Nominal Techniques: Quiz

Assuming that a and b are distinct variables, is it possible to find λ-terms $M_1..M_7$ that make the following pairs α-equivalent?

- $\lambda a.\lambda b.(M_1\, b)$ and $\lambda b.\lambda a.(a\, M_1)$
- $\lambda a.\lambda b.(M_2\, b)$ and $\lambda b.\lambda a.(a\, M_3)$
- $\lambda a.\lambda b.(b\, M_4)$ and $\lambda b.\lambda a.(a\, M_5)$
- $\lambda a.\lambda b.(b\, M_6)$ and $\lambda a.\lambda a.(a\, M_7)$

If there is one solution for a pair, can you describe all its solutions?

Nominal Techniques: Quiz

Assuming that a and b are distinct variables, is it possible to find λ-terms $M_1..M_7$ that make the fo

Don't be fooled by the question's innocent look: some lambda-calculus experts had problems with it. Also, the really interesting question is the one below.

Quiz will be solved on Friday. ;o)

- λ
- λ
- λ
- $\lambda a.\lambda b.(b\, M_6)$ and $\lambda a.\lambda a.(a\, M_7)$

If there is one solution for a pair, can you describe all its solutions?

Nominal Techniques Course

every day this week from 11:00 to 12:30 in Room C2

Christian Urban

University of Cambridge

What this course will be about

- syntax **with binders** (e.g. lambda-calculus)
- how to reason **formally** about binders
- how to use structural induction and structural recursion **conveniently**
- no de-Bruijn indices, no hand-waving using a Barendregt-style naming convention...
- a surprisingly **fresh** look at something quite familiar (unless you have already read the papers by Pitts, of course)

Relevance to Some Other Courses?

Two examples:

- Morrill: Type logical grammar (lambda-calculus)
- Koller et al: Computational semantics (accidental bindings, also gives an implementation of the lambda-calculus)
- probably others

Relevance to Some Other Courses?

Two examples:

- Morrill: Type logical grammar (lambda-calculus)
- Koller et al: Computational semantics (accidental bindings, also gives an implementation of the lambda-calculus)
- pr

spend one page of their reader on what we shall spend 7.5 hours

Relevance to Some Other Courses?

Two examples:

- Morrill: Type logical grammar (lambda-calculus)
- Koller et al: Computational semantics (accidental bindings, also gives an implementation of the lambda-calculus)
- probably others

What is the Problem

(Surely you know this, but just to make sure.)

- Mathematical version:

$$\int_0^1 x^2 + y\,dx = y + \frac{1}{3}$$

What is the Problem

(Surely you know this, but just to make sure.)

- Mathematical version:

$$\int_0^1 x^2 + y\,dx = y + \frac{1}{3}$$

naïvely applying $[y := x]$ gives the incorrect equation

$$\int_0^1 x^2 + x\,dx = x + \frac{1}{3}$$

What is the Problem

(Surely you know this, but just to make sure.)

- Computer-scientist version:

$$\lambda a.(b\,a)[b := a] \xrightarrow{\text{naïvely}} \lambda a.(a\,a)$$

Naïve substitution does not respect α-equivalence. What needs to be renamed is determined by subtle side-constraints. This makes formal reasoning hard.

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

Another Problem

(If you know it, you probably choose to ignore it.)

Assume we define the set Λ of (raw) lambda-terms **inductively** by the grammar:

$$\begin{array}{lcll} t & ::= & a & \text{variables} \\ & | & t\,t & \text{applications} \\ & | & \lambda a.t & \text{abstractions} \end{array}$$

Another Problem

(If you know it, you probably choose to ignore it.)

Assume we define the set Λ of (raw) lambda-terms **inductively** by the grammar:

$$\begin{array}{lcll} t & ::= & a & \text{variables} \\ & | & t\,t & \text{applications} \\ & | & \lambda a.t & \text{abstractions} \end{array}$$

We can easily define functions over Λ by structural recursion; for example

$$\begin{array}{lcl} \text{depth}\,(a) & \stackrel{\text{def}}{=} & 0 \\ \text{depth}\,(t\,t') & \stackrel{\text{def}}{=} & 1 + \max(\text{depth}(t), \text{depth}(t')) \\ \text{depth}\,(\lambda a.t) & \stackrel{\text{def}}{=} & 1 + \text{depth}(t) \end{array}$$

Another Problem

(If you know it, you probably choose to ignore it.)

