the pure type system called λP

logical verification

week 9

2004 11 10

where we are in the course

propositional logic	$\leftrightarrow$	simple type theory $\lambda{\rightarrow}$
predicate logic	$\leftrightarrow$	type theory with dependent types λP
2nd order propositional logic	$\leftrightarrow$	polymorphic type theory $\lambda 2$

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

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

λP

syntax

- **two sorts**

  $*$, $\Box$

- **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 type declarations

a type declaration has the form $x : M$
with x a variable name and M a term

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

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

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

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

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

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

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

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

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

- reduction $\twoheadrightarrow_{\beta}$

  zero or more steps

- convertibility $=_{\beta}$

  smallest equivalence relation

cheat sheet

axiom, application, abstraction, product

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

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

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

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

weakening, variable, conversion

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

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

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

example 1

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

example 2

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

example 3

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

Curry-Howard-de Bruijn for minimal predicate logic

introduction rules versus abstraction rule

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

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

elimination rules versus application rule

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

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

example 4

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

example 5

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

from automath to twelf

- **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

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

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

logosphere

Schürmann, Pfenning, Kohlhase, Shankar, Owre

making proof assistants talk to each other

approach:

- define contexts for the logics of the various proof assistants in twelf
- import the libraries of the proof assistants in twelf

see:

`http://www.logosphere.org/`

pure type systems

Berardi & Terlouw:

framework for defining and studying typed λ-calculi

PTS $=$ pure type system

the PTS rules are exactly* the λP rules as presented here

(* except for **one** symbol, in the product rule)

variations on the product rule

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

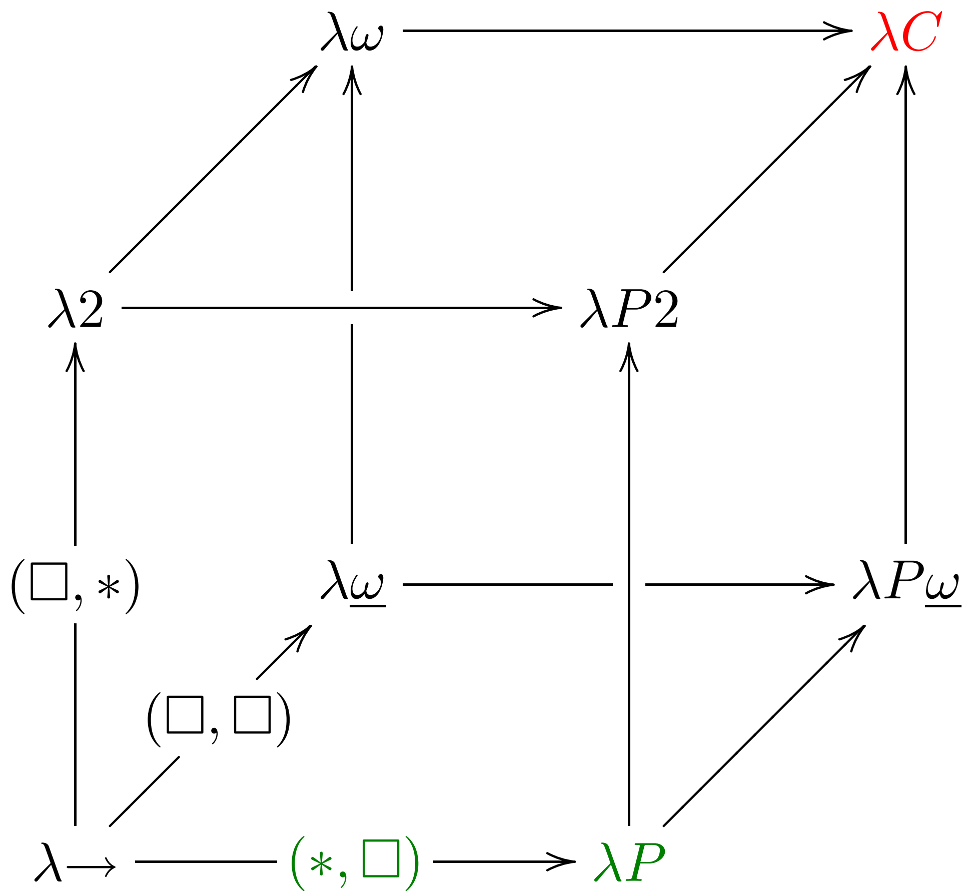
λP $\quad s_1 = *,\ s_2 \in \{*, \Box\}$

$\quad (s_1, s_2) \in \{(*, *), (*, \Box)\}$

$\lambda\!\rightarrow$ $\quad (s_1, s_2) \in \{(*, *)\}$

λC $\quad (s_1, s_2) \in \{(*, *), (*, \Box), (\Box, *), (\Box, \Box)\}$

the Barendregt cube



two schools of type theory