

Nominal Techniques as an Isabelle/HOL-Theory

Christian Urban

University of Munich (LMU)

jointwork with Christine Tasson, ENS Cachan

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(a[a := t_2])$

subcase 2: $a \notin FV(b[a := t_2])$

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$ by assumption
subcase 2: $a \notin FV(b[a := t_2])$

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$

by assumption

subcase 2: $a \notin FV(b)$

ok

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$

by assumption

subcase 2: $a \notin FV(b)$

ok

case applications:

$a \notin FV(s_1 s_2 [a := t_2])$

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$

by assumption

subcase 2: $a \notin FV(b)$

ok

case applications:

$a \notin FV(s_1[a := t_2] s_2[a := t_2])$

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 .

case variables:

subcase 1: $a \notin FV(t_2)$

by assumption

subcase 2: $a \notin FV(b)$

ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$ by assumption
subcase 2: $a \notin FV(b)$ ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$ by IH

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$ by assumption
subcase 2: $a \notin FV(b)$ ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$ by IH

case abstractions (c sufficiently fresh):

$a \notin FV(\lambda c.s [a := t_2])$

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$ by assumption
subcase 2: $a \notin FV(b)$ ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$ by IH

case abstractions (c sufficiently fresh):

$a \notin FV(\lambda c.(s[a := t_2]))$

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$ by assumption
subcase 2: $a \notin FV(b)$ ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$ by IH

case abstractions (c sufficiently fresh):

$a \notin FV(s[a := t_2]) - \{c\}$

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$ by assumption
subcase 2: $a \notin FV(b)$ ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$ by IH

case abstractions (c sufficiently fresh):

$a \notin FV(s[a := t_2]) - \{c\}$ by IH

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By induction on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$ by assumption
subcase 2: $a \notin FV(b)$ ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$ by IH

case abstractions (c sufficiently fresh):

$a \notin FV(s[a := t_2]) - \{c\}$ by IH

Done.

On Paper

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

Proof: By **induction** on the structure of t_1 :

case variables:

subcase 1: $a \notin FV(t_2)$ by assumption
subcase 2: $a \notin FV(b)$ ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$ by IH

case abstractions (**c sufficiently fresh**):

$a \notin FV(s[a := t_2]) - \{c\}$ by IH

Done.

Existing Formalisation Techniques

- with "bare hands"

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

Existing Formalisation Techniques

■ with "bare hands"

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

■ de-Bruijn indices

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

Existing Formalisation Techniques

with "bare hands"

For example, the weakening lemma in the simply-typed lambda calculus says:

for all Γ, t, τ :

if $\Gamma \vdash t : \tau$ then

for all $x \notin \Gamma, \sigma: \{x:\sigma\} \cup \Gamma \vdash t : \tau$

How do you want to formulate this with de Bruijn indices?

Existing Formalisation Techniques

■ with "bare hands"

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

■ de-Bruijn indices

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

■ HOAS

... yes, but induction is problematic, no way to define conveniently notions such as simultaneous substitution

And Now in Isabelle

lemma assumes "a # t₂" shows "a # t₁ [a ::= t₂]"

I will write:

$a \# t$ for $a \notin FV(t)$
 $\text{Am } a, \text{Pr } s_1 s_2, [a].t$ for var's, app's and lam's

I will also start from some lemmas, for example:

$$\text{Am } a [a ::= t] = t$$

$$\text{Am } b [a ::= t] = \text{Am } b \quad \text{if } a \neq b$$

$$\text{Pr } s_1 s_2 [a ::= t] = \text{Pr}(s_1 [a ::= t])(s_2 [a ::= t])$$

$$[b].s [a ::= t] = [b].(s [a ::= t]) \quad \text{if } b \# (a, t)$$

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1[a ::= t2]"
proof(induct rule:
  ind[of "λt1 (a, t2). a # t1[a ::= t2] (a, t2)"])
  case 1 show ?case
  by (cases "b=a", simp_all add: subs_simps)
next
  case 2 show ?case by (simp add: subs_simps)
next
  case 3
  have "∃c. c # (a, t2)" by (rule exists_fresh)
  thus ?case by (force intro: fresh_absI1
    simp add: fresh_prod subs_simps)
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"
```

```
proof (induct rule:
```

```
  ind [of "λt1 (a, t2). a # t1 [a ::= t2] (a, t2)"])
```

```
  case 1 show ?case
```

```
  by (cases "b=a", simp_all add: subs_simps)
```

```
next
```

```
  case prems:
```

```
next  a # t2
```

```
  case goal (lemma, 1 subgoal):
```

```
  have 1. a # t1 [a ::= t2]
```

```
  thus . case by (force intro: fresh_prod)
```

```
    simp add: fresh_prod subs_simps)
```

```
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"  
proof (induct rule:   
      ind [of "λt1 (a, t2). a # t1 [a ::= t2] (a, t2)"])  
  case 1 show ?case  
  by (cases "b=a", simp_all add: subs_simps)  
next  
  prems:  
  case a # t2  
next  
  goal (lemma, 3 subgoals):  
  case  
  hav  
  thu  
  1. a # Am b [a ::= t2]  
  2.  $\bigwedge t_1 t_{2a}. a \# t_1 [a ::= t_2] \wedge a \# t_{2a} [a ::= t_2]$   
      $\implies a \# \text{Pr } t_1 t_{2a} [a ::= t_2]$   
  3.  $\exists c. c \# (a, t_2) \wedge$   
      $(\forall t. a \# t [a ::= t_2] \longrightarrow a \# [c]. t [a ::= t_2])$   
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"
proof (induct rule:
  ind [of "λt1 (a, t2). a # t1 [a ::= t2] (a, t2)"])
  case 1 show ?case
  by (cases "b=a", simp_all add: subs_simps)
next
  case 2 show ?case by (simp add: subs_simps)
next
  prems:
  case a # t2
  have goal (show, 1 subgoal):
  thus 1. a # Am b [a ::= t2]
    simp add: fresh_prod subs_simps)
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1[a ::= t2]"
proof(induct rule:
      ind[of "λt1 (a, t2). a # t1[a ::= t2] (a, t2)"])
  case 1 show ?case
  by (cases "b=a", simp_all add: subs_simps)
next
  case 2 show ?case by (simp add: subs_simps)
next
  case 3
  have "∃c. c # (a, t2)" by (rule exists_fresh)
  thus ?case by (force intro: fresh_absI1
      simp add: fresh_prod subs_simps)
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"  
proof (induct rule:   
      ind [of "λt1 (a, t2). a # t1 [a ::= t2] (a, t2)"])  
  case 1 show ?case  
  by (cases "b=a", simp_all add: subs_simps)  
next  
  case 2 show ?case by (simp add: subs_simps)  
next  
  case prems:  
  have a # t2  
  thus a # t1 [a ::= t2] ∧ a # t2a [a ::= t2]  
  goal (lemma, 1 subgoal):  
  1. a # Pr t1 t2a [a ::= t2]  
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1[a ::= t2]"  
proof(induct rule:   
      ind[of "λt1 (a, t2). a # t1[a ::= t2] (a, t2)"])  
  case 1 show ?case  
  by (cases "b=a", simp_all add: subs_simps)  
next  
  case 2 show ?case by (simp add: subs_simps)  
next  
  case 3  
  have "∃c. c # (a, t2)" by (rule exists_fresh)  
  thus ?case by (force intro: fresh_absI1  
                simp add: fresh_prod subs_simps)  
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"
proof (induct rule:
prems:
a # t2
case 1
by goal (lemma, 1 subgoal):
1.  $\exists c. c \# (a, t_2) \wedge$ 
 $(\forall t. a \# t [a ::= t_2] \longrightarrow a \# [c]. t [a ::= t_2])$ 
case 2 show ?case by (simp add: subs_simps)
next
case 3
have " $\exists c. c \# (a, t_2)$ " by (rule exists_fresh)
thus ?case by (force intro: fresh_absI1
simp add: fresh_prod subs_simps)
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"
```

