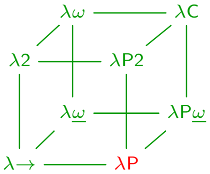
predicate logic & dependent types

Freek Wiedijk

Type Theory & Coq
2024–2025
Radboud University Nijmegen

September 27, 2024



function types become dependent

for predicate logic one needs to generalize

$$(\lambda x : A.\, M) \quad : \quad A \to B$$

function types

to

can contain x

$$(\lambda x : A.\, M) \quad : \quad \Pi x : A.\, \overset{\downarrow}{B}$$

dependent function types

$$\forall x : A.\, B$$

`forall x : A, B`

three different notations for the same type

we skip chapters 3 and 5 for now

inductive types

$\downarrow$

next week

program extraction

$\downarrow$

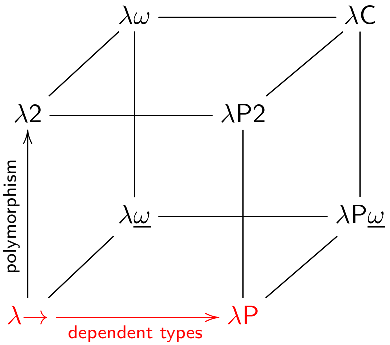
last week (briefly)

last week and this week

Curry-Howard correspondence:

propositional logic	$\lambda{\rightarrow}$	simple types
predicate logic	λP	dependent types
second order logic	$\lambda 2$	polymorphic types
beyond minimal logic	CIC	inductive types

dependent types in the lambda cube



$$A, B ::= a \mid A \to B \mid A \wedge B \mid A \vee B \mid \neg A \mid \top \mid \bot$$

$$\begin{array}{cc}
I{\to} & E{\to} \\
I{\wedge} & El{\wedge}\ \ Er{\wedge} \\
Il{\vee}\ \ Ir{\vee} & E{\vee} \\
I{\neg} & E{\neg} \\
I{\top} & E{\bot}
\end{array}$$

recap: simply typed lambda calculus

$$A, B ::= a \mid A \to B$$
$$M, N ::= x \mid M\,N \mid \lambda x : A.\, M$$

$$\frac{}{\Gamma \vdash x : A} \qquad \text{for } (x : A) \in \Gamma$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A.\, M) : A \to B} \qquad \frac{\Gamma \vdash F : A \to B \qquad \Gamma \vdash M : A}{\Gamma \vdash F M : B}$$

predicate logic

three predicate logics

- minimal predicate logic

$$\to \quad \forall$$

- constructive predicate logic

$$\to \quad \land \quad \lor \quad \neg \quad \top \quad \bot \quad \forall \quad \exists$$

- classical predicate logic

$$A \lor \neg A$$
$$\neg\neg A \to A$$

$$A, B ::= a \mid A \to B \mid A \wedge B \mid A \vee B \mid \neg A \mid \top \mid \bot$$

$$\begin{aligned} M, N &::= x \mid f(\overrightarrow{M}) \\ \overrightarrow{M} &::= \cdot \mid \overrightarrow{M}, N \\ A, B &::= p(\overrightarrow{M}) \mid A \to B \mid A \wedge B \mid A \vee B \mid \neg A \mid \top \mid \bot \mid \\ &\quad\ \forall x.\, A \mid \exists x.\, A \end{aligned}$$

$\forall$ and $\exists$ bind weakly

minimal propositional versus minimal predicate logic: rules

$$\dfrac{\begin{array}{c}[A^H]\\ \vdots\\ B\end{array}}{A \to B}\; I[H]{\to} \qquad\qquad \dfrac{\begin{array}{c}\vdots\\ A \to B\end{array} \quad \begin{array}{c}\vdots\\ A\end{array}}{B}\; E{\to}$$

$$\dfrac{\begin{array}{c}\vdots\\ A\end{array}}{\forall x.\, A}\; I\forall \qquad\qquad \dfrac{\begin{array}{c}\vdots\\ \forall x.\, A\end{array}}{A[x := M]}\; E\forall$$

