2$" by (simp ...)
then obtain τ_1 τ_2 where τ: "$\tau = \tau_1 \to \tau_2$"
and ty: "$((a,\tau_1)\#\Gamma) \vdash t : \tau_2$" by blast
from ih have "$\forall s \in Red_{\tau_1}.\ (\theta[t])[a ::= s] \in Red_{\tau_2}$" using fresh ty θ_cond
by (force dest: fresh_context simp add: psubst_subst)
hence "Lam $[a].(\theta[t]) \in Red_{\tau_1 \to \tau_2}$" by (simp only: abs_Red)
thus "θ[Lam $[a].t] \in Red_\tau$" using fresh τ by simp
qed

Last Lem. in the SN-Proof

lemma all_Red:
assumes a: "$\Gamma \vdash t : \tau$"
    and b: "$\forall (x,\sigma) \in \mathsf{set}\,\Gamma.\ x \in \mathsf{dom}(\theta) \wedge \theta\langle x\rangle \in Red_\sigma$"
shows "$\theta[t] \in Red_\tau$"
using a b proof (nominal_induct t avoiding: $\Gamma\ \tau\ \theta$ rule: lam.induct)
  case (Lam a t)
  have ih: "$\bigwedge \Gamma\tau\theta.\ [\Gamma \vdash t : \tau; \forall (x,\sigma) \in \mathsf{set}\,\Gamma.\ x \in \mathsf{dom}(\theta) \wedge \theta\langle x\rangle \in Red_\sigma]$
          $\Longrightarrow \theta[t] \in Red_\tau$"
  and θ_cond: "$\forall (x,\sigma) \in \mathsf{set}\,\Gamma.\ x \in \mathsf{dom}(\theta) \wedge \theta\langle x\rangle \in Red_\sigma$"
  and fresh: "$a \# \Gamma$" "$a \# \theta$"
  and "$\Gamma \vdash$ Lam
  hence "$\exists \tau_1 \tau_2.$ … by (simp ...)
  then obtain τ_1
  … blast
  from ih have "$\forall s \in Red_{\tau_1}.\ (\theta[t])[a ::= s] \in Red_{\tau_2}$" using fresh ty
θ_cond
    by (force dest: fresh_context simp add: psubst_subst)
  hence "Lam $[a].(\theta[t]) \in Red_{\tau_1 \to \tau_2}$" by (simp only: abs_Red)
  thus "$\theta[$Lam $[a].t] \in Red_\tau$" using fresh τ by simp
qed

The End?