```
proof (induct rule:
```

```
  prems:
```

```
  a # t2
```

```
  case
```

```
  by
```

```
  goal (have, 1 subgoal):
```

```
next
```

```
  1.  $\exists c. c \# (a, t_2)$ 
```

```
  case 2 show ?case by (simp add: subs_simps)
```

```
next
```

```
  case 3
```

```
  have " $\exists c. c \# (a, t_2)$ " by (rule exists_fresh)
```

```
  thus ?case by (force intro: fresh_absI1
```

```
    simp add: fresh_prod subs_simps)
```

```
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"  
proof (induct rule:   
      ind [of "λt1 (a, t2). a # t1 [a ::= t2] (a, t2)"]  
    case 1 show ?case  
      by (cases "b=a", simp_all add: subs_simps)  
next  
    case 2 show ?case by (simp add: subs_simps)  
next  
    case 3  
      have "∃c. c # (a, t2)" by (rule exists_fresh)  
      thus ?case by (force intro: fresh_absI1  
        simp add: fresh_prod subs_simps)  
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"
```

```
proof (induct rule:
```

```
  prems:
```

```
  a # t2
```

```
  case
```

```
   $\exists c. c \# (a, t_2)$ 
```

```
  by
```

```
next
```

```
  goal (show, 1 subgoal):
```

```
  case
```

```
  1.  $\exists c. c \# (a, t_2) \wedge$ 
```

```
     $(\forall t. a \# t [a ::= t_2] \longrightarrow a \# [c]. t [a ::= t_2])$ 
```

```
next
```

```
  case 3
```

```
  have " $\exists c. c \# (a, t_2)$ " by (rule exists_fresh)
```

```
  thus ?case by (force intro: fresh_absI1
```

```
    simp add: fresh_prod subs_simps)
```

```
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"
proof (induct rule:
  ind [of "λt1 (a, t2). a # t1 [a ::= t2] (a, t2)"])
  case 1 show ?case
  by (cases "b=a", simp_all add: subs_simps)
next
  case 2 show ?case by (simp add: subs_simps)
next
  case 3
  have "∃c. c # (a, t2)" by (rule exists_fresh)
  thus ?case by (force intro: fresh_absI1
    simp add: fresh_prod subs_simps)
qed
```

And Now in Isabelle

```
lemma assumes "a # t2" shows "a # t1 [a ::= t2]"  
proof (induct rule:   
      ind [of "λt1 (a, t2). a # t1 [a ::= t2] (a, t2)"]  
    case 1 show ?case  
      by (cases "b=a", simp_all add: subs_simps)  
next  
    case 2 show ?case by (simp add: subs_simps)  
next  
    case 3  
      have "∃c. c # (a, t2)" by (rule exists_fresh)  
      thus ?case by (force intro: fresh_absI1  
                    simp add: fresh_prod subs_simps)  
qed
```

And Now in Isabelle