variable condition of $I\forall$: x not free in available assumptions

same four rules with explicit assumptions

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,I{\to} \qquad\qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\,E{\to}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.\, A}\,I\forall \qquad\qquad \frac{\Gamma \vdash \forall x.\, A}{\Gamma \vdash A[x := M]}\,E\forall$$

variable condition of $I\forall$: x not free in Γ

rules for the existential quantifier

$$\dfrac{\begin{matrix}\vdots\\ A[x := M]\end{matrix}}{\exists x.\, A}\; I\exists \qquad\qquad \dfrac{\begin{matrix}\vdots\\ \exists x.\, A\end{matrix} \qquad \begin{matrix}\vdots\\ \forall x.\, A \to C\end{matrix}}{C}\; E\exists$$

variable condition of $E\exists$: x not free in C

example proof in minimal logic

$$
\dfrac{\dfrac{\dfrac{[\forall y.\, p(y)^H]}{p(x)}\, E\forall}{(\forall y.\, p(y)) \to p(x)}\, I[H]{\to}}{\forall x.\big((\forall y.\, p(y)) \to p(x)\big)}\, I\forall
$$

variable condition check:

at the $I\forall$ step the variable x does not occur in assumptions (there are no available assumptions at that point)

example with existential quantifier

$A \leftrightarrow B := (A \to B) \land (B \to A)$

$$(\exists x.\, p(x)) \leftrightarrow \neg(\forall x.\, \neg p(x))$$

not constructively valid

$$\begin{aligned} (\exists x.\, p(x)) &\to \neg(\forall x.\, \neg p(x)) \\ \neg(\forall x.\, \neg p(x)) &\to \neg\neg(\exists x.\, p(x)) \end{aligned}$$

example from left to right

$$\dfrac{\dfrac{\dfrac{[\exists x.\, p(x)^{H_0}] \qquad [\forall x.\, p(x) \to \bot^{H_1}]}{\bot}\, E\exists}{\neg(\forall x.\, \neg p(x))}\, I[H_1]\neg}{(\exists x.\, p(x)) \to \neg(\forall x.\, \neg p(x))}\, I[H_0]{\to}$$

variable condition check:

at the $E\exists$ step the variable x does not occur in $\bot$

example from right to left

$$
\dfrac{\dfrac{[\neg(\forall x.\, \neg p(x))^{H_0}] \qquad \dfrac{\dfrac{\dfrac{[\neg(\exists x.\, p(x))^{H_1}] \qquad \dfrac{[p(x)^{H_2}]}{\exists x.\, p(x)}\, I\exists}{\bot}\, E{\to}}{\neg p(x)}\, I[H_2]\neg}{\forall x.\, \neg p(x)}\, I\forall}{\dfrac{\bot}{\neg\neg(\exists x.\, p(x))}\, I[H_1]\neg}\, E\neg}{\neg(\forall x.\, \neg p(x)) \to \neg\neg(\exists x.\, p(x))}\, I[H_0]{\to}
$$

variable condition check:

at the $I\forall$ step the variable x does not occur in assumptions

types thus far

- atomic types
  $a, b, c, \dots$
- types of proof objects of propositions
  $a, b, c, \dots$
- function types
- datatypes
  - natural numbers
  - Booleans
  - lists
  - binary trees
  - ...

how are types introduced?

