

predicate logic & dependent types

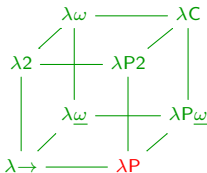
Freek Wiedijk

Type Theory & Rocq

2025-2026

Radboud University Nijmegen

September 24, 2025



introduction

function types become dependent

for predicate logic one needs to generalize

$$(\lambda x : A. M) : A \rightarrow B$$

function types

to

can contain x



$$(\lambda x : A. M) : \Pi x : A. B$$

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



next week

program extraction



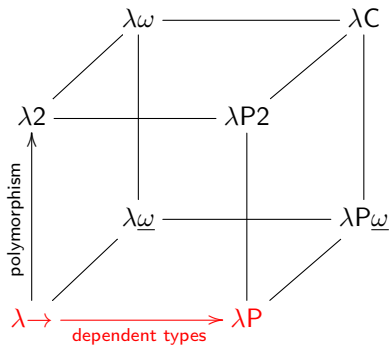
last week (briefly)

two weeks ago 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



recap: propositional logic

$$A, B ::= a \mid A \rightarrow B \mid A \wedge B \mid A \vee B \mid \neg A \mid \top \mid \perp$$

$$\begin{array}{ll} I \rightarrow & E \rightarrow \\ I \wedge & El \wedge \quad Er \wedge \\ Il \vee \quad Ir \vee & E \vee \\ I \neg & E \neg \\ I \top & E \perp \end{array}$$

recap: simply typed lambda calculus

$$\begin{aligned} A, B &::= a \mid A \rightarrow B \\ M, N &::= x \mid MN \mid \lambda x : A. M \end{aligned}$$

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

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

$$\frac{\Gamma \vdash F : A \rightarrow B \quad \Gamma \vdash M : A}{\Gamma \vdash FM : B}$$

predicate logic

three predicate logics

- ▶ minimal predicate logic

$$\rightarrow \forall$$

- ▶ constructive predicate logic

$$\rightarrow \wedge \vee \neg \top \perp \forall \exists$$

- ▶ classical predicate logic

$$\begin{aligned} & A \vee \neg A \\ & \neg\neg A \rightarrow A \end{aligned}$$

propositional versus predicate logic: syntax

$$A, B ::= a \mid A \rightarrow B \mid A \wedge B \mid A \vee B \mid \neg A \mid \top \mid \perp$$

$$M, N ::= x \mid f(\vec{M})$$

$$\vec{M} ::= \cdot \mid \vec{M}, N$$

$$A, B ::= p(\vec{M}) \mid A \rightarrow B \mid A \wedge B \mid A \vee B \mid \neg A \mid \top \mid \perp \mid \\ \forall x. A \mid \exists x. A$$

$\forall$ and $\exists$ bind weakly

minimal propositional versus minimal predicate logic: rules

$$\frac{\begin{array}{c} [A^H] \\ \vdots \\ B \end{array}}{A \rightarrow B} I[H] \rightarrow$$

$$\frac{\begin{array}{c} \vdots \\ A \rightarrow B \end{array} \quad \begin{array}{c} \vdots \\ A \end{array}}{B} E \rightarrow$$

$$\frac{\begin{array}{c} \vdots \\ A \end{array}}{\forall x. A} I\forall$$

$$\frac{\begin{array}{c} \vdots \\ \forall x. A \end{array}}{A[x := M]} E\forall$$

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

same four rules with explicit assumptions

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} I \rightarrow$$

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} E \rightarrow$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x. A} I \forall$$

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

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

rules for the existential quantifier

$$\frac{\vdots \quad A[x := M]}{\exists x. A} I\exists \qquad \frac{\vdots \quad \exists x. A \quad \vdots \quad \forall x. A \rightarrow C}{C} E\exists$$

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

example proof in minimal logic

$$\frac{\frac{\frac{[\forall y. p(y)^H]}{p(x)} E\forall}{(\forall y. p(y)) \rightarrow p(x)} I[H] \rightarrow}{\forall x. ((\forall y. p(y)) \rightarrow p(x))} I\forall$$