```
lemma assumes "a#t2" shows "a#t1[a::=t2]"
```

```
proof(induct rule:
```

```
ind[of "\t1 (a#t2) a#t1[a::=t2] (a#t2)"])
```

```
case
```

```
by
```

```
next
```

```
case
```

```
next
```

```
case
```

```
hav
```

```
thus ?case by (force intro: fresh_absI1
```

```
simp add: fresh_prod subs_simps)
```

```
qed
```

Well, I cheated you by one line (the induction requires us to prove four subgoals), but apart from that, this **is** the proof in Isabelle. And you have to admit, one can hardly get any "closer" to the original "proof".

The aim of this talk is to show you what is going on behind this proof.

What Is Behind?

- a trick
- and plenty of ideas from Pitts' nominal work*
- (only) two concepts are a bit complicated

*In its original formulation Nominal Logic is incompatible with the axiom of choice; FM-sets aren't the best starting point for formal verifications. Therefore some adjustments are needed for theorem provers.

First a Detour: Definition of α -Equivalence

Here is a maybe-unfamiliar definition of α -equivalence, but it does define α -equivalence:

$$\frac{}{a \approx a} \text{ var}$$

$$\frac{t_1 \approx s_1 \quad t_2 \approx s_2}{(t_1 t_2) \approx (s_1 s_2)} \text{ app}$$

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

$$\frac{t \approx (a b) \bullet s \quad a \notin FV(s)}{\lambda a. t \approx \lambda b. s} \text{ lam}_2$$

First a Detour: Definition of α -Equivalence

Here is a maybe-unfamiliar definition of α -equivalence, but it does define α -equivalence:

$$\frac{}{a \approx a} \text{ var}$$

swap all variables, be they free, bound or binding, e.g.

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

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

$$\frac{t \approx (a \ b) \bullet s \quad a \notin FV(s)}{\lambda a. t \approx \lambda b. s} \text{ lam}_2$$

What is so Special about Swappings?

Problem: substitution does not respect α -equivalence, e.g.

$\lambda a.b$

$\lambda c.b$

What is so Special about Swappings?

Problem: substitution does not respect α -equivalence, e.g.

$$\begin{aligned} [b := a] \lambda a. b \\ = \lambda a. a \end{aligned}$$

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

What is so Special about Swappings?

Problem: substitution does not respect α -equivalence, e.g.

$$\begin{aligned} [b := a] \lambda a. b \\ = \lambda a. a \end{aligned}$$

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

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

What is so Special about Swappings?

Problem: substitution does not respect α -equivalence, e.g.

$$(b\ a) \cdot \lambda a.b \\ = \lambda b.a$$

$$(b\ a) \cdot \lambda c.b \\ = \lambda c.a$$

Swapping is a nice alternative: because it is less complicated!

What is so Special about Swappings?

Problem: substitution does not respect α -equivalence, e.g.

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

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

Swapping is a nice alternative: because it is less complicated!

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

What is so Special about Swappings?

Problem: substitution does not respect α -equivalence, e.g.

$(b\ a) \cdot \lambda a.b$

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

Preview:

In the next few slides we shall extend 'swappings' to 'lists of swappings'

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

also called **permutations**.

Swapping
less com

Unlike
have if $t \approx t$

then $(\sigma a) \cdot t \approx (\sigma a) \cdot t$.

Permutations on Atoms

A permutation (a list of pairs of atoms) **acts** on an atom as follows:

$$\begin{aligned} [] \cdot a &\stackrel{\text{def}}{=} a \\ ((a_1 a_2) :: \pi) \cdot a &\stackrel{\text{def}}{=} \begin{cases} a_1 & \text{if } \pi \cdot a = a_2 \\ a_2 & \text{if } \pi \cdot a = a_1 \\ \pi \cdot a & \text{otherwise} \end{cases} \end{aligned}$$

- $[]$ 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
- the **disagreement set** of two permutations π and π' is the set of atoms

$$ds(\pi, \pi') \stackrel{\text{def}}{=} \{a \mid \pi \cdot a \neq \pi' \cdot a\}$$

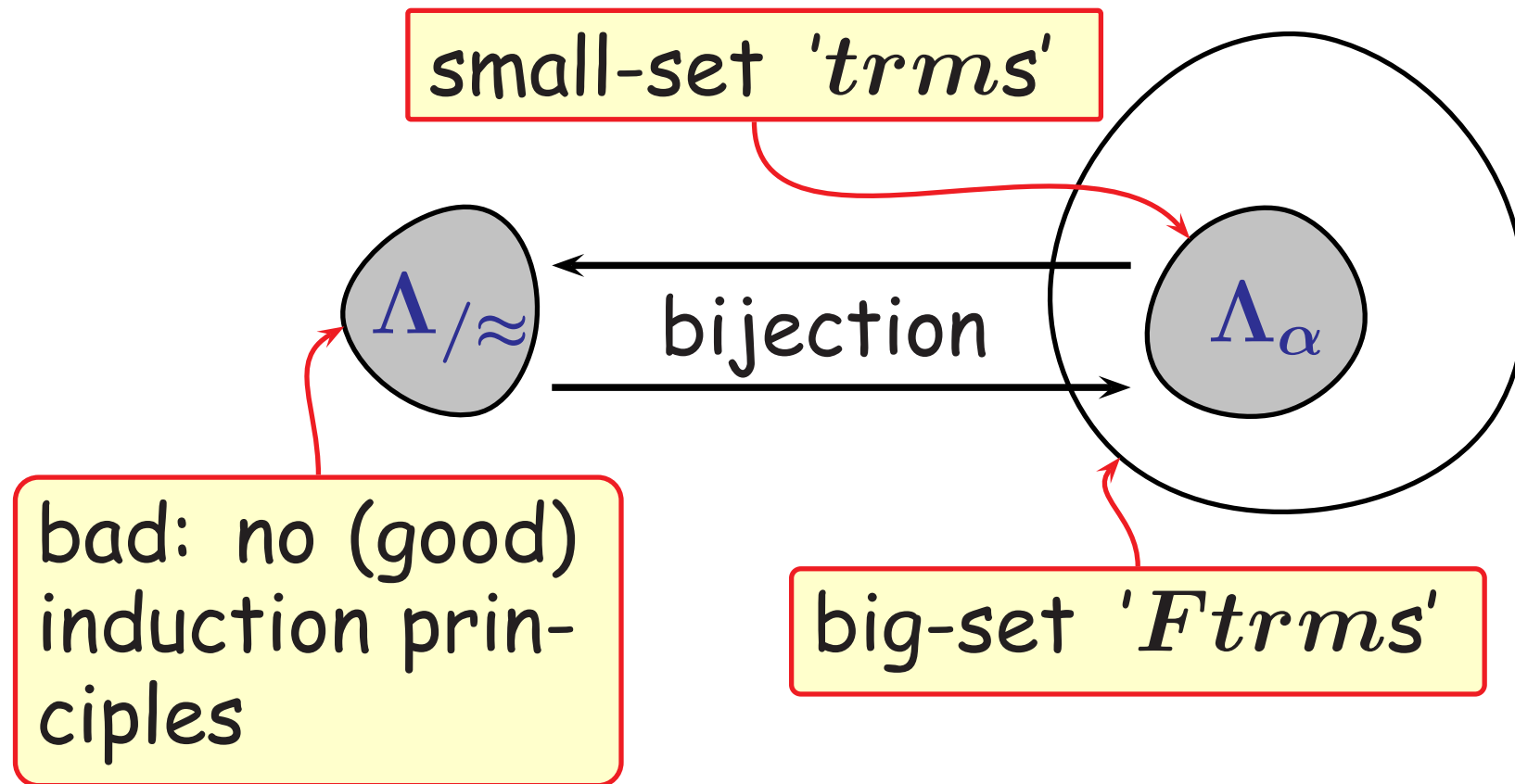
- $\pi \sim \pi' \stackrel{\text{def}}{=} ds(\pi, \pi') = \emptyset.$

Now in Earnest: The Trick

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

Now in Earnest: The Trick

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



Now in Earnest: The Trick

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

