

Proving with Computer Assistance

Lecture **Dependent Type Theory  $\lambda P$**

Herman Geuvers

For the slides, thanks to: Freek Wiedijk

where we are in the course

## recap

propositional logic  $\leftrightarrow$  simple type theory  
 $\lambda \rightarrow$

predicate logic  $\leftrightarrow$  type theory with dependent types  
 $\lambda P$

2nd order propositional logic  $\leftrightarrow$  polymorphic type theory  
 $\lambda 2$

higher order predicate logic  $\leftrightarrow$  calculus of constructions  
 $\lambda C$

the main difference between  $\lambda \rightarrow$  and  $\lambda P$

$$A \rightarrow B$$

'type of functions from  $A$  to  $B$ '

$$\prod x : A. B$$

'type of functions from  $A$  to  $B$ '

dependent product  
dependent function type

type of function value  $B$  now can depend on function argument  $x$   
arrow type becomes a special case

# syntax

$\lambda P$

- **two sorts**  
 $*$ ,  $\square$
- **variables**  
 $x, y, z, \dots$
- **function application**  
 $MN$
- **function abstraction**  
 $\lambda x : A. M$
- **dependent product**  
 $\prod x : A. M$

## Rocq syntax versus $\lambda P$ syntax

*	$\leftrightarrow$	Set
*	$\leftrightarrow$	Prop
$\square$	$\leftrightarrow$	Type
x	$\leftrightarrow$	x
M N	$\leftrightarrow$	M N
$\lambda x : A. M$	$\leftrightarrow$	fun x:A => M
$\prod x : A. M$	$\leftrightarrow$	forall x:A, M

$\lambda P$  does not make the distinction between Set and Prop

## pseudo-terms versus terms

any expression according to the  $\lambda P$  grammar is called a **pseudo-term**

$$\begin{aligned} & (\square *) \\ & \lambda n : \text{nat}. \lambda x : n. x \\ & (\lambda x : \text{nat}. x x) (\lambda x : \text{nat}. x x) \end{aligned}$$

if also all types are okay, then the expression is called a **term**

$$\begin{aligned} & \square \\ & \lambda n : \text{nat}. \text{nat} \\ & (\lambda f : (\prod m : \text{nat}. \text{nat}). \lambda x : \text{nat}. f x) (\lambda n : \text{nat}. n) \\ & (\lambda f : \text{nat} \rightarrow \text{nat}. \lambda x : \text{nat}. f x) (\lambda n : \text{nat}. n) \end{aligned}$$

## contexts and judgments

a **judgment** has the form  $\Gamma \vdash M : N$

with  $\Gamma$  a context and  $M$  and  $N$  terms

a **context**  $\Gamma$  is a list of variable declarations

a variable **declaration** has the form  $x : M$

with  $x$  a variable name and  $M$  a term (usually a type)

$$A : *, P : (\Pi x : A. *), a : A \vdash (\Pi w : P a. *) : \square$$

$$A : *, P : A \rightarrow *, a : A \vdash (P a) \rightarrow * : \square$$

## the seven rules of $\lambda P$

- one rule for each kind of term
  - ▶ axiom rule (for the **sorts**)
  - ▶ **variable** rule
  - ▶ **product** rule
  - ▶ **abstraction** rule
  - ▶ **application** rule
- two more rules
  - ▶ weakening rule (for the contexts)
  - ▶ **conversion** rule

## rule 1: axiom

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

gives the type of the **sort**  $*$   
the only rule with no premises!

## rules 2 and 3: variable and weakening

in these rules  $s$  is either  $*$  or  $\square$

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

gives the type of the **variable**  $x$

if the variable is not the last in the context we need the **weakening** rule

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

## rule 4: product

$$\frac{\Gamma \vdash A : * \quad \Gamma \vdash B : s}{\Gamma \vdash A \rightarrow B : s}$$

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

gives the type of a **dependent product**

## rule 5: abstraction

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

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

gives the type of a **function abstraction**

## rule 6: application

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\frac{\Gamma \vdash M : \Pi x : A. B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

gives the type of a **function application**

## rule 6: application

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

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

gives the type of a **function application**

## rule 7: conversion

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'}$$

with  $B =_{\beta} B'$

is needed to make everything work

# reduction and convertibility

- step

$$\dots ((\lambda x : A. M) N) \dots \rightarrow_{\beta} \dots (M[x := N]) \dots$$

- reduction  $\rightarrow_{\beta}$   
zero or more steps
- convertibility  $=_{\beta}$   
smallest equivalence relation

axiom, application, abstraction, product

### cheat sheet

$$\frac{}{\vdash * : \square} \quad (\text{ax})$$

$$\frac{\Gamma \vdash M : \Pi x : A. B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]} \quad (\text{app})$$