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

$$A \leftrightarrow B := (A \rightarrow B) \wedge (B \rightarrow A)$$

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

not constructively valid

$$\begin{aligned} &(\exists x. p(x)) \rightarrow \neg(\forall x. \neg p(x)) \\ &\neg(\forall x. \neg p(x)) \rightarrow \textcolor{blue}{\neg}(\exists x. p(x)) \end{aligned}$$

example from left to right

$$\frac{\frac{\frac{[\exists x. p(x)^{H_0}] \quad [\forall x. p(x) \rightarrow \perp^{H_1}]}{E\exists} \quad \perp}{\neg(\forall x. \neg p(x))} I[H_1] \neg}{(\exists x. p(x)) \rightarrow \neg(\forall x. \neg p(x))} I[H_0] \rightarrow$$

variable condition check:

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

example from right to left

$$\begin{array}{c}
 \frac{[\neg(\exists x. p(x))^{H_1}] \quad \frac{[p(x)^{H_2}]}{\exists x. p(x)} I\exists}{E\rightarrow} \\
 \frac{\perp}{\neg p(x)} I[H_2]\neg \\
 \frac{[\neg(\forall x. \neg p(x))^{H_0}] \quad \frac{\forall x. \neg p(x)}{I\forall}}{E\neg} \\
 \frac{\perp}{\neg\neg(\exists x. p(x))} I[H_1]\neg \\
 \frac{\neg\neg(\exists x. p(x)) \quad I[H_0]\rightarrow}{\neg(\forall x. \neg p(x)) \rightarrow \neg\neg(\exists x. p(x))}
 \end{array}$$

variable condition check:

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

dependent type theory

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
 - ▶ $\dots$

how are types introduced?

- ▶ free type variables

STT = simple type theory

- ▶ in the context = 'axiomatic'

PTSs = pure type systems: $\lambda \rightarrow$ λP $\lambda 2$ λC

$$\Gamma \vdash M : A$$

$$\text{nat} : *, O : \text{nat}, S : \text{nat} \rightarrow \text{nat} \vdash S (S (S O)) : \text{nat}$$

Rocq: Parameter

- ▶ definitions $\longrightarrow$ next week

CIC = Calculus of Inductive Constructions

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

Rocq: Definition
Inductive

star and box

the **sorts** of the lambda cube:

***** = the type of types

□ = the type of *

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

object : type

type : kind

kind : □

object : type : **kind** : □

S : nat → nat : ***** : □

example contexts

natural numbers:

$\text{nat} : *$

$0 : \text{nat}$

$S : \text{nat} \rightarrow \text{nat}$

$\text{add} : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}$

Booleans:

$\text{bool} : *$

$\text{true} : \text{bool}$

$\text{false} : \text{bool}$

lists

lists of Booleans:

```
list : *  
nil : list  
cons : bool → list → list  
hd : list → bool  
tl : list → list  
append : list → list → list  
reverse : list → list
```

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'

$$\text{trueslist } 0 = \text{nil}$$
$$\text{trueslist } (S\ 0) = \text{cons true nil}$$
$$\text{trueslist } (S\ (S\ 0)) = \text{cons true (cons true nil)}$$
$$\text{trueslist } (S\ (S\ (S\ 0))) = \text{cons true (cons true (cons true nil))}$$
$$\dots$$
$$\text{trueslist} : \text{nat} \rightarrow \text{list}$$

vectors and truesvec

vectors of Booleans:

$\text{vec} : \text{nat} \rightarrow *$	
$\text{vec } 0 : *$	vectors of length 0
$\text{vec } (S\ 0) : *$	vectors of length 1
$\text{vec } (S\ (S\ 0)) : *$	vectors of length 2
$\dots$	

$\text{trueslist} : \text{nat} \rightarrow \text{list}$

$\text{truesvec} : \text{nat} \rightarrow \text{vec } n$

$\text{truesvec} : \prod n : \text{nat}. \text{vec } n$

vnil and vcons

`vnil : vec 0`

`vcons : nat → bool → vec n → vec (S n)`

`vcons :  $\prod n : \text{nat.}$  bool → vec n → vec (S n)`

`vcons` has three arguments!