- free type variables

  STT = simple type theory

- in the context = 'axiomatic'

  PTSs = pure type systems: $\lambda{\to}$ λP $\lambda 2$ λC

  $$\Gamma \vdash M : A$$

  $$\mathsf{nat} : *,\ \mathsf{O} : \mathsf{nat},\ \mathsf{S} : \mathsf{nat} \to \mathsf{nat} \vdash \mathsf{S}\ (\mathsf{S}\ (\mathsf{S}\ \mathsf{O})) : \mathsf{nat}$$

  Coq: `Parameter`

- definitions ⟶ next week

  CIC = Calculus of Inductive Constructions

  $$E[\Gamma] \vdash M : A$$

  Coq: `Definition` `Inductive`

star and box

the sorts of the lambda cube:

$$
\begin{array}{ccl}
* & = & \text{the type of types} \\
\Box & = & \text{the type of } *
\end{array}
$$

the sort $*$ is a kind
$=$ third of four levels

$$
\begin{array}{r}
\text{object} : \text{type} \\
\text{type} : \text{kind} \\
\text{kind} : \Box
\end{array}
$$

$$\text{object} : \text{type} : \text{kind} : \Box$$

$$\mathsf{S} : \text{nat} \to \text{nat} : * : \Box$$

example contexts

natural numbers:

$$\begin{aligned} \mathsf{nat} &: * \\ \mathsf{O} &: \mathsf{nat} \\ \mathsf{S} &: \mathsf{nat} \to \mathsf{nat} \\ \mathsf{add} &: \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat} \end{aligned}$$

Booleans:

$$\begin{aligned} \mathsf{bool} &: * \\ \mathsf{true} &: \mathsf{bool} \\ \mathsf{false} &: \mathsf{bool} \end{aligned}$$

lists

lists of Booleans:

$$\begin{aligned} \mathsf{list} &: * \\ \mathsf{nil} &: \mathsf{list} \\ \mathsf{cons} &: \mathsf{bool} \to \mathsf{list} \to \mathsf{list} \\ \mathsf{hd} &: \mathsf{list} \to \mathsf{bool} \\ \mathsf{tl} &: \mathsf{list} \to \mathsf{list} \\ \mathsf{append} &: \mathsf{list} \to \mathsf{list} \to \mathsf{list} \\ \mathsf{reverse} &: \mathsf{list} \to \mathsf{list} \end{aligned}$$

cons has two arguments
(the head and tail to be put together)

trueslist

the function 'trueslist' maps a number n to a list of length n consisting of n copies of 'true'

$$\begin{aligned}
\text{trueslist O} &= \text{nil} \\
\text{trueslist (S O)} &= \text{cons true nil} \\
\text{trueslist (S (S O))} &= \text{cons true (cons true nil)} \\
\text{trueslist (S (S (S O)))} &= \text{cons true (cons true (cons true nil))} \\
&\dots
\end{aligned}$$

$$\text{trueslist} \;:\; \text{nat} \to \text{list}$$

vectors and truesvec

vectors of Booleans:

$$\mathsf{vec} : \mathsf{nat} \to *$$

$\mathsf{vec}\ \mathsf{O} : *$	vectors of length 0
$\mathsf{vec}\ (\mathsf{S}\ \mathsf{O}) : *$	vectors of length 1
$\mathsf{vec}\ (\mathsf{S}\ (\mathsf{S}\ \mathsf{O})) : *$	vectors of length 2
$\dots$	

$$\mathsf{trueslist} : \mathsf{nat} \to \mathsf{list}$$

$$\mathsf{truesvec} : \mathsf{nat} \to \mathsf{vec}\ n$$

$$\mathsf{truesvec} : \Pi n : \mathsf{nat}.\, \mathsf{vec}\ n$$

vnil and vcons

$$\text{vnil} : \text{vec } \mathsf{O}$$

$$\text{vcons} : \text{nat} \to \text{bool} \to \text{vec } n \to \text{vec } (\mathsf{S}\ n)$$

$$\text{vcons} : \Pi n : \text{nat}.\ \text{bool} \to \text{vec } n \to \text{vec } (\mathsf{S}\ n)$$

vcons has three arguments!