Assume we define the set $\mathbf{\Lambda}$ of (raw) lambda-terms **inductively** by the grammar:

$$
\begin{array}{rcll}
t & ::= & a & \text{variables} \\
& | & t\,t & \text{applications} \\
& | & \lambda a.t & \text{abstractions}
\end{array}
$$

However, if we form the quotient-set $\mathbf{\Lambda}_{/=_\alpha}$ then what is the structural recursion principle?

$$
\begin{array}{rcll}
(a)\,[b := s] & \stackrel{\text{def}}{=} & \text{if } a = b \text{ then } s \text{ else } a & \\
(t\,t')\,[b := s] & \stackrel{\text{def}}{=} & (t[b := s])\,(t'[b := s]) & \\
(\lambda a.t)\,[b := s] & \stackrel{\text{def}}{=} & \lambda a.(t[b := s]) & \text{plus conditions}
\end{array}
$$

Another Problem

(If you know it, you probably choose to ignore it.)

Assume we define the set $\boldsymbol{\Lambda}$ of (raw) lambda-te … nmar:

Equating a set by a relation does **not** produce automatically an inductive set.

However, if we form the quotient-set $\boldsymbol{\Lambda}_{/=_\alpha}$ then what is the structural recursion principle?

$$
\begin{aligned}
(a)\,[b := s] &\stackrel{\text{def}}{=} \text{if } a = b \text{ then } s \text{ else } a\\
(t\,t')\,[b := s] &\stackrel{\text{def}}{=} (t[b := s])\,(t'[b := s])\\
(\lambda a.t)\,[b := s] &\stackrel{\text{def}}{=} \lambda a.(t[b := s]) \quad \text{plus conditions}
\end{aligned}
$$

Another Problem

(If you know it, you probably choose to ignore it.)

Assume we define the set $\mathbf{\Lambda}$ of (raw) lambda-terms **inductively** by the grammar:

$$\begin{array}{lcll} t & ::= & a & \text{variables} \\ & \mid & t\,t & \text{applications} \\ & \mid & \lambda a.t & \text{abstractions} \end{array}$$

However, if we form the quotient-set $\mathbf{\Lambda}_{/=_\alpha}$ then what is the structural recursion principle?

$$\begin{array}{rcl} (a)\,[b := s] & \stackrel{\text{def}}{=} & \text{if } a = b \text{ then } s \text{ else } a \\ (t\,t')\,[b := s] & \stackrel{\text{def}}{=} & (t[b := s])\,(t'[b := s]) \\ (\lambda a.t)\,[b := s] & \stackrel{\text{def}}{=} & \lambda a.(t[b := s]) \quad \text{plus conditions} \end{array}$$

Another Problem

(If you know it, you probably choose to ignore it.)

Of course, this can be turned into a proper definition — by recursion on the depth of α-equated lambda-terms.

But for this we need to lift the depth function from raw to α-equated lambda-terms, because clearly depth can also not be directly defined by structural recursion.

$$
\begin{array}{rcl}
(a)\,[b := s] & \stackrel{\text{def}}{=} & \text{if } a = b \text{ then } s \text{ else } a \\
(t\,t')\,[b := s] & \stackrel{\text{def}}{=} & (t[b := s])\,(t'[b := s]) \\
(\lambda a.t)\,[b := s] & \stackrel{\text{def}}{=} & \lambda a.(t[b := s]) \quad \text{plus conditions}
\end{array}
$$

De-Bruijn Indices

Of course, of course — all these problems would go away, if we had used de-Bruijn indices to encode bindings. Like

$$\begin{array}{ccc} \lambda a.\lambda b.(a\,b\,c) & \mapsto & \lambda\lambda(1\,0\,2) \\ \lambda a.\lambda b.(a\,(\lambda c.c\,a)\,b) & \mapsto & \lambda\lambda(1\,(\lambda(0\,2))\,0) \end{array}$$

But it just is a fact of life that de-Bruijn indices are hard to read and some important definitions are too far 'away' from their named counter-parts (see reader, page 3, for a definition of substitution with de-Bruijn indices). So we should attempt to do better.

De-Bruijn Indices

Of course, of course — all these problems would go away, if we had used de-Bruijn indices to enco

Aside: We insist on names. In case you were wondering what 'nominal' stands for... Well, that we insist on names.

$\boldsymbol{\lambda a}$ $\mathbf{0\,2)}$

$\boldsymbol{\lambda a.\lambda b}$ $\mathbf{(0\,2))\,0)}$

But it just is a fact of life that de-Bruijn indices are hard to read and some important definitions are too far 'away' from their named counter-parts (see reader, page 3, for a definition of substitution with de-Bruijn indices). So we should attempt to do better.