The small set Λ_α will consist of *trms* having the form

$$t ::= \text{Am}(a)^* \\ | \text{Pr } t \ t \\ | [a].t$$

be

in *atoms come from an infinite set, e.g. the natural numbers

principles

Big-Set / *Ftrms*

One step back: naive attempt for big-set

Ftrm ::=

	<i>Am</i> : <i>Atom</i>	'atoms'
	<i>Pr</i> : <i>Ftrm</i> × <i>Ftrm</i>	'pairs'
	<i>Se</i> : <i>Ftrm Set</i>	'α-eq-cl'



Big-Set / *Ftrms*

One step back: naive attempt for big-set

Ftrm ::=

	<i>Am</i> : <i>Atom</i>	'atoms'
	<i>Pr</i> : <i>Ftrm</i> × <i>Ftrm</i>	'pairs'
	<i>Se</i> : <i>Ftrm Set</i>	'α-eq-cl'

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

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



Big-Set / $Ftrms$

Better attempt for big-set

$Ftrm ::=$	Er	'error'
	$Am : Atom$	'atoms'
	$Pr : Ftrm \times Ftrm$	'pairs'
	$Se : Atom \rightarrow Ftrm$	' α -eq-cl'

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

"if $t' \in [\lambda a.t]_\alpha$ then 'yes' else Er "



Permutations for Big-Set

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

$$\begin{aligned}\pi \cdot Er &\stackrel{\text{def}}{=} Er \\ \pi \cdot Am(a) &\stackrel{\text{def}}{=} Am(\pi \cdot a) \\ \pi \cdot Pr(t_1, t_2) &\stackrel{\text{def}}{=} Pr(\pi \cdot t_1, \pi \cdot t_2) \\ \pi \cdot Se(fn) &\stackrel{\text{def}}{=} Se(\lambda a. \pi \cdot (fn(\pi^{-1} \cdot a)))\end{aligned}$$



Permutations for Big-Set

Ok, slowly: fn is a function from *Atoms* to *Ftrms*

$$fn = \lambda a.(fn\ a)$$

So we should have

$$\pi \bullet fn = \pi \bullet \lambda a.(fn\ a)$$

We want to permute all free atoms in fn
(= $\lambda a.(fn\ a)$)— a is clearly **not** free). Therefore

$$\lambda a.\pi \bullet (fn\ a)$$

is wrong, as it will also permute a (wherever it ends up).
However, if we substitute $\pi^{-1} \bullet a$ first, then the π that
is too much will go away.



Properties of Permutations

We can prove

- $(a\ a) \bullet t = t$

- $\pi^{-1} \bullet (\pi \bullet t) = t$

- $\pi \bullet t_1 = t_2$ iff $t_1 = \pi^{-1} \bullet t_2$

- etc.

Actually we can do all this abstractly by requiring:

- $[] \bullet x = x$

- $(\pi_1 @ \pi_2) \bullet x = \pi_1 \bullet (\pi_2 \bullet x)$

- $\pi_1 \sim \pi_2$ implies $\pi_1 \bullet x = \pi_2 \bullet x$

Axclasses in Isabelle

PTypes

UI

FSTypes

$$\blacksquare [] \bullet x = x$$

$$\blacksquare (\pi_1 @ \pi_2) \bullet x = \pi_1 \bullet (\pi_2 \bullet x)$$

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

Given an appropriate definition of a permutation action, everything is a PType (in particular our big-set and small-set).

Support and Not in the Support

We can define for every $PType$ the notion of support, a set of atoms (in a minute).

An old friend can be defined in terms of support:

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

So for our small-set, the support should coincide with $FV(x)$.

SUPPORT!!!

Something to chew on:

$\text{supp} : PType \rightarrow Atom Set$

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

In words: all atoms a where the set

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

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

Support of an Atom

What is the support of the atom c ?

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

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

Support of an Atom

What is the support of the atom c ?

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

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

$$a: \quad (a ?) \cdot c \neq c$$

Support of an Atom

What is the support of the atom c ?

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

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

$$a: \quad (a ?) \cdot c \neq c \quad \text{no}$$

$$b: \quad (b ?) \cdot c \neq c$$

Support of an Atom

What is the support of the atom c ?

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

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

$$a: \quad (a ?) \cdot c \neq c \quad \text{no}$$

$$b: \quad (b ?) \cdot c \neq c \quad \text{no}$$

$$c: \quad (c ?) \cdot c \neq c$$

Support of an Atom

What is the support of the atom c ?

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

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

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

Support of an Atom

What is the support of the atom c ?

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

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