$$\text{truesvec } (\mathsf{S}\ (\mathsf{S}\ \mathsf{O})) \ : \ \text{vec } (\mathsf{S}\ (\mathsf{S}\ \mathsf{O}))$$

$$\text{truesvec } (\mathsf{S}\ (\mathsf{S}\ \mathsf{O})) = \text{vcons } (\mathsf{S}\ \mathsf{O}) \text{ true } (\text{vcons } \mathsf{O} \text{ true vnil})$$

types when applying vcons

$$\text{truesvec (S O)} = \text{vcons O true vnil}$$

$$\begin{aligned}
\text{truesvec} &: \Pi n : \text{nat}.\, \text{vec}\ n \\
\text{truesvec (S O)} &: \text{vec (S O)} \\[1em]
\text{vcons} &: \Pi n : \text{nat}.\, \text{bool} \to \text{vec}\ n \to \text{vec (S}\ n) \\
\text{O} &: \text{nat} \\
\text{vcons O} &: \text{bool} \to \text{vec O} \to \text{vec (S O)} \\
\text{true} &: \text{bool} \\
\text{vcons O true} &: \text{vec O} \to \text{vec (S O)} \\
\text{vnil} &: \text{vec O} \\
\text{vcons O true vnil} &: \text{vec (S O)}
\end{aligned}$$

Coq

BHK-interpretation

proof of $A \to B$	function from proofs of A to proofs of B
proof of $\neg A$	function from proofs of A to proofs of $\bot$
proof of $A \wedge B$	pair of a proof of A and a proof of B
proof of $A \vee B$	either a proof of A and a proof of B
proof of $\forall x.\, A$	dependent function from objects to proofs of A
proof of $\exists x.\, A$	dependent pair of an object and a proof of A

dependent $=$ A is not fixed but *depends* on the object x

$$A \vee B \;\leftrightarrow\; \exists x \in \{\mathsf{left}, \mathsf{right}\}.\, (x = \mathsf{left} \to A) \wedge (x = \mathsf{right} \to B)$$

Curry-Howard correspondence

type theory	Coq	minimal logic
$A \to B$	`A -> B`	$A \to B$
$\Pi x : D.\, A$	`forall x : D, A`	$\forall x.\, A$

$D : *$ is called `Terms` in the notes

proof term		type	statement proved
H	:	$A \to B$	$A \to B$
H'	:	A	A
$H H'$	:	B	B
H	:	$\Pi x : D.\, P x$	$\forall x.\, P(x)$
$H M$	:	$P M$	$P(M)$

Coq version of the first example

```
Parameter D : Set.
Parameter p : D -> Prop.

Lemma four :
  forall x : D,
  (forall y : D, p y) ->
  p x.
intros x H.
apply H.
Qed.

Print four.
```

```
four =
fun (x : D) (H : forall y : D, p y)
=> H x
     : forall x : D,
       (forall y : D, p y) -> p x

Arguments four _ _%function_scope
```

$$\dfrac{\dfrac{\dfrac{[\forall y.\, p(y)^H]}{p(x)}\, E\forall}{(\forall y.\, p(y)) \to p(x)}\, I[H]\!\to}{\forall x.\, (\forall y.\, p(y)) \to p(x)}\, I\forall$$

$$(\lambda x : D.\, \lambda H : (\Pi y : D.\, py).\, Hx) : \Pi x : D.\, (\Pi y : D.\, py) \to px$$

understanding the proof term

$$\forall x.\, (\forall y.\, p(y)) \to p(x)$$

$$\lambda x : D.\, \lambda H : (\Pi y : D.\, py).\, Hx$$

the second example in Coq, from left to right

```
Parameter D : Set.
Parameter p : D -> Prop.

Lemma five :
  (exists x : D, p x) ->
  ~ (forall x : D, ~ p x).
intros H0 H1.
elim H0.
apply H1.
Qed.
```

$$\dfrac{\dfrac{\dfrac{[\exists x.\, p(x)^{H_0}] \qquad [\forall x.\, p(x) \to \bot^{H_1}]}{\bot}\, E\exists}{\neg(\forall x.\, \neg p(x))}\, I[H_1]\neg}{(\exists x.\, p(x)) \to \neg(\forall x.\, \neg p(x))}\, I[H_0]{\to}$$