De-Bruijn Indices

Of course, of course — all these problems would go away, if we had used de-Bruijn indices to encode bindings. Like

$$\begin{array}{rcl} \lambda a.\lambda b.(a\,b\,c) & \mapsto & \lambda\lambda(1\,0\,2) \\ \lambda a.\lambda b.(a\,(\lambda c.c\,a)\,b) & \mapsto & \lambda\lambda(1\,(\lambda(0\,2))\,0) \end{array}$$

But it just is a fact of life that de-Bruijn indices are hard to read and some important definitions are too far 'away' from their named counter-parts (see reader, page 3, for a definition of substitution with de-Bruijn indices). So we should attempt to do better.

De-Bruijn Indices

Of course, of course — all these problems

There is a great deal of other work (e.g. HOAS) which alleviate some of these problems (no time to be more specific about them in this course*).

However, none of them has made life cosy and none of them has reached universal acceptance for formal reasoning with binders.

*HOAS would, for example, deserve its own course.

definition of substitution with de-Bruijn indices). So we should attempt to do better.

Plan for the Course

Tentative:

- Today: further motivation and some 'exercises' to become familiar with some of the main nominal concepts (e.g. definition of α-equivalence)
- Tuesday: Nominal Logic—a showcase for the nominal techniques
- Wednesday + Thursday: Justification for the nominal techniques (a bit mathematical)
- Friday: a nice application of the nominal techniques—unification of terms with binders

Barendregt-style Naming Convention

Roughly:

If lambda-terms $M_1, \ldots, M_n$ occur in a certain context, their bound variables are chosen to be different from the free variables.

or (my version)

Close your eyes and hope everything goes well.*

*not to be tried whilst driving

Weakening Property

...but sometimes eyes just cannot be closed :o(

Example: weakening property for the simply-typed lambda-calculus

$$\frac{a:\tau \in \Gamma}{\Gamma \vdash a:\tau} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1\, t_2 : \tau_2}$$

$$\frac{\Gamma, a:\tau_1 \vdash t:\tau_2}{\Gamma \vdash \lambda a.t : \tau_1 \to \tau_2}\; a \notin \mathsf{dom}(\Gamma)$$

Weakening Property

. . . but someti ... osed :o(

Example: wea ... simply-typed

Assume for the moment that Γ is a set of variable $\times$ type-pairs with some well-formedness constraints.

$$\frac{a : \tau \in \Gamma}{\Gamma \vdash a : \tau} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1\, t_2 : \tau_2}$$

$$\frac{\Gamma, a : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda a.t : \tau_1 \to \tau_2}\ a \notin \mathsf{dom}(\Gamma)$$

Weakening Property

...but sometimes eyes just cannot be closed :o(

Example: weakening property for the simply-typed lambda-calculus

$$\frac{a : \tau \in \Gamma}{\Gamma \vdash a : \tau} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1\, t_2 : \tau_2}$$

$$\frac{\Gamma, a : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda a.t : \tau_1 \to \tau_2}\ a \notin \mathsf{dom}(\Gamma)$$

If $\Gamma \vdash t : \tau$, then also $\Gamma, a : \tau' \vdash t : \tau$.

Weakening Property

…but sometimes eyes just cannot be closed :o(

Example: weakening property for the simply-typed lambda-calculus

$$\frac{a : \tau \in \Gamma}{\Gamma \vdash a : \tau} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1\, t_2 : \tau_2}$$

$$\frac{\Gamma, a : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda a.t : \tau_1 \to \tau_2}\; a \notin \mathrm{dom}(\Gamma)$$

$$(\forall \Gamma)(\forall t)(\forall \tau)\; \Gamma \vdash t : \tau \Rightarrow (\forall \tau')(\forall a \notin \mathrm{dom}(\Gamma))\; \Gamma, a : \tau' \vdash t : \tau$$

Raw Lambda-Terms? No!

This property does **not** hold for raw lambda-terms: since

$$\frac{a : \tau \vdash a : \tau}{\varnothing \vdash \lambda a.a : \tau \rightarrow \tau}$$

is derivable, but

$$a : \tau' \vdash \lambda a.a : \tau \rightarrow \tau$$

is not, because

$$\frac{\Gamma, a : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda a.t : \tau_1 \rightarrow \tau_2}\ a \notin \mathsf{dom}(\Gamma)$$

Raw Lambda-Terms? No!