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

Support of an Atom

What is the support of the atom c ?

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

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

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

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

Support of an Application

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

Support of an Application

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

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

Support of an Application

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

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

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 an Application

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

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

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

Support of an Application

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

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

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

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

Support of an Application

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

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

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

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

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

Support of an Application

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

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

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

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

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

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

Support of an Application

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

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

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

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

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

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

$$\text{supp}(t_1)$$

∪

$$\text{supp}(t_2)$$

Support of an Application

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

$$\begin{aligned} & \{a \mid \text{inf}\{b \mid (a b) \bullet t_1 \neq t_1 \vee (a b) \bullet t_2 \neq t_2\}\} \\ & \{a \mid \text{inf}(\{b \mid (a b) \bullet t_1 \neq t_1\} \cup \{b \mid (a b) \bullet t_2 \neq t_2\})\} \\ & \{a \mid \text{inf}\{b \mid (a b) \bullet t_1 \neq t_1\} \vee \text{inf}\{b \mid (a b) \bullet t_2 \neq t_2\}\} \\ & \{a \mid \text{inf}\{b \mid (a b) \bullet t_1 \neq t_1\}\} \cup \{a \mid \text{inf}\{b \mid (a b) \bullet t_2 \neq t_2\}\} \\ & \qquad \text{supp}(t_1) \qquad \qquad \cup \qquad \qquad \text{supp}(t_2) \end{aligned}$$

It is as Simple as That...

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

It is as Simple as That...

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

Proof: case $a = b$ clear

It is as Simple as That...

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

Proof: case $a \neq b$:

$$(1) \inf\{c \mid (a c) \bullet x \neq x\} \\ \inf\{c \mid (b c) \bullet x \neq x\}$$

from Ass. +Def. of $\#$

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

It is as Simple as That...

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. (1)

It is as Simple as That...

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. (1)

It is as Simple as That...

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. (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...

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. (1)
- (3') $\text{inf}\{c \mid (a c) \bullet x = x \wedge (b c) \bullet x = x\}$ f. (2')

It is as Simple as That...

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. (1)
- (3') $\text{inf}\{c \mid (a c) \bullet x = x \wedge (b c) \bullet x = x\}$ 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 .

It is as Simple as That...

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. (1)
- (3') $\text{inf}\{c \mid (a c) \bullet x = x \wedge (b c) \bullet x = x\}$ 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...

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. (1)
- (3') $\text{inf}\{c \mid (a c) \bullet x = x \wedge (b c) \bullet x = x\}$ 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...