the second example in Coq, from right to left

```
Parameter D : Set.
Parameter p : D -> Prop.

Lemma six :
  ~ (forall x : D, ~ p x) ->
  ~ ~ (exists x : D, p x).
intros H0 H1.
apply H0.
intros x H2.
apply H1.
exists x.
apply H2.
Qed.
```

$$
\dfrac{\dfrac{[\neg(\forall x.\, \neg p(x))^{H_0}] \qquad \dfrac{\dfrac{\dfrac{[\neg(\exists x.\, p(x))^{H_1}] \qquad \dfrac{[p(x)^{H_2}]}{\exists x.\, p(x)}\, I\exists}{\bot}\, E{\rightarrow}}{\neg p(x)}\, I[H_2]\neg}{\forall x.\, \neg p(x)}\, I\forall}{\dfrac{\bot}{\neg\neg(\exists x.\, p(x))}\, I[H_1]\neg}\, E\neg}{\neg(\forall x.\, \neg p(x)) \rightarrow \neg\neg(\exists x.\, p(x))}\, I[H_0]{\rightarrow}
$$

proof rules versus Coq tactics

$I{\rightarrow}\ \ I\forall$	`intro` `intros`
$E{\rightarrow}\ \ E\forall$	`apply`
$E\wedge\ \ E\vee\ \ E\exists$	`elim` `destruct` intro-patterns
$I\wedge$	`split`
$I\vee$	`left` `right`
$I\exists$	`exists`

detours

proof normalization

detour = introduction rule directly followed by a elimination rule for the same connective

cut = corresponding notion in sequent calculus

can be eliminated $\longrightarrow$ reduction of the proof term

detour for implication:

'prove a lemma A and then prove B using this lemma'
detour elimination is inlining the proof$_1$ of the lemma everywhere

$$\dfrac{\dfrac{\begin{matrix}[A^H]\\ \vdots_2 \\ B\end{matrix}}{A \to B}\,I[H]{\to} \qquad \begin{matrix}\vdots_1\\ A\end{matrix}}{B}\,E{\to} \qquad \longrightarrow_\beta \qquad \begin{matrix}\vdots_1\\ A\\ \vdots_2\\ B\end{matrix}$$

detour for the universal quantifer

generalize the statement $A[x := M]$ to A with arbitrary x
elimination is specializing the proof$_1$ to M

$$\dfrac{\dfrac{\begin{matrix}\vdots_1 \\ A\end{matrix}}{\forall x.\, A}\, I\forall}{A[x := M]}\, E\forall \qquad \longrightarrow_\beta \qquad \begin{matrix}\vdots_1[x := M] \\ A[x := M]\end{matrix}$$

$$(\lambda x : D.\, H_1)M \to_\beta H_1[x := M]$$

STT versus λP syntax

STT:

$$
\begin{aligned}
A, B &::= a \mid A \to B \\
M, N &::= x \mid M\,N \mid \lambda x : A.\, M \\
\Gamma &::= \cdot \mid \Gamma,\, x : A \\
\mathcal{J} &::= \Gamma \vdash M : A
\end{aligned}
$$

λP:

$$
\begin{aligned}
s &::= * \mid \square \\
M, N, A, B &::= x \mid M\,N \mid \lambda x : A.\, M \mid \overbrace{\Pi x : A.\, B}^{A \to B} \mid s \\
\Gamma &::= \cdot \mid \Gamma,\, x : A \\
\mathcal{J} &::= \Gamma \vdash M : A
\end{aligned}
$$

$$
\overbrace{a : *,\ b : *,\ f : a \to b,\ x : a}^{\Gamma} \vdash \overbrace{f\,x}^{M} : \overbrace{b}^{A}
$$

