Lambda-Calculus and Type Theory

ISR 2024
Obergurgl, Austria

Herman Geuvers & Niels van der Weide

Radboud University Nijmegen, The Netherlands

Lecture 3

Dependent Types

The main difference between $\lambda\!\!\rightarrow$ and λP

$$A \rightarrow B$$

'type of functions from A to B'

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

'type of functions from A to B'

dependent product
dependent function type

The type of function values B now can depend on function argument x

The 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

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

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

elimination rules versus application rule

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

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

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

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

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

Example: Deriving irreflexivity from anti-symmetry

Assume a special type $\perp$.

$$\begin{aligned} \text{AntiSym}\, R &:= \forall x, y. R(x, y) \to R(y, x) \to \perp \\ \text{Irrefl}\, R &:= \forall x. R(x, x) \to \perp \end{aligned}$$

Derivation in predicate logic:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall x, y. R(x, y) \to R(y, x) \to \perp}{\forall y. R(x, y) \to R(y, x) \to \perp}}{R(x, x) \to R(x, x) \to \perp} \qquad [R(x, x)]}{R(x, x) \to \perp} \qquad [R(x, x)]}{\dfrac{\perp}{\dfrac{R(x, x) \to \perp}{\forall x. R(x, x) \to \perp}}}$$

Derivation in type theory, with terms

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{H : \Pi x, y{:}D. R\,x\,y \to R\,y\,x \to \bot}{H\,x : \Pi y{:}D. R\,x\,y \to R\,y\,x \to \bot}}{H\,x\,x : R\,x\,x \to R\,x\,x \to \bot} \qquad [H' : R\,x\,x]}{H\,x\,x\,H' : R\,x\,x \to \bot} \qquad [H' : R\,x\,x]}{H\,x\,x\,H'\,H' : \bot}}{\dfrac{\lambda H'{:}(R\,x\,x).H\,x\,x\,H'\,H' : R\,x\,x \to \bot}{\lambda x{:}D.\lambda H'{:}(R\,x\,x).H\,x\,x\,H'\,H' : \Pi x{:}D.R\,x\,x \to \bot}}
$$

From Automath to Twelf and Dedukti

Logical Framework

- **Automath**
  1968, de Bruijn
  start of proof checking of mathematics
- **LF**
  1987, Harper & Honsell & Plotkin
  framework for defining logics
  the type of theory of LF is precisely λP
- **Twelf**
  1998, Pfenning & Schürmann
  current implementation of LF
- **Dedukti**
  2016, Blanqui & Dowek & many others
  interpreting and exchanging formal systems via λP-modulo rewriting

Logics in a logical framework

each logic has a λP context that contains syntax and rules of the logic

Example: minimal propositional logic as a λP context

$$
\begin{gathered}
P : * \,, \\
\Rightarrow : P \to P \to P \,, \\
T : P \to * \,, \\
I : \Pi p : P.\, \Pi q : P.\, (Tp \to Tq) \to T(p \Rightarrow q)\,, \\
E : \Pi p : P.\, \Pi q : P.\, T(p \Rightarrow q) \to Tp \to Tq \\
\vdash \\
\ldots
\end{gathered}
$$

Idea: let the LF deal with variable binding and substitution, define

the logic via a declaration of constants in a context (signature).

Dedukti

Blanqui & Dowek & many others

Make proof assistants use each others results

Approach:

- Define contexts for the logics of the various proof assistants in λP-modulo
- Import a library of proof assistant (A) in Dedukti
- Translate this library in the format of proof assistant (B) in Dedukti and export.

see:

https://deducteam.github.io/

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