This property does **not** hold for raw lambda-terms: since

$$\frac{a : \tau \vdash a : \tau}{\varnothing \vdash \lambda a.a : \tau \to \tau}$$

is derivab

We really mean weakening for α-equated lambda-terms.

$$a : \tau' \vdash \lambda a.a : \tau \to \tau$$

is not, because

$$\frac{\Gamma, a : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda a.t : \tau_1 \to \tau_2} \; a \notin \mathrm{dom}(\Gamma)$$

Let's Make This Explicit

Nobody usually bothers, but let's explicitly write $[t]_\alpha$ for the set of (raw) lambda-terms α-equivalent with t:

$$[t]_\alpha \stackrel{\text{def}}{=} \{t' \,|\, t' =_\alpha t\}\,.$$

Typing-rules for α-equated lambda-terms:

$$\frac{a : \tau \in \Gamma}{\Gamma \vdash [a]_\alpha : \tau} \qquad \frac{\Gamma \vdash [t_1]_\alpha : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash [t_2]_\alpha : \tau_1}{\Gamma \vdash [t_1\, t_2]_\alpha : \tau_2}$$

$$\frac{\Gamma, a : \tau_1 \vdash [t]_\alpha : \tau_2}{\Gamma \vdash [\lambda a.t]_\alpha : \tau_1 \rightarrow \tau_2}\; a \notin \text{dom}(\Gamma)$$

Let's Make This Explicit

Nobody usually bothers, but let's explicitly write ... erms α-equ...

Remember, we write $[t_1]_\alpha$, **but** we mean a set of terms $\{t_1, \ldots\}$ — namely the α-equivalence class of t_1.

Typing-rules for α-equated lambda-terms:

$$\frac{a : \tau \in \Gamma}{\Gamma \vdash [a]_\alpha : \tau} \qquad \frac{\Gamma \vdash [t_1]_\alpha : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash [t_2]_\alpha : \tau_1}{\Gamma \vdash [t_1\, t_2]_\alpha : \tau_2}$$

$$\frac{\Gamma, a : \tau_1 \vdash [t]_\alpha : \tau_2}{\Gamma \vdash [\lambda a.t]_\alpha : \tau_1 \rightarrow \tau_2}\ a \notin \mathrm{dom}(\Gamma)$$

Attempting the Proof

We proceed by rule induction and try to show that the predicate $\varphi(\Gamma; [t]_\alpha; \tau)$ given by

$$(\forall \tau')(\forall a' \not\in \mathsf{dom}(\Gamma))\ \Gamma, a' : \tau' \vdash [t]_\alpha : \tau$$

is closed under the axiom and the two inference rules. Interesting case:

$$\frac{\Gamma, a : \tau_1 \vdash [t]_\alpha : \tau_2}{\Gamma \vdash [\lambda a.t]_\alpha : \tau_1 \rightarrow \tau_2}\ a \not\in \mathsf{dom}(\Gamma)$$

Attempting the Proof

We proceed by rule
that the predicate

$$(\forall \tau')(\forall a' \notin \text{dom}$$

is closed under the
inference rules. Int

We know (for the premise):

1. $\varphi(\Gamma, a : \tau_1; [t]_\alpha; \tau_2)$
2. $a \notin \text{dom}(\Gamma)$

We have to prove:

$\varphi(\Gamma, a' : \tau'; [\lambda a.t]_\alpha; \tau_2)$

for **all** τ' and $a' \notin \text{dom}(\Gamma)$.

$$\frac{\Gamma, a : \tau_1 \vdash [t]_\alpha : \tau_2}{\Gamma \vdash [\lambda a.t]_\alpha : \tau_1 \rightarrow \tau_2} \; a \notin \text{dom}(\Gamma)$$

Attempting the Proof

We proceed by rule

that the predicate

$(\forall \tau')(\forall a' \notin \mathsf{dom}$

is closed under the

inference rules. Int

We know (for the premise):

1. $\varphi(\Gamma, a : \tau_1; [t]_\alpha; \tau_2)$
2. $a \notin \mathsf{dom}(\Gamma)$

We have to prove:

$\varphi(\Gamma, a' : \tau'; [\lambda a.t]_\alpha; \tau_2)$

for **all** τ' and $a' \notin \mathsf{dom}(\Gamma)$.

$$\frac{\Gamma, a : \tau_1 \vdash [t]_\alpha : \tau_2}{\Gamma \vdash [\lambda a.t]_\alpha : \tau_1 \to \tau_2} \; a \notin \mathsf{dom}(\Gamma)$$

But this fails for $a' = a$!

Moral of this Example

- Does this mean the weakening property does **not** hold for the simply-typed lambda-calculus?

  Clearly, **NO**!