STT versus λP

STT

3 rules

$$x \mid MN \mid \lambda x : A.\, M$$

λP

7 rules

$$x \mid MN \mid \lambda x : A.\, M \mid \Pi x : A.\, M \mid *$$

box does not have a type
$\longrightarrow$ no rule for box

variables have two rules

conversion rule

typing rules

STT

λP

$$\frac{}{\vdash * : \Box}$$

axiom rule

start rule

$$\frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\, B : s}$$

product rule

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.\, M : A \to B}$$

$$\frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.\, B : s}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B}$$

abstraction rule

typing rules (continued)

STT

$$\frac{\Gamma \vdash F : A \to B \qquad \Gamma \vdash M : A}{\Gamma \vdash FM : B}$$

λP

$$\frac{\Gamma \vdash F : \Pi x : A.\, B \qquad \Gamma \vdash M : A}{\Gamma \vdash FM : B[x := M]}$$

application rule

$$\begin{aligned} \mathsf{truesvec} &: \Pi n : \mathsf{nat}.\, \mathsf{vec}\ n \\ \mathsf{truesvec}\ (\mathsf{S}\ (\mathsf{S}\ \mathsf{O})) &: \mathsf{vec}\ (\mathsf{S}\ (\mathsf{S}\ \mathsf{O})) \end{aligned}$$

typing rules (continued)

STT	λP
(context Γ is a set)	(context Γ is a list)
	correct: $a : *,\ x : a \vdash x : a$
	incorrect: $x : a,\ a : * \vdash x : a$
$\dfrac{}{\Gamma \vdash x : A}\ (x : A) \in \Gamma$	$\dfrac{\Gamma \vdash A : s}{\Gamma,\ x : A \vdash x : A}$
	variable rule
	$\dfrac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma,\ y : B \vdash M : A}$
	weakening rule

the final typing rule

$$\mathsf{vecappend} : \Pi n_1 : \mathsf{nat}.\, \Pi n_2 : \mathsf{nat}.\; \mathsf{vec}\; n_1 \to \mathsf{vec}\; n_2 \to \mathsf{vec}\; (\mathsf{add}\; n_1\; n_2)$$

$\mathsf{vecappend}\; 3\; 4\;\; v_1\; v_2 : \mathsf{vec}\; (\mathsf{add}\; 3\; 4)$

$\mathsf{vecappend}\; 3\; 4\;\; v_1\; v_2 : \mathsf{vec}\; 7$

λP

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A' : s}{\Gamma \vdash M : A'}\; A =_\beta A'$$

conversion rule

A and A' are convertible

$=_\beta$ is defined on preterms

λP on one slide

$$M, F, A, B ::= x \mid FM \mid \lambda x : A.\, M \mid \Pi x : A.\, B \mid s$$

$$s ::= * \mid \Box$$

$$\frac{}{\vdash * : \Box} \qquad \frac{\Gamma \vdash A : s}{\Gamma,\, x : A \vdash x : A} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma,\, y : B \vdash M : A}$$

$$\frac{\Gamma,\, x : A \vdash M : B \quad \Gamma \vdash \Pi x : A.\, B : s}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B} \qquad \frac{\Gamma \vdash A : * \quad \Gamma,\, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\, B : s}$$

$$\frac{\Gamma \vdash F : \Pi x : A.\, B \quad \Gamma \vdash M : A}{\Gamma \vdash FM : B[x := M]} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash A' : s}{\Gamma \vdash M : A'}\; A =_\beta A'$$

the lambda cube

only the product rules differ:

$\lambda{\to}$

$$\frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi x : A.\, B : *}$$

λP

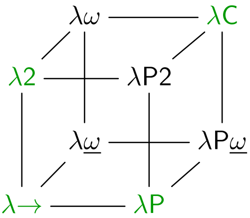
$$\frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\, B : s}$$

λ2

$$\frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi x : A.\, B : *}$$

λC

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.\, B : s_2}$$



pure type systems

$$\frac{}{\vdash s_1 : s_2}\ (s_1, s_2) \in \mathcal{A}$$

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.\, B : s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