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. (1)
- (3') $\text{inf}\{c \mid (a c) \bullet x = x \wedge (b c) \bullet x = x\}$ 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...

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. (1)
- (3') $\text{inf}\{c \mid (a c) \bullet x = x \wedge (b c) \bullet x = x\}$ 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...

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. (1)
- (3') $\text{inf}\{c \mid (a c) \bullet x = x \wedge (b c) \bullet x = x\}$ 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...

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. (1)

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

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

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

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

It is as Simple as That...

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. (1)
- (3') $\text{inf}\{c \mid (a c) \bullet x \neq x \vee (b c) \bullet x \neq x\}$ f. (2')
- (4) (i) $(a c) \bullet \pi_1 \sim \pi_2 \Rightarrow \pi_1 \bullet x = \pi_2 \bullet x$ $a c \in (3')$
(ii) $(b c) \bullet \pi_1 \sim \pi_2 \Rightarrow \pi_1 \bullet x = \pi_2 \bullet x$ $a c \in (3')$
- (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.

It is as Simple as That...

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. (1)
- (3') $\text{inf}\{c \mid (a c) \bullet x = x \wedge (b c) \bullet x = x\}$ 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.

Finite Support

PTypes

■ $\text{finite}(\text{supp}(x))$

We know what the support of an atom is, of an application. What about abstractions?
For $[a].t$ we have

FS

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

x

if the support of t is finite.

Finite Support

PTypes

■ $\text{finite}(\text{supp}(x))$

UI

Why finite support?

FSTypes

$(\forall x : FSType)(\exists a : Atm) a \# x$

Remember...

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

By induction on the structure of t_1 .

case variables:

subcase 1: $a \notin FV(t_2)$

by assumption

subcase 2: $a \notin FV(b)$

ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$ by IH

case abstractions (c sufficiently fresh):

$a \notin FV(s[a := t_2]) - \{c\}$ by IH

Done.

Finite Support

PTypes

■ $\text{finite}(\text{supp}(x))$

UI

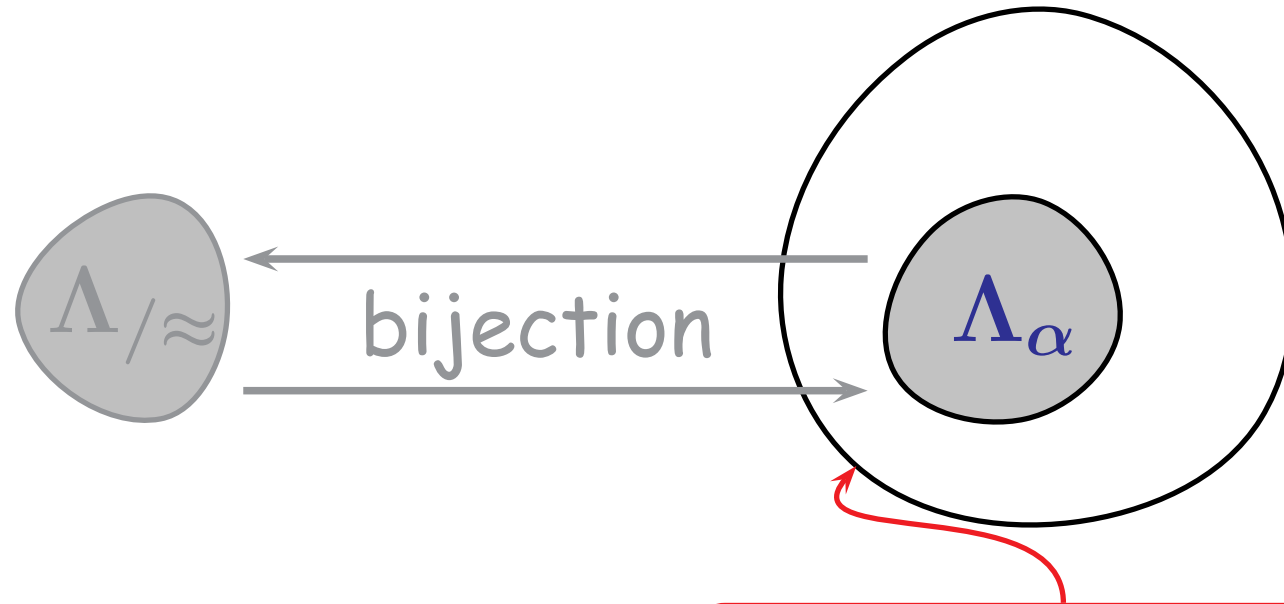
Why finite support?

FSTypes

$(\forall x : FSType)(\exists a : Atm) a \# x$

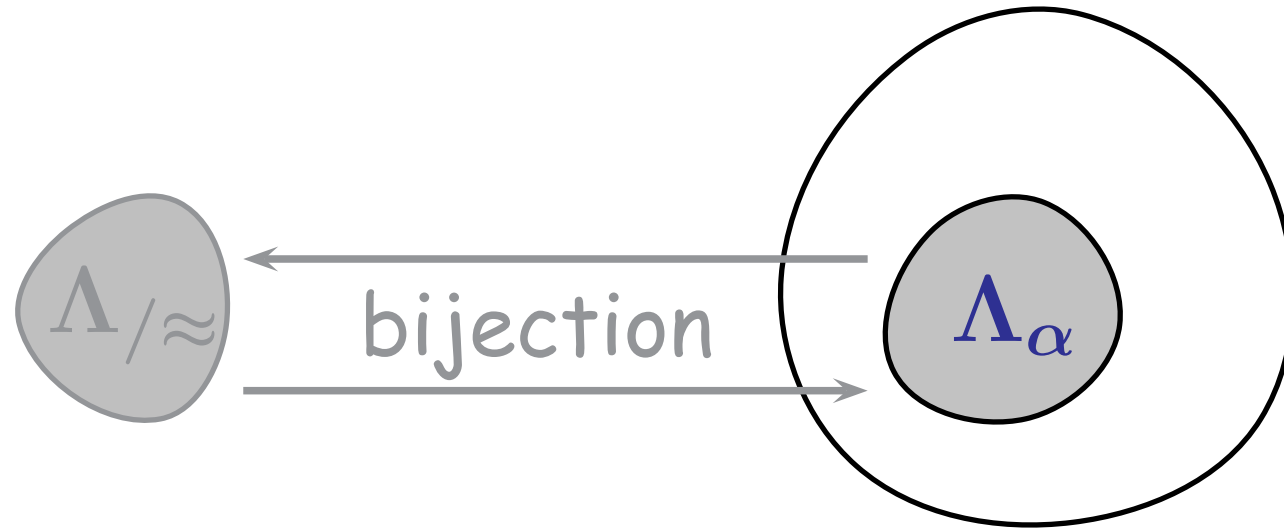
Atoms are finitely supported; so are pairs of FSTypes, lists, finite sets of FSTypes, ... Our small-set is finitely supported (set of free variables), but big-set isn't.

What About Small-Set?



$t ::= E_r$
| $A_m(a)$
| $Pr(t_1, t_2)$
| $Se(fn)$

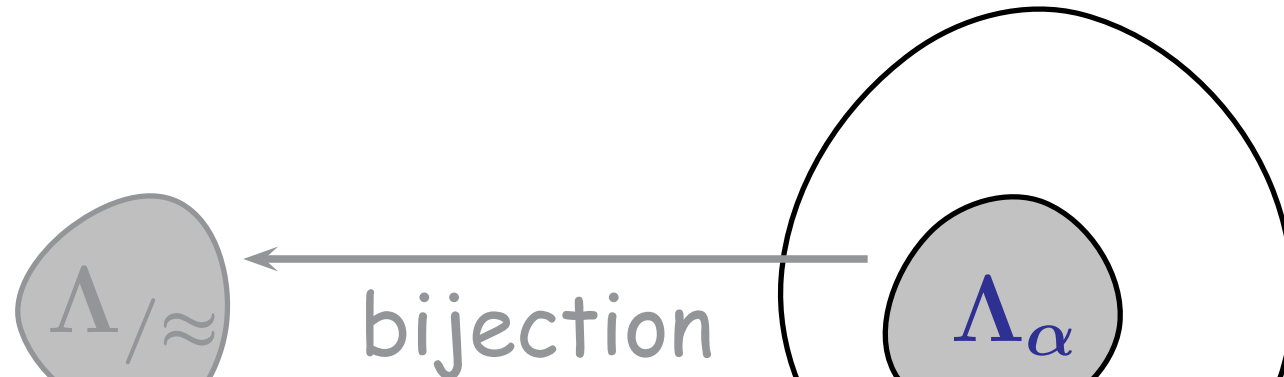
What About Small-Set?



For Λ_α , we are only interested in some very specific functions, namely

$[a].t \stackrel{\text{def}}{=} \text{Se } (\lambda b. \text{ if } a = b$
then t
else if $b \neq t$ then $(b a) \bullet t$ else $\text{Er})$

What About Small-Set?



Breath deeply: the user will never ever see anything about functions!

For Λ_α , we have specific functions, namely

$[a].t \stackrel{\text{def}}{=} \text{Se } (\lambda b. \text{ if } a = b \text{ then } t \text{ else if } b \neq t \text{ then } (b \ a) \bullet t \text{ else } \text{Er})$

Function $[a].t \text{ '}' \equiv \text{' } [\lambda a.t]_a$

$[a].t \stackrel{\text{def}}{=} \text{Se } (\lambda b. \text{ if } a = b$
then t
else if $b \# t$ then $(b a) \bullet t$ else $\text{Er})$

Function $[a].t \text{ '}' \equiv \text{' } [\lambda a.t]_{\alpha}$

$[a].t \stackrel{\text{def}}{=} \text{Se } (\lambda b. \text{ if } a = b$
then t
else if $b \# t$ then $(b a) \bullet t$ else $\text{Er})$

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

Function $[a].t \text{ '=='} [\lambda a.t]_a$

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

Se $(\lambda b. \text{if } a = b$
then $Pr(a, c)$
else if $b \neq Pr(a, c)$
then $(b a) \bullet Pr(a, c)$ else $Er)$

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

Function $[a].t$ '= $=$ ' $[\lambda a.t]_a$

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

$Se (\lambda b. \text{if } a = b$
 then $Pr(a, c)$
 else if $b \neq Pr(a, c)$
 then $(b a) \bullet Pr(a, c)$ else Er)

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

$[a].Pr(a, c)$ 'applied to' a 'gives' $Pr(a, c)$

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