`truevec (S (S 0)) : vec (S (S 0))`

`truevec (S (S 0)) = vcons (S 0) true (vcons 0 true vnil)`

types when applying vcons

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

$\text{truesvec} : \prod n : \text{nat}. \text{vec } n$

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

$\text{vcons} : \prod n : \text{nat}. \text{bool} \rightarrow \text{vec } n \rightarrow \text{vec } (S\ n)$

$O : \text{nat}$

$\text{vcons } O : \text{bool} \rightarrow \text{vec } O \rightarrow \text{vec } (S\ O)$

$\text{true} : \text{bool}$

$\text{vcons } O\ \text{true} : \text{vec } O \rightarrow \text{vec } (S\ O)$

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

$\text{vcons } O\ \text{true}\ \text{vnil} : \text{vec } (S\ O)$

BHK-interpretation

proof of $A \rightarrow B$	function from proofs of A to proofs of B
proof of $\neg A$	function from proofs of A to proofs of $\perp$
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 or 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

Curry-Howard correspondence

type theory	Rocq	minimal logic
$A \rightarrow B$	$A \rightarrow B$	$A \rightarrow B$
$\Pi x : D. A$	<code>forall x : D, A</code>	$\forall x. A$

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

proof term	type	statement proved
H	$: A \rightarrow B$	$A \rightarrow B$
H'	$: A$	A
HH'	$: B$	B
H	$: \Pi x : D. Px$	$\forall x. P(x)$
HM	$: PM$	$P(M)$