	$\mathcal{S}$ sorts	$\mathcal{A}$ axioms	$\mathcal{R}$ rules
$\lambda *$	$\{*\}$	$\{(*, *)\}$	$\{(*, *, *)\}$
$\lambda\!\to$	$\{*, \Box\}$	$\{(*, \Box)\}$	$\{(*, *, *)\}$
λP	$\{*, \Box\}$	$\{(*, \Box)\}$	$\{(*, *, *), (*, \Box, \Box)\}$
λC	$\{*, \Box\}$	$\{(*, \Box)\}$	$\{(*, *, *), (*, \Box, \Box), (\Box, *, *), (\Box, \Box, \Box)\}$
λPRED	$\{*_s, \Box_s, *_p, \Box_p\}$	$\{(*_s, \Box_s), (*_p, \Box_p)\}$	$\{(*_s, *_s, *_s), (*_s, \Box_s, \Box_s), (*_p, *_p, *_p), (*_s, *_p, *_p)\}$

example PTS type derivation

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{}{\vdash * : \square}
      }{a : * \vdash a : *}
      \quad
      \dfrac{
        \dfrac{}{\vdash * : \square}
      }{a : * \vdash a : *}
    }{a : *,\, x : a \vdash a : *}
  }{a : *,\, x : a,\, y : a \vdash y : a}
  \quad
  \dfrac{\substack{\text{(next slide)} \\ \vdots}}{\color{red}{a : *,\, x : a \vdash a \to a : *}}
}{a : *,\, x : a \vdash (\lambda y : a.\, y) : a \to a}
\quad
\dfrac{
  \dfrac{
    \dfrac{}{\vdash * : \square}
  }{a : * \vdash a : *}
}{a : *,\, x : a \vdash x : a}
\over
a : *,\, x : a \vdash (\lambda y : a.\, y)x : a
$$

$$
\dfrac{
  \dfrac{
    \dfrac{}{x : a,\, y : a \vdash y : a}
  }{x : a \vdash (\lambda y : a.\, y) : a \to a}
  \quad
  \dfrac{}{x : a \vdash x : a}
}{x : a \vdash (\lambda y : a.\, y)x : a}
$$

example PTS type derivation (continued)

$$
\dfrac{
  \dfrac{
    \dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *}
    \qquad
    \dfrac{
      \dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *}
      \quad
      \dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *}
    }{a : *,\, y : a \vdash a : *}
  }{a : * \vdash (\Pi y : a.\, a) : *}
  \qquad
  \dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *}
}{a : *,\, x : a \vdash \underbrace{a \to a}_{\Pi y : a.\, a} : *}
$$

$$
\dfrac{}{\vdash * : \Box}
\qquad
\dfrac{\Gamma \vdash A : s}{\Gamma,\, x : A \vdash x : A}
\qquad
\dfrac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma,\, y : B \vdash M : A}
$$

$$
\dfrac{\Gamma \vdash A : * \quad \Gamma,\, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\, B : s}
$$

conclusion

computing the sort of a type

$$\begin{aligned}
\mathsf{type}_\Gamma(\lambda x : A.\, M) &= \Pi x : A.\, \mathsf{type}_{\Gamma, x:A}(M) \\
\mathsf{type}_\Gamma(\Pi x : A.\, B) &= \mathsf{type}_{\Gamma, x:A}(B) \\
\mathsf{type}_\Gamma(A \to B) &= \mathsf{type}_\Gamma(B)
\end{aligned}$$

$$\begin{aligned}
\mathsf{type}_\Gamma(*) &= \Box \\
\mathsf{type}_\Gamma(x) &= \Gamma(x)
\end{aligned}$$

$$\mathsf{type}_\Gamma(F M) = \ldots$$

summary

- predicate logic
- dependent types
  - vectors
  - Curry-Howard
- λP
  - lambda cube
  - PTSs
- detours

contents