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

Se $(\lambda b. \text{if } a = b$
then $Pr(a, c)$
else if $b \neq Pr(a, c)$
then $(b a) \bullet Pr(a, c)$ else Er)

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

$[a].Pr(a, c)$ 'applied to' a 'gives' $Pr(a, c)$

$[a].Pr(a, c)$ 'applied to' b 'gives' $Pr(b, c)$

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

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

$Se (\lambda b. \text{if } a = b$
 then $Pr(a, c)$
 else if $b \neq Pr(a, c)$
 then $(b a) \bullet Pr(a, c)$ else Er)

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

$[a].Pr(a, c)$ 'applied to' a 'gives' $Pr(a, c)$

$[a].Pr(a, c)$ 'applied to' b 'gives' $Pr(b, c)$

$[a].Pr(a, c)$ 'applied to' c 'gives' Er

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

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

$Se (\lambda b. \text{if } a = b$
 $\text{then } Pr(a, c)$
 $\text{else if } b \neq Pr(a, c)$
 $\text{then } (b a) \bullet Pr(a, c) \text{ else } Er)$

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

$[a].Pr(a, c)$ 'applied to' a 'gives' $Pr(a, c)$

$[a].Pr(a, c)$ 'applied to' b 'gives' $Pr(b, c)$

$[a].Pr(a, c)$ 'applied to' c 'gives' Er

$[a].Pr(a, c)$ 'applied to' d 'gives' $Pr(d, c)$

⋮

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

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

$Se (\lambda b. \text{if } a = b$
 then $Pr(a, c)$
 else if $b \neq Pr(a, c)$
 then $(b a) \bullet Pr(a, c)$ else Er)

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

$[a].Pr(a, c)$ 'applied to' a 'gives' $Pr(a, c)$ ' $\lambda a.(a c)$ '

$[a].Pr(a, c)$ 'applied to' b 'gives' $Pr(b, c)$ ' $\lambda b.(b c)$ '

$[a].Pr(a, c)$ 'applied to' c 'gives' Er

$[a].Pr(a, c)$ 'applied to' d 'gives' $Pr(d, c)$ ' $\lambda d.(d c)$ '

⋮

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

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

Se $(\lambda b. \text{if } a = b$
then $Pr(a, c)$
else if $b \neq Pr(a, c)$
then $(b a) \bullet Pr(a, c)$ else Er)

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

$[a].Pr(a, c)$ 'applied to' a 'gives' $Pr(a, c)$

$[a].Pr(a, c)$ 'applied to' b 'gives' $Pr(b, c)$

$[a].Pr(a, c)$ 'applied to' c 'gives' Er

$[a].Pr(a, c)$ 'applied to' d 'gives' $Pr(d, c)$

\vdots

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

' $\lambda a.(a c)$ '

' $\lambda b.(b c)$ '

' $\lambda d.(d c)$ '

\vdots

Properties of $[a].t$

Nominal Abstractions need to satisfy:

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

$$\blacksquare t_1 = t_2 \Leftrightarrow [a].t_1 = [a].t_2$$

$$\blacksquare a \neq b \Rightarrow t_1 = (a \ b) \bullet t_2 \wedge a \# t_2 \\ \Leftrightarrow [a].t_1 = [b].t_2$$

The last two axioms say that $[a].t$ behaves like an abstraction. Note that α -equivalence is encoded as equality on functions.

Properties of $[a].t$

Nominal Abstractions need to satisfy:

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

$$\blacksquare t_1 = t_2 \Leftrightarrow [a].t_1 = [a].t_2$$

$$\blacksquare a \neq b \Rightarrow t_1 = (a b) \bullet t_2 \wedge a \# t_2 \\ \Leftrightarrow [a].t_1 = [b].t_2$$

The last two axioms say that $[a].t$ behaves like an abstraction. Note that α -equivalence is encoded as

ed Remember the definition of α -equivalence from the detour:

$$\frac{t_1 \approx t_2}{\lambda a.t_1 \approx \lambda a.t_2} \quad \frac{t_1 \approx (a b) \bullet t_2 \quad a \notin FV(t_2)}{\lambda a.t_1 \approx \lambda b.t_2}$$

Properties of $[a].t$

Nominal Abstractions need to satisfy:

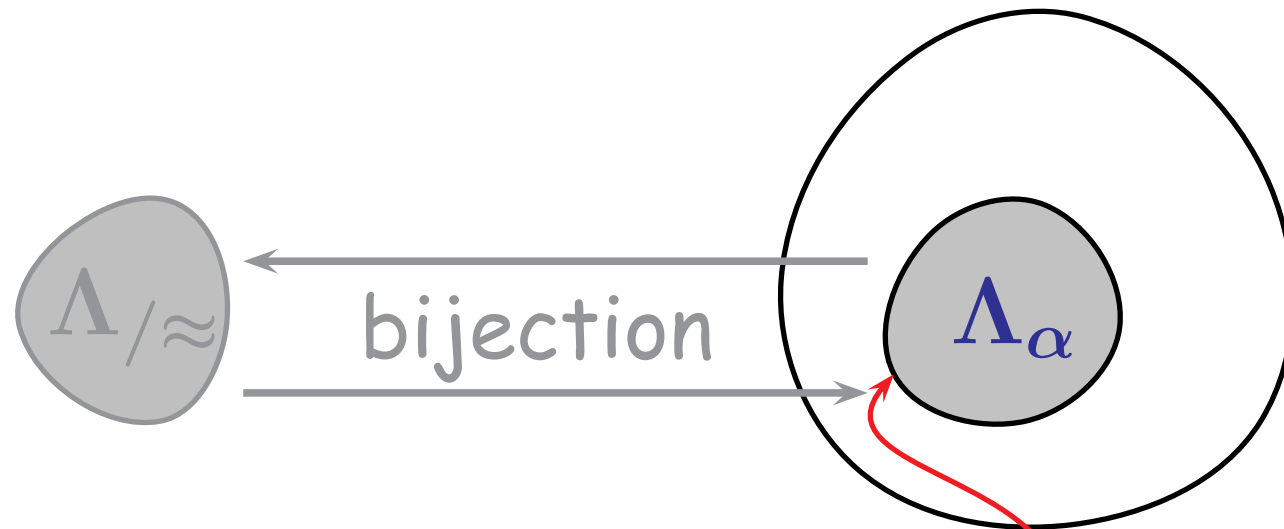
$$\blacksquare \pi \bullet ([a].t) = [\pi \bullet a].(\pi \bullet t)$$

$$\blacksquare t_1 = t_2 \Leftrightarrow [a].t_1 = [a].t_2$$

$$\blacksquare a \neq b \Rightarrow t_1 = (a \ b) \bullet t_2 \wedge a \# t_2 \\ \Leftrightarrow [a].t_1 = [b].t_2$$

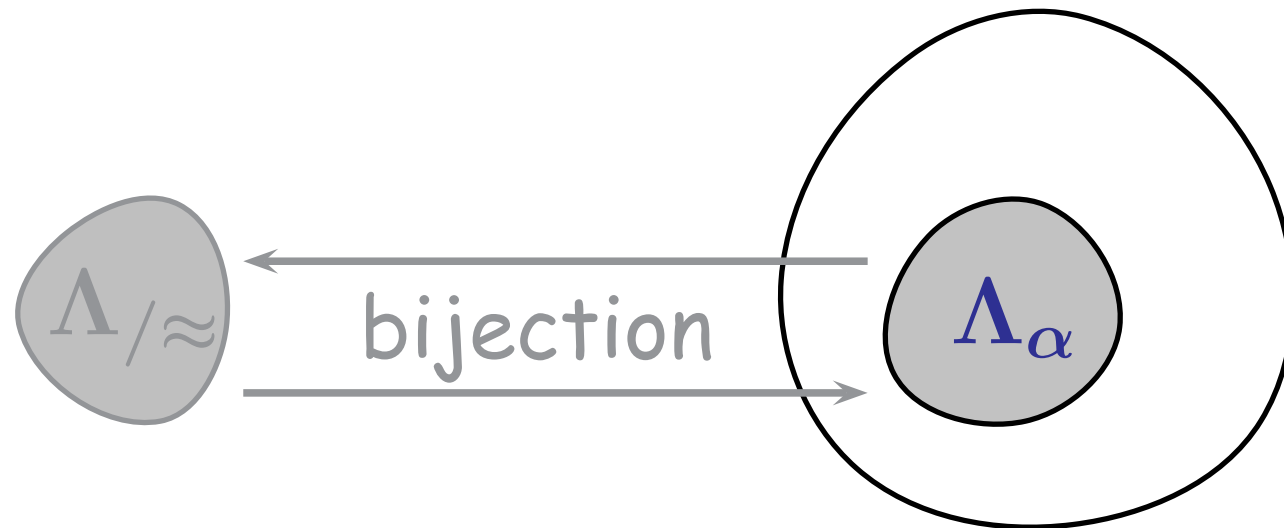
The last two axioms say that $[a].t$ behaves like an abstraction. Note that α -equivalence is encoded as equality on functions.

Definition of Small-Set



$t ::= Am(a)$
| $Pr(t_1, t_2)$
| $[a].t$

Definition of Small-Set



$$F(X) \stackrel{\text{def}}{=} AM \cup PR(X) \cup AS(X)$$

$$\Lambda_\alpha \stackrel{\text{def}}{=} \text{lfp}(F) = \bigcup_n F_n$$

$$\text{where } F_0 \stackrel{\text{def}}{=} F(\emptyset)$$

$$F_{n+1} \stackrel{\text{def}}{=} F(F_n)$$

Definition of Small-Set

Which also means that we have a familiar induction principle in place for Λ_α (over n).