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

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

## weakening, variable, conversion

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \quad (\text{weak})$$

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad (\text{var})$$

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \quad \text{if } B =_{\beta} B' \quad (\text{conv})$$

example 1

**examples**

$X : *, x : X \vdash x : X$

## example 2

$$X : * \vdash (X \rightarrow X) : *$$

### example 3

$A : *$ ,  $P : A \rightarrow *$ ,  $a : A \vdash (P a) \rightarrow * : \square$

# introduction rules versus abstraction rule

## Curry-Howard-de Bruijn for minimal predicate logic

$$\frac{\begin{array}{c} [A^x] \\ \vdots \\ B \end{array}}{A \rightarrow B} \quad I[\rightarrow] \quad \frac{\begin{array}{c} \vdots \\ B \end{array}}{\forall x. B} \quad I\forall$$

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

## elimination rules versus application rule

$$\frac{\begin{array}{c} \vdots \\ A \rightarrow B \end{array} \quad \begin{array}{c} \vdots \\ A \end{array}}{B} E_{\rightarrow} \qquad \frac{\begin{array}{c} \vdots \\ \forall x. B \end{array}}{B[x := M]} E_{\forall}$$

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

## example 4

### examples

$$\forall x. (P(x) \rightarrow (\forall y. P(y) \rightarrow A) \rightarrow A)$$

## example 5

$$(\forall x. P(x) \rightarrow Q(x)) \rightarrow (\forall x. P(x)) \rightarrow \forall y. Q(y)$$

Exercise! (See the exercise sheet, also for more exercises!)

## example 6

$$\forall x. (P(x) \rightarrow P(f(x))) \vdash \forall x. (P(x) \rightarrow P(f(f(x))))$$

## example 7

$$\begin{aligned} &\forall x. (P(x) \rightarrow R(x, f(x))), \\ &\forall x, y. (R(x, y) \rightarrow R(y, x)), \\ &\forall x, y. (R(x, y) \rightarrow R(f(y), x)) \vdash \forall x. (P(x) \rightarrow R(f(x), f(x))) \end{aligned}$$

Exercise! (See the exercise sheet, also for more exercises!)

# Properties of $\lambda P$

- ▶ **Uniqueness of types**

If  $\Gamma \vdash M : \sigma$  and  $\Gamma \vdash M : \tau$ , then  $\sigma =_{\beta} \tau$ .

- ▶ **Subject Reduction**

If  $\Gamma \vdash M : \sigma$  and  $M \rightarrow_{\beta} N$ , then  $\Gamma \vdash N : \sigma$ .

- ▶ **Strong Normalization**

If  $\Gamma \vdash M : \sigma$ , then all  $\beta$ -reductions from  $M$  terminate.

Proof of SN is by defining a reduction preserving map from  $\lambda P$  to  $\lambda \rightarrow$ .

# Decidability Questions

$\Gamma \vdash M : \sigma?$  TCP  
 $\Gamma \vdash M : ?$  TSP  
 $\Gamma \vdash ? : \sigma$  TIP

For  $\lambda P$ :

- ▶ TIP is **undecidable** (TIP is equivalent to **provability** in minimal predicate logic.)
- ▶ TCP/TSP: simultaneously with **Context checking**

# Type Checking

Define algorithms  $\text{Ok}(-)$  and  $\text{Type}_(-)$  simultaneously:

- ▶  $\text{Ok}(-)$  takes a **context** and returns 'true' or 'false'
- ▶  $\text{Type}_(-)$  takes a **context** and a **term** and returns a **term** or 'false'.

DEFINITION The **type synthesis algorithm**  $\text{Type}_(-)$  is **sound** if (for all  $\Gamma$  and  $M$ )

$$\text{Type}_\Gamma(M) = A \implies \Gamma \vdash M : A$$

The **type synthesis algorithm**  $\text{Type}_(-)$  is **complete** if (for all  $\Gamma$ ,  $M$  and  $A$ )

$$\Gamma \vdash M : A \implies \text{Type}_\Gamma(M) =_\beta A$$

- ▶ A proof assistant like Rocq is based on a type checking algorithm.
- ▶ The type checking algorithm is the **trusted kernel** of Rocq

$$\text{Ok}(\langle \rangle) = \text{'true'}$$

$$\text{Ok}(\Gamma, x:A) = \text{Type}_\Gamma(A) \in \{*, \square\},$$

$$\text{Type}_\Gamma(x) = \text{if } \text{Ok}(\Gamma) \text{ and } x:A \in \Gamma \text{ then } A \text{ else 'false'},$$

