Proving with Computer Assistance

Lecture Dependent Type Theory λP

Herman Geuvers
For the slides, thanks to: Freek Wiedijk

where we are in the course

recap

propositional logic	$\leftrightarrow$	simple type theory $\lambda{\to}$
predicate logic	$\leftrightarrow$	type theory with dependent types λP
2nd order propositional logic	$\leftrightarrow$	polymorphic type theory $\lambda 2$
higher order predicate logic	$\leftrightarrow$	calculus of constructions λC

the main difference between $\lambda\!\to$ and λP

$$A \to B$$

'type of functions from A to B'

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

'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

λP

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

Coq syntax versus λP syntax

$*$	$\leftrightarrow$	`Set`
$*$	$\leftrightarrow$	`Prop`
$\Box$	$\leftrightarrow$	`Type`
x	$\leftrightarrow$	`x`
$M\,N$	$\leftrightarrow$	`M N`
$\lambda x : A.\, M$	$\leftrightarrow$	`fun x:A => M`
$\Pi x : A.\, M$	$\leftrightarrow$	`forall x:A, M`

λP does not make the distinction between `Set` and `Prop`

pseudo-terms versus terms

any expression according to the λP grammar is called a pseudo-term

$$(\Box\, *)$$
$$\lambda n : \mathsf{nat}.\, \lambda x : n.\, x$$
$$(\lambda x : \mathsf{nat}.\, x\, x)\, (\lambda x : \mathsf{nat}.\, x\, x)$$

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

$$\Box$$
$$\lambda n : \mathsf{nat}.\, \mathsf{nat}$$
$$(\lambda f : (\Pi m : \mathsf{nat}.\, \mathsf{nat}).\, \lambda x : \mathsf{nat}.\, f\, x)\, (\lambda n : \mathsf{nat}.\, n)$$
$$(\lambda f : \mathsf{nat} \to \mathsf{nat}.\, \lambda x : \mathsf{nat}.\, f\, x)\, (\lambda n : \mathsf{nat}.\, n)$$

contexts and judgments

a judgment has the form $\Gamma \vdash M : N$
with Γ a context and M and N terms

a context Γ 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)

$$A : *,\ P : (\Pi x : A.\, *),\ a : A \quad \vdash \quad (\Pi w : P\, a.\, *) : \Box$$
$$A : *,\ P : A \to *,\ a : A \quad \vdash \quad (P\, a) \to * : \Box$$

the seven rules of λP

- 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

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

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 $\Box$

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

gives the type of the variable x

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

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

rule 4: product

$$\frac{\Gamma \vdash A : * \qquad \Gamma \vdash B : s}{\Gamma \vdash A \to B : s}$$

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

gives the type of a dependent product

rule 5: abstraction

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

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

gives the type of a function abstraction

rule 6: application

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

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

gives the type of a function application

rule 6: application

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

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

gives the type of a function application

rule 7: conversion

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \qquad \text{with } B =_\beta B'$$

is needed to make everything work

reduction and convertibility

- step

$$\ldots((\lambda x : A.\, M)N)\ldots \;\to_\beta\; \ldots(M[x := N])\ldots$$

- reduction $\twoheadrightarrow_\beta$
  zero or more steps

- convertibility $=_\beta$
  smallest equivalence relation

axiom, application, abstraction, product

cheat sheet

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

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

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

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

weakening, variable, conversion

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

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

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

example 1

examples

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

example 2

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

example 3

$$A : *,\ P : A \to *,\ a : A \vdash (P\, a) \to * : \Box$$

introduction rules versus abstraction rule

Curry-Howard-de Bruijn for minimal predicate logic

$$\dfrac{\begin{matrix}[A^x] \\ \vdots \\ B\end{matrix}}{A \to B}\; I[x]{\to} \qquad\qquad \dfrac{\begin{matrix}\vdots \\ B\end{matrix}}{\forall x.\, B}\; I\forall$$