  Just our simple-minded reasoning did not work. We have to take into account some facts about α-equivalent classes and their typing.

- And, closing your eyes is a non-starter.

Now We Start in Earnest

Some bookkeeping first.

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 Wednesday

Now We Start in Earnest

Some bookkeeping first.

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

Now We Start in Earnest

Some bookkeeping first.

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

Now We Start in Earnest

Some bookkeeping first.

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

Now We Start in Earnest

Some bookkeeping first.

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

Now We Start in Earnest

Some bookkeeping first.

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

Now We Start in Earnest

Some bookkeeping first.

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

Now We Start in Earnest

Some bookkeeping first.

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

Now We Start in Earnest

Some bookkeeping first.

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 $\mathbf{2}$ does not make sense

Now We Start in Earnest

Some bookkeeping first.

We introduce **atoms**. Everything that is **bound**, **bindin**… …ndent from t…

examp…

Why atoms? Because an operation we introduce shortly will act on atoms **only** and leaves everything else alone.

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

binding 2 does not make sense

Swappings

Recall the problem: substitution does not respect α-equivalence, e.g.

$$\boldsymbol{\lambda a.b} \qquad\qquad \boldsymbol{\lambda c.b}$$

Swappings

Recall the problem: substitution does not respect α-equivalence, e.g.

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

Swappings

Recall the problem: substitution does not respect α-equivalence, e.g.

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

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

Swappings

Recall the problem: substitution does not respect α-equivalence, e.g.

$$\begin{aligned} (b\,a)\bullet \lambda a.b &= \lambda b.a \end{aligned} \qquad \begin{aligned} (b\,a)\bullet \lambda c.b &= \lambda c.a \end{aligned}$$

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

Swappings

Recall the problem: substitution does 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

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

be they bound, binding or bindable

Swappings

Recall the problem: substitution does 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

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

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:

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

- $[]$ 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

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

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
- th[...]
  pe[...] s

$$\pi = \begin{pmatrix} a \mapsto b \\ b \mapsto c \\ c \mapsto a \end{pmatrix} \qquad \pi^{-1} = \begin{pmatrix} b \mapsto a \\ c \mapsto b \\ a \mapsto c \end{pmatrix}$$

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

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

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

Permutations on Atoms (ct.)

- the **composition** of two permutations is give
  π'@

  > for (finite) permutations this set is always finite (namely a subset of the atoms occurring π and π')

- the
  list reversal, written as π^{-1}, and
- the **disagreement set** of two permutations π and π' is the set of atoms

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

Permutations on Atoms (ct.)

- the
  give
  $\pi'@$
- the
  list
- the **disagreement set** of two permutations π and π' is the set of atoms

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

Example $ds((a\,c)(a\,b),(a\,b))$?

$$\begin{pmatrix} a \mapsto b \\ b \mapsto c \\ c \mapsto a \end{pmatrix} \quad \begin{pmatrix} a \mapsto b \\ b \mapsto a \\ c \mapsto c \end{pmatrix}$$

$$= \{b, c\}$$

Properties of Permutations

Here a, b and c are arbitrary atoms:

- $(b\,b){\bullet}a = a$, $(b\,c){\bullet}a = (c\,b){\bullet}a$
- $\pi^{-1}{\bullet}(\pi{\bullet}a) = a$
- $\pi{\bullet}a = b$ if and only if $a = \pi^{-1}{\bullet}b$
- $\pi_1 @ \pi_2{\bullet}a = \pi_1{\bullet}(\pi_2{\bullet}a)$
- $\pi{\bullet}((b\,c){\bullet}a) = (\pi{\bullet}b\;\;\pi{\bullet}c){\bullet}(\pi{\bullet}a)$

the first, second and last fact can be generalised to

- if $ds(\pi, \pi') = \varnothing$ then $\pi{\bullet}a = \pi'{\bullet}a$

Properties of Permutations

Here a, b and c are arbitrary atoms:

Preview: in the future, permutations will be completely characterised by the properties:

- $[]\bullet x = x$
- $\pi_1 @ \pi_2 \bullet x = \pi_1 \bullet (\pi_2 \bullet x)$
- if $ds(\pi, \pi') = \varnothing$ then $\pi \bullet x = \pi' \bullet x$

where x stands also for other 'things', not just atoms. Don't worry this will become clearer later on.

th

ge

- if $ds(\pi, \pi') = \varnothing$ then $\pi \bullet a = \pi' \bullet a$

Properties of Permutations

Here a, b and c are arbitrary atoms:

- $(b\,b)\bullet a = a$, $(b\,c)\bullet a = (c\,b)\bullet a$
- $\pi^{-1}\bullet(\pi\bullet a) = a$
- $\pi\bullet a = b$ if and only if $a = \pi^{-1}\bullet b$
- $\pi_1 @ \pi_2 \bullet a = \pi_1\bullet(\pi_2\bullet a)$
- $\pi\bullet((b\,c)\bullet a) = (\pi\bullet b\;\;\pi\bullet c)\bullet(\pi\bullet a)$

the first, second and last fact can be generalised to

- if $ds(\pi, \pi') = \varnothing$ then $\pi\bullet a = \pi'\bullet a$

Permutations on λ-Terms

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

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

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

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

(The attentive listener might like to prove these properties. You never know what you are being told.)

Permutations on λ-Terms

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

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

$$\pi\bullet(\lambda a.t) \stackrel{\text{def}}{=} \lambda(\pi\bullet a).(\pi\bullet t_2)$$

We have:

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

'we treat lambdas as if there were no binders'

(The attentive listener might like to prove these properties. You never know what you are being told.)

Permutations on λ-Terms

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

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

$$\pi \bullet (\lambda a.t) \stackrel{\text{def}}{=} \lambda(\pi \bullet a).(\pi \bullet t_2)$$

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

(The attentive listener might like to prove these properties. You never know what you are being told.)

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 very simple definition of α-equivalence (shown next)
- and don't get me started ;o)

(The attentive listener might like to prove these properties. You never know what you are being told.)

α-Equivalence

Consider the following four rules:

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

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

assuming $a \neq b$

α-Equivalence

Consider the following four rules:

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

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

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

$\lambda a.t \approx \lambda b.s$ 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*

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

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. This is achieved by freshness, written $a \# s$.

*there is a typo in the reader where this example is given

α-Equivalence

Consider the following four rules:

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

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

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

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

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.

Freshness

$$\frac{}{a \;\#\; b}\;\#\text{-atm} \qquad \frac{a \;\#\; t_1 \quad a \;\#\; t_2}{a \;\#\; t_1\, t_2}\;\#\text{-app}$$

$$\frac{}{a \;\#\; \lambda a.t}\;\#\text{-lam}_1 \qquad \frac{a \;\#\; t}{a \;\#\; \lambda b.t}\;\#\text{-lam}_2$$

assuming $a \neq b$

Be careful, we have defined two relations over **raw** lambda-terms. 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

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

is defined rather unsymmetrically. 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 $t_1 \approx t_3$

$\approx$ is an Equivalence

You might be an agnostic and notice that

because $\approx$ and $\#$ have very good properties:

- $t \approx t'$ then $\pi \bullet t \approx \pi \bullet t'$
- $a \# t$ then $\pi \bullet a \# \pi \bullet t$
- $t \approx \pi \bullet t'$ then $(\pi^{-1}) \bullet t \approx t'$
- $a \# \pi \bullet t$ then $(\pi^{-1}) \bullet a \# t$
- $a \# t$ and $t \approx t'$ then $a \# t'$

is d… hav…

(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 $t_1 \approx t_3$

Comparison with $=_\alpha$

Traditionally $=_\alpha$ is defined as

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

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

- with $\approx$ and $\#$ we never need to choose a 'fresh' atom (good for implementations and for nominal unification—wait until Friday)
- permutation respects both relations, whilst renaming-substitution does not

Does This Help?

...with our proof for the weakening property. Let's first extend the permutation operation to:

- sets of lambda-terms

  $$\pi\bullet\{t_1,\ldots,t_n\} \stackrel{\text{def}}{=} \{\pi\bullet t_1,\ldots,\pi\bullet t_n\}$$

- pairs

  $$\pi\bullet(x,y) \stackrel{\text{def}}{=} (\pi\bullet x,\pi\bullet y)$$

- types $\tau ::= X \mid \tau\rightarrow\tau$

  $$\pi\bullet\tau \stackrel{\text{def}}{=} \tau$$

Does This Help?

...with our proof for the weakening property. Let's first extend the permutation operation to:

- sets of lambda-terms

$$\pi \bullet \{t_1, \ldots, t_n\} \stackrel{\text{def}}{=} \{\pi \bullet t_1, \ldots, \pi \bullet t_n\}$$

- pairs

you are probably by now not surprised that we have:

- $t \in X$ if and only if $(\pi \bullet t) \in (\pi \bullet X)$
- $\pi \bullet [t]_\alpha = [\pi \bullet t]_\alpha$

Does This Help?