$$F(X) \stackrel{\text{def}}{=} AM \cup PR(X) \cup AS(X)$$

$$\Lambda_\alpha \stackrel{\text{def}}{=} \text{lfp}(F) = \bigcup_n F_n$$

$$\text{where } F_0 \stackrel{\text{def}}{=} F(\emptyset)$$

$$F_{n+1} \stackrel{\text{def}}{=} F(F_n)$$

Bijection

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

$$\begin{aligned} q(a) &\stackrel{\text{def}}{=} Am(a) \\ q(t_1 t_2) &\stackrel{\text{def}}{=} Pr(q(t_1), q(t_2)) \\ q(\lambda a.t) &\stackrel{\text{def}}{=} [a].q(t) \end{aligned}$$

with the property

$$t_1 \approx t_2 \iff q(t_1) = q(t_2)$$

Bijection

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

$$\begin{aligned} q(a) & \text{ Believe me—I am not going to show this in more detail.} \\ q(t_1 \sigma_2) & = \sigma_1(q(\sigma_1), q(\sigma_2)) \\ q(\lambda a.t) & \stackrel{\text{def}}{=} [a].q(t) \end{aligned}$$

with the property

$$t_1 \approx t_2 \iff q(t_1) = q(t_2)$$

Induction on Λ_α

Since we defined Λ_α over n , we can prove:

$$(\forall a) P (Am(a)) x$$

$$(\forall t_1, t_2) P t_1 x \wedge P t_2 x \Rightarrow P (Pr(t_1, t_2)) x$$

$$(\forall a) a \# x \Rightarrow (\forall t) P t x \Rightarrow P ([a].t) x$$

$$(\forall t) P t x$$

Proof: By induction on n (the "stages" of constructing Λ_α).

But Remember...

Lemma: $a \notin FV(t_2)$ implies $a \notin FV(t_1[a := t_2])$

By induction on the structure of t_1 .

case variables:

subcase 1: $a \notin FV(t_2)$

by assumption

subcase 2: $a \notin FV(b)$

ok

case applications:

$a \notin FV(s_1[a := t_2]) \cup FV(s_2[a := t_2])$ by IH

case abstractions (c sufficiently fresh):

$a \notin FV(s[a := t_2]) - \{c\}$ by IH

Done.

Induction on Λ_α

$$(\forall a) P (Am(a)) x$$

$$(\forall t_1, t_2) P t_1 x \wedge P t_2 x \Rightarrow P (Pr(t_1, t_2)) x$$

$$(\forall a) a \# x \Rightarrow (\forall t) P t x \Rightarrow P ([a].t) x$$

$$(\forall t) P t x$$

Induction on Λ_α

$eqvt(P)$

$(\forall a) P (Am(a)) x$

$(\forall t_1, t_2) P t_1 x \wedge P t_2 x \Rightarrow P (Pr(t_1, t_2)) x$

$(\exists a) a \# x \wedge (\forall t) P t x \Rightarrow P ([a].t) x$

$(\forall t) P t x$

We can strengthen our induction principle if we know that the induction hypothesis is **equivariant**—invariant under renamings/permutations.

Equivariance

$$\text{eqvt}(P) \stackrel{\text{def}}{=} (\forall t : \Lambda_a) (\forall x : \mathit{FSType}) (\forall \pi) \\ P t x \Rightarrow P(\pi \bullet t)(\pi \bullet x)$$

In my inductions, I have an Λ_a -term as first argument and an FSType as second argument. Therefore, this slightly unusual definition for equivariance (this is a workaround, see complaints).

Some /Any-Property

Assuming $\text{eqvt}(P)$ then

$$(\exists a) a \# x \wedge (\forall t) P t x \Rightarrow P ([a].t) x$$

if and only if

$$(\forall a) a \# x \Rightarrow (\forall t) P t x \Rightarrow P ([a].t) x$$

Proof: Essentially you chose a fresh c , use the equivariance property of P and

$$a \# x \wedge c \# x \Rightarrow (a c) \bullet x = x$$

Some /Any-Property

Assuming $eqvt(P)$ then

$(\exists c)$ Equivariance is the justification for the Barendregt-variable-convention. x
if c He should have proved it!

$(\forall a) a \# x \Rightarrow (\forall t) P t x \Rightarrow P ([a].t) x$

Proof: Essentially you chose a fresh c , use the equivariance property of P and

$$a \# x \wedge c \# x \Rightarrow (a c) \bullet x = x$$

What I Have Done

- constructed an inductive set (Λ_α) that is bijective with the α -equated lambda-terms
- Λ_α has very much the feel of (named) lambda-terms equated up to α -equivalence
- We took advantage of the phenomenon that one and the same function can be defined in many different ways (e.g. $[a].Am(a) = [b].Am(b)$).

What I Have Done

- if we can prove equivariance for the IH, then we only need to prove the abstraction-case for **one** fresh atom
- formalised Henk's proof of Church-Rosser given at the beginning of the lambda-calculus book (includes a fair deal of inductions on the structure of α -equated lambda-terms)
- formalised Girard's SN-proof from "Proof and Types" (is extremely sensitive w.r.t. substitutions and α -equivalence classes; simultaneous substs.)

Conclusion

- Wouldn't it be nice to adapt the existing datatype package to hide all this from the user?
- I showed all this for the lambda-calculus, but of course everything can be generalised to any datatype involving binders, I claim.
- Even if it did not look so (that is purely my fault), it is really very easy stuff.