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

elimination rules versus application rule

$$\dfrac{\begin{matrix}\vdots & & \vdots \\ A \to B & & A\end{matrix}}{B}\ E{\to} \qquad \dfrac{\begin{matrix}\vdots \\ \forall x.\, B\end{matrix}}{B[x := N]}\ E\forall$$

$$\dfrac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

example 4

examples

$$\forall x.\,(P(x) \to (\forall y.\,P(y) \to A) \to A)$$

example 5

$$(\forall x.\, P(x) \to Q(x)) \to (\forall x.\, P(x)) \to \forall y.\, Q(y)$$

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

example 6

$$\forall x. (P(x) \to P(f(x))) \quad \vdash \quad \forall x. (P(x) \to P(f(f(x))))$$

example 7

$$\begin{array}{l} \forall x.\,(P(x) \to R(x, f(x))), \\ \forall x, y.\,(R(x, y) \to R(y, x)), \\ \forall x, y.\,(R(x, y) \to R(f(y), x)) \quad \vdash \quad \forall x.\,(P(x) \to R(f(x), f(x))) \end{array}$$

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

Properties of λP

- 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 $M \longrightarrow_\beta N$, then $\Gamma \vdash N : \sigma$.
- Strong Normalization
  If $\Gamma \vdash M : \sigma$, then all β-reductions from M terminate.

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

Decidability Questions

$$\begin{array}{ll} \Gamma \vdash M : \sigma? & \text{TCP} \\ \Gamma \vdash M : ? & \text{TSP} \\ \Gamma \vdash ? : \sigma & \text{TIP} \end{array}$$

For λP:

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

Type Checking

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

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

The type synthesis algorithm $\mathrm{Type}_{\_}(-)$ is sound if (for all Γ and M)

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

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

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

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

$$
\begin{array}{rcl}
\mathrm{Ok}(<>) & = & \text{'true'} \\
\mathrm{Ok}(\Gamma, x{:}A) & = & \mathrm{Type}_\Gamma(A) \in \{*, \Box\}, \\
\mathrm{Type}_\Gamma(x) & = & \text{if } \mathrm{Ok}(\Gamma) \text{ and } x{:}A \in \Gamma \text{ then } A \text{ else 'false'}, \\
\mathrm{Type}_\Gamma(*) & = & \text{if } \mathrm{Ok}(\Gamma) \text{then } \Box \text{ else 'false'}, \\
\mathrm{Type}_\Gamma(MN) & = & \text{if } \mathrm{Type}_\Gamma(M) = C \text{ and } \mathrm{Type}_\Gamma(N) = D \\
 & & \quad \text{then} \quad \text{if } C \twoheadrightarrow_\beta \Pi x{:}A.B \text{ and } A =_\beta D \\
 & & \quad \qquad\qquad \text{then } B[x := N] \text{ else 'false'} \\
 & & \quad \text{else} \quad\; \text{'false'},
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{Type}_{\Gamma}(\lambda x{:}A.M) & = & \text{if } \mathrm{Type}_{\Gamma,x:A}(M) = B \\
& & \text{then} \quad \begin{array}{l} \text{if } \mathrm{Type}_{\Gamma}(\Pi x{:}A.B) \in \{*, \Box\} \\ \text{then } \Pi x{:}A.B \text{ else `false'} \end{array} \\
& & \text{else `false'}, \\
\mathrm{Type}_{\Gamma}(\Pi x{:}A.B) & = & \text{if } \mathrm{Type}_{\Gamma}(A) = * \text{ and } \mathrm{Type}_{\Gamma,x:A}(B) = s \\
& & \text{then } s \text{ else `false'}
\end{array}
$$

Soundness and Completeness

Soundness

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

Completeness

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

As a consequence:

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

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 $\mathrm{Type}_{\_}(-)$ generate a set of possible types)

Termination

We want $\mathrm{Type}_{\_}(-)$ to terminate on all inputs.
Interesting cases: λ-abstraction and application:

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

! Recursive call is not on a smaller term!
Replace the side condition

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

by

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

Termination

We want $\mathrm{Type}_{\_}(-)$ to terminate on all inputs.
Interesting cases: λ-abstraction and application:

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

! Need to decide β-reduction and β-equality!
For this case, termination follows from:

- Soundness of 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.

Coq dependent type theory and predicate logic

- First order language: domain D, with variables $x, y, z : D$ and possibly functions over D, e.g. $f : D \to D$, $g : D \to D \to D$.
- 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 Coq: $D : \texttt{Set}$ and $\varphi : \texttt{Prop}$. You can use any name for variables, but often

$H : \varphi$	for assumptions: Suppose H is a proof of φ
$x : D$	for variable declarations: Let x be an element of D