$$\text{Type}_\Gamma(*) = \text{if } \text{Ok}(\Gamma) \text{ then } \square \text{ else 'false'},$$

$$\begin{aligned} \text{Type}_\Gamma(MN) = & \text{if } \text{Type}_\Gamma(M) = C \text{ and } \text{Type}_\Gamma(N) = D \\ & \text{then} \quad \text{if } C \rightarrow_\beta \Pi x:A. B \text{ and } A =_\beta D \\ & \quad \text{then } B[x := N] \text{ else 'false'} \\ & \text{else} \quad \text{'false'}, \end{aligned}$$

$$\text{Type}_\Gamma(\lambda x:A.M) = \text{if } \text{Type}_{\Gamma,x:A}(M) = B$$

$$\quad \text{then} \quad \text{if } \text{Type}_\Gamma(\Pi x:A.B) \in \{*, \square\}$$

$$\quad \quad \text{then } \Pi x:A.B \text{ else 'false'}$$

$$\quad \text{else 'false'},$$

$$\text{Type}_\Gamma(\Pi x:A.B) = \text{if } \text{Type}_\Gamma(A) = * \text{ and } \text{Type}_{\Gamma,x:A}(B) = s$$

$$\quad \text{then } s \text{ else 'false'}$$

# Soundness and Completeness

Soundness

$$\text{Type}_\Gamma(M) = A \implies \Gamma \vdash M : A$$

Completeness

$$\Gamma \vdash M : A \implies \text{Type}_\Gamma(M) =_\beta A$$

As a consequence:

$$\text{Type}_\Gamma(M) = \text{'false'} \implies M \text{ is not typable in } \Gamma$$

NB 1. Completeness implies that `Type` terminates on **all well-typed terms**. We want that `Type` terminates on **all pseudo terms**.

NB 2. Completeness only makes sense if we have **uniqueness of types**

(Otherwise: let `TypeΓ(-)` generate a **set of possible types**)

# Termination

We want  $\text{Type}_\Gamma(-)$  to **terminate** on all inputs.

Interesting cases:  $\lambda$ -abstraction and application:

$$\begin{aligned} \text{Type}_\Gamma(\lambda x:A.M) = & \text{if } \text{Type}_{\Gamma,x:A}(M) = B \\ & \text{then} \quad \text{if } \text{Type}_\Gamma(\Pi x:A.B) \in \{*, \square\} \\ & \quad \text{then } \Pi x:A.B \text{ else 'false'} \\ & \text{else 'false'}, \end{aligned}$$

! Recursive call is not on a **smaller** term!

Replace the side condition

$$\text{if } \text{Type}_\Gamma(\Pi x:A.B) \in \{*, \square\}$$

by

$$\text{if } \text{Type}_\Gamma(A) \in \{*\}$$

# Termination

We want  $\text{Type}_\Gamma(-)$  to **terminate** on all inputs.

Interesting cases:  $\lambda$ -abstraction and application:

$$\begin{aligned} \text{Type}_\Gamma(MN) &= \text{if } \text{Type}_\Gamma(M) = C \text{ and } \text{Type}_\Gamma(N) = D \\ &\quad \text{then if } C \rightarrow_\beta \Pi x:A.B \text{ and } A =_\beta D \\ &\quad \quad \text{then } B[x := N] \text{ else 'false'} \\ &\quad \text{else 'false'}, \end{aligned}$$

! Need to decide  $\beta$ -reduction and  $\beta$ -equality!

For this case, **termination** follows from:

- ▶ Soundness of  $\text{Type}$  and
- ▶ **Decidability of equality** on **well-typed** terms.

This decidability of equality follows from **SN** (strong normalization) and **CR** (Church-Rosser property) – to be discussed in later lectures.

## Rocq dependent type theory and predicate logic

- ▶ First order language: domain  $D$ , with variables  $x, y, z : D$  and possibly functions over  $D$ , e.g.  $f : D \rightarrow D$ ,  $g : D \rightarrow D \rightarrow D$ .
- ▶ NB There are two “kinds” of variables: the first order variables (ranging over the domain  $D$ ) and the “proof variables” (used as [local] assumptions of formulas).
- ▶ In Rocq:  $D : \text{Set}$  and  $\varphi : \text{Prop}$ . You can use any name for variables, but often

$H : \varphi$      for assumptions: Suppose  $H$  is a proof of  $\varphi$

$x : D$      for variable declarations: Let  $x$  be an element of  $D$