Rocq 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
```

$$\frac{\frac{\frac{[\forall y. p(y)^H]}{p(x)} E\forall}{(\forall y. p(y)) \rightarrow p(x)} I[H] \rightarrow}{\forall x. (\forall y. p(y)) \rightarrow p(x)} I\forall$$

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

$$\forall x. (\forall y. p(y)) \rightarrow p(x)$$

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

the second example in Rocq, 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.

$$\frac{\frac{\frac{[\exists x. p(x)^{H_0}] \quad [\forall x. p(x) \rightarrow \perp^{H_1}]}{\perp} E\exists}{\neg(\forall x. \neg p(x))} I[H_1] \neg}{(\exists x. p(x)) \rightarrow \neg(\forall x. \neg p(x))} I[H_0] \rightarrow$$

the second example in Rocq, 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.

$$\frac{\frac{\frac{[\neg(\exists x. p(x))^{H_1}]}{\frac{\frac{\perp}{\neg p(x)} I[H_2] \neg} \forall x. \neg p(x) I\forall} \exists x. p(x) I\exists}{E\rightarrow} \frac{[\neg(\forall x. \neg p(x))^{H_0}]}{\neg\neg(\exists x. p(x))} E\neg}{\neg(\forall x. \neg p(x)) \rightarrow \neg\neg(\exists x. p(x))} I[H_0] \rightarrow$$

proof rules versus Rocq 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 proof₁ of the lemma everywhere

$$\frac{\frac{\frac{[A^H]}{\vdots_2} B}{A \rightarrow B} \textcolor{red}{I[H] \rightarrow} \quad \frac{\vdots_1}{A} \textcolor{red}{E \rightarrow}}{B} \longrightarrow_{\beta} \frac{\vdots_1}{A} \textcolor{red}{E \rightarrow} \quad \frac{\vdots_2}{B}$$

detours for predicate logic

detour for the universal quantifier

generalize the statement $A[x := M]$ to A with arbitrary x
elimination is specializing the proof₁ to M

$$\frac{\frac{\frac{\vdots_1}{A}}{\forall x. A} I\forall}{A[x := M]} E\forall \quad \longrightarrow_{\beta} \quad \frac{\vdots_1[x := M]}{A[x := M]}$$

$$(\lambda x : D. H_1)M \rightarrow_{\beta} H_1[x := M]$$

λP

STT versus λP syntax

STT:

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

λP :

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

$$\overline{\Gamma} \quad \overline{M} \quad \overline{A} \\ a : *, b : *, f : a \rightarrow b, x : a \vdash fx : b$$

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

→ no rule for box

variables have two rules

conversion rule

typing rules

STT

λP

$$\overline{\vdash * : \square}$$

axiom rule

start rule

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

product rule

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

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

abstraction rule

typing rules (continued)

STT

$$\frac{\Gamma \vdash F : A \rightarrow B \quad \Gamma \vdash M : A}{\Gamma \vdash FM : B}$$

λP

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

application rule

truesvec : $\Pi n : \text{nat}. \text{vec } n$
truesvec (S (S O)) : $\text{vec } (\text{S } (\text{S } \text{O}))$

typing rules (continued)

STT

(context Γ is a set)

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

λP

(context Γ is a list)

correct: $a : *, x : a \vdash x : a$

incorrect: $x : a, a : * \vdash x : a$

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$$

variable rule

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma, y : B \vdash M : A}$$

weakening rule

the final typing rule

$\text{vecappend} : \Pi n_1 : \text{nat}. \Pi n_2 : \text{nat}.$

$\text{vec } n_1 \rightarrow \text{vec } n_2 \rightarrow \text{vec } (\text{add } n_1 \ n_2)$

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

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

λP

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

conversion rule

A and A' are convertible

$=_{\beta}$ is defined on preterms

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

$$s ::= * \mid \square$$

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

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

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

the lambda cube

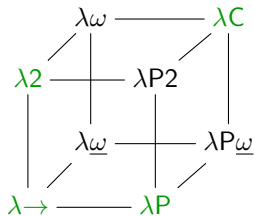
only the product rules differ:

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

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

$$\lambda 2 \frac{\Gamma \vdash A : s \quad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi x : A. B : *}$$

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



pure type systems

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

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

	$\mathcal{S}$ sorts	$\mathcal{A}$ axioms	$\mathcal{R}$ rules
$\lambda*$	$\{*\}$	$\{(*, *)\}$	$\{(*, *, *)\}$
$\lambda \rightarrow$	$\{*, \square\}$	$\{(*, \square)\}$	$\{(*, *, *)\}$
λP	$\{*, \square\}$	$\{(*, \square)\}$	$\{(*, *, *), (*, \square, \square)\}$
λC	$\{*, \square\}$	$\{(*, \square)\}$	$\{(*, *, *), (*, \square, \square),$ $(\square, *, *), (\square, \square, \square)\}$
$\lambda PRED$	$\{*_s, \square_s, *_p, \square_p\}$	$\{(*_s, \square_s), (*_p, \square_p)\}$	$\{(*_s, *_s, *_s), (*_s, \square_s, \square_s),$ $(*_p, *_p, *_p), (*_s, *_p, *_p)\}$

example PTS type derivation

$$\begin{array}{c}
 \frac{\overline{\vdash * : \square}}{a : * \vdash a : *} \quad \frac{\overline{\vdash * : \square}}{a : * \vdash a : *} \\
 \hline
 a : *, x : a \vdash a : * \\
 \hline
 a : *, x : a, y : a \vdash y : a \quad a : *, x : a \vdash a \rightarrow a : * \quad \frac{\overline{\vdash * : \square}}{a : * \vdash a : *} \\
 \hline
 a : *, x : a \vdash (\lambda y : a. y) : a \rightarrow a \quad a : *, x : a \vdash x : a \\
 \hline
 a : *, x : a \vdash (\lambda y : a. y)x : a
 \end{array}$$

(next slide)

$$\begin{array}{c}
 \frac{\overline{x : a, y : a \vdash y : a}}{x : a \vdash (\lambda y : a. y) : a \rightarrow a} \quad \frac{\overline{x : a \vdash x : a}}{x : a \vdash (\lambda y : a. y)x : a} \\
 \hline
 x : a \vdash (\lambda y : a. y)x : a
 \end{array}$$

example PTS type derivation (continued)

$$\begin{array}{c}
 \frac{}{\overline{\vdash * : \Box}} \quad \frac{\overline{\vdash * : \Box}}{a : * \vdash a : *} \quad \frac{\overline{\vdash * : \Box}}{a : * \vdash a : *} \\
 \hline
 \frac{a : * \vdash a : * \quad a : *, y : a \vdash a : *}{a : * \vdash (\Pi y : a. a) : *} \quad \frac{\overline{\vdash * : \Box}}{a : * \vdash a : *} \\
 \hline
 a : *, x : a \vdash \underbrace{a \rightarrow a}_{\Pi y : a. a} : *
 \end{array}$$

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

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

conclusion

computing the sort of a type

$$\text{type}_{\Gamma}(\lambda x : A. M) = \Pi x : A. \text{type}_{\Gamma, x:A}(M)$$

$$\text{type}_{\Gamma}(\Pi x : A. B) = \text{type}_{\Gamma, x:A}(B)$$

$$\text{type}_{\Gamma}(A \rightarrow B) = \text{type}_{\Gamma}(B)$$

$$\text{type}_{\Gamma}(*) = \square$$

$$\text{type}_{\Gamma}(x) = \Gamma(x)$$

$$\text{type}_{\Gamma}(FM) = \dots$$

summary

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

table of contents

contents

introduction

predicate logic

dependent type theory

Rocq

detours

λP

conclusion

table of contents