...with our proof for the weakening property. Let's first extend the permutation operation to:

- sets of lambda-terms

$$\pi\bullet\{t_1,\ldots,t_n\} \stackrel{\text{def}}{=} \{\pi\bullet t_1,\ldots,\pi\bullet t_n\}$$

- pairs

$$\pi\bullet(x,y) \stackrel{\text{def}}{=} (\pi\bullet x,\pi\bullet y)$$

- types $\tau ::= X \mid \tau\rightarrow\tau$

$$\pi\bullet\tau \stackrel{\text{def}}{=} \tau$$

Does This Help?

…with our proof for the weakening property. Let's first extend the permutation operation to:

- se … $t_n\}$
- po
- ty

$\pi \bullet \tau = \tau$

So given a typing-context Γ

$$\pi \bullet \Gamma$$

will always be a typing-context, while

$$\Gamma[a := b]$$

is only in some specific circumstances.

Equivariance of $\approx$ and $\#$

A relation (or predicate) is called **equivariant** provided it is preserved under permutations, that is its validity is invariant under permutations. For example:

$$t_1 \approx t_2 \quad \text{if and only if} \quad \pi \bullet t_1 \approx \pi \bullet t_2$$

$$a \# t \quad \text{if and only if} \quad \pi \bullet a \# \pi \bullet t$$

It seems, equivariance is an important concept when reasoning about properties involving binders.

... Also $\vdash$ and φ

- the typing relation is equivariant

$$\Gamma \vdash t : \tau \quad \Leftrightarrow \quad \pi\bullet\Gamma \vdash \pi\bullet t : \pi\bullet\tau$$

$$\boxed{\frac{a : \tau \in \Gamma}{\Gamma \vdash [a]_\alpha : \tau}} \quad \Leftrightarrow \quad \boxed{\frac{\pi\bullet(a : \tau) \in \pi\bullet\Gamma}{\pi\bullet\Gamma \vdash [\pi\bullet a]_\alpha : \pi\bullet\tau}}$$

- our induction-hypothesis is equivariant, i.e. $\varphi(\Gamma;[t]_\alpha;\tau) \Leftrightarrow \varphi(\pi\bullet\Gamma;\pi\bullet[t]_\alpha;\pi\bullet\tau)$

$$(\forall\tau')(\forall a' \notin \mathsf{dom}(\Gamma))\ \Gamma, a' : \tau' \vdash [t]_\alpha : \tau$$
$$\Leftrightarrow$$
$$(\forall\tau')(\forall a' \notin \mathsf{dom}(\pi\bullet\Gamma))\ \pi\bullet\Gamma, a' : \tau' \vdash \pi\bullet[t]_\alpha : \pi\bullet\tau$$

. . . Also $\vdash$ and φ

- the typing relation is equivariant

Be careful! The $\forall$-quantifiers are not allowed to quantify anything in π—if they do, we have to rename the quantified **meta**-variables. How this is done conveniently will be explained on Tuesday and Wednesday.

$t : \pi \bullet \tau$

$\in \pi \bullet \Gamma$

$]_\alpha : \pi \bullet \tau$

- our induction-hypothesis is equivariant, i.e. $\varphi(\Gamma; [t]_\alpha; \tau) \Leftrightarrow \varphi(\pi \bullet \Gamma; \pi \bullet [t]_\alpha; \pi \bullet \tau)$

$$(\forall \tau')(\forall a' \notin \mathrm{dom}(\Gamma))\ \Gamma, a' : \tau' \vdash [t]_\alpha : \tau$$

$$\Leftrightarrow$$

$$(\forall \tau')(\forall a' \notin \mathrm{dom}(\pi \bullet \Gamma))\ \pi \bullet \Gamma, a' : \tau' \vdash \pi \bullet [t]_\alpha : \pi \bullet \tau$$

Now the Proof

Case $a' = a$: from the premise we know

1. $\varphi(\Gamma, a : \tau_1; [t]_\alpha; \tau_2)$ 2. $a \notin \mathrm{dom}(\Gamma)$

Now the Proof

Case $a' = a$: from the premise we know

1. $\varphi(\Gamma, a : \tau_1; [t]_\alpha; \tau_2)$ 2. $a \not\in \mathrm{dom}(\Gamma)$

By equivariance we know

1′. $\varphi(\Gamma, b : \tau_1; [(a\,b)\bullet t]_\alpha; \tau_2)$ 2′. $b \not\in \mathrm{dom}(\Gamma)$

for any **fresh** atom b, i.e. one not occurring in Γ, t, or $\{a, a'\}$.

Now the Proof

Case $a' = a$: from the premise we know

1. $\varphi(\Gamma, a : \tau_1; [t]_\alpha; \tau_2)$ 2. $a \notin \mathrm{dom}(\Gamma)$

By equivariance we know

1′. $\varphi(\Gamma, b : \tau_1; [(a\,b) \bullet t]_\alpha; \tau_2)$ 2′. $b \notin \mathrm{dom}(\Gamma)$

for any **fresh** atom b, i.e. one not occurring in Γ, t, or $\{a, a'\}$.

This looks very much like we are closing our eyes again. But not quite! It very much depends on how easy it is to work with 'fresh'. Also, we do not need to explicitly give a b—its existence will be enough.

Now the Proof

Case $a' = a$: from the premise we know

1. $\varphi(\Gamma, a : \tau_1; [t]_\alpha; \tau_2)$ 2. $a \not\in \mathsf{dom}(\Gamma)$

By equivariance we know

1′. $\varphi(\Gamma, b : \tau_1; [(a\,b)\bullet t]_\alpha; \tau_2)$ 2′. $b \not\in \mathsf{dom}(\Gamma)$

for any **fresh** atom b, i.e. one not occurring in Γ, t, or $\{a, a'\}$.

By definition of φ we have $\forall a' \not\in \mathsf{dom}(\Gamma, b : \tau_1)$

3. $\Gamma, b : \tau_1, a' : \tau' \vdash [(a\,b)\bullet t]_\alpha : \tau_2$

Now the Proof

Case $a' = a$: from the premise we know

1. $\varphi(\Gamma, a : \tau_1; [t]_\alpha; \tau_2)$ 2. $a \notin \mathrm{dom}(\Gamma)$

By equivariance we know

1'. $\varphi(\Gamma, b : \tau_1; [(a\, b) \bullet t]_\alpha; \tau_2)$ 2'. $b \notin \mathrm{dom}(\Gamma)$

for any **fresh** atom b, i.e. one not occurring in Γ, t, or $\{a, a'\}$.

By definition of φ we have $\forall a' \notin \mathrm{dom}(\Gamma, b : \tau_1)$

3. $\Gamma, b : \tau_1, a' : \tau' \vdash [(a\, b) \bullet t]_\alpha : \tau_2$

By choice of b we can now apply the typing-rule and get

4. $\Gamma, a' : \tau' \vdash [\lambda b.(a\, b) \bullet t]_\alpha : \tau_1 \to \tau_2$

Now the Proof

Case $a' = a$: from the premise we know

1. $\varphi(\Gamma$

By equiv

1'. $\varphi(\Gamma$ … $\Gamma)$

for any … Γ, t, or
$\{a, a'\}$.

By defin … $_1)$

3. $\Gamma, b : \tau_1, a' : \tau' \vdash [(a\,b)\bullet t]_\alpha : \tau_2$

By choice of b we can now apply the typing-rule and get

4. $\Gamma, a' : \tau' \vdash [\lambda b.(a\,b)\bullet t]_\alpha : \tau_1 \to \tau_2$

But now

$$\lambda b.(a\,b)\bullet t \approx \lambda a.t$$

so we have

$$[\lambda b.(a\,b)\bullet t]_\alpha = [\lambda a.t]_\alpha$$

and **finally** we know that

$$\Gamma, a' : \tau' \vdash [\lambda a.t]_\alpha : \tau_1 \to \tau_2$$

holds in the case $a' = a$. Done. :o)

A Bird's Eye View

Old World	Nominal World
meta-language binders, quantifiers	**meta-language** binders, quantifiers
object-language HOAS FOAS	**object-language** FOAS

A Bird's Eye View

Old World	Nominal World
meta-language binders, quantifiers	**meta-language** binders, quantifiers
object-language HOAS / FOAS	**object-language** FOAS

lambda-calculus,
pi-calculus,
proof-theory,...

A Bird's Eye View

Old World	Nominal World
meta-language binders, quantifiers	**meta-language** binders, quantifiers
object-language HOAS, FOAS	**object-language** FOAS

A Bird's Eye View

Old World	Nominal World
meta-language binders, quantifiers	**meta-language** binders, quantifiers
object-language HOAS, FOAS	**object-language** NAS

A Bird's Eye View

Old World	Nominal World
meta-language binders, quantifiers	**meta-language** binders, quantifiers
object-language HOAS FOAS	**object-langu** NAS

Tomorrow

Two Points to Sleep Over

- if you need to rename binders:

  permutations behave much better than renaming-substitutions

- if you are trying to prove something about syntax with binders:

  equivariance seems to be the key