λP

Henk Barendregt and Freek Wiedijk
assisted by Andrew Polonsky

Radboud University Nijmegen

March 12, 2012

$\lambda\!\rightarrow$

$$\frac{}{\Gamma \vdash x : A} \quad x : A \in \Gamma$$

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

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

λP

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

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

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

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

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

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

dependent types

dependent types

$=$ types that are parametrized

by *objects*

not by types $\rightsquigarrow$ polymorphism

- programming languages
  - Agda (Sweden)
  - Coq (France)
  - Epigram (UK)
- proof assistants
  - Coq (France)
  - Mizar (Poland)
  - *not* HOL (UK)
  - *not* Isabelle (UK/Germany)

in mathematics

- non-dependent types:
  - complex number
  - ordinal number
  - group
  - field
  - ...
- *dependent* types:
  - vector space over K
  - n-dimensional vector space
  - field extension of K
  - well-ordering of X
  - ...

in computer science

- non-dependent types:
  - integer `int`
  - floating point number
  - algebraic datatypes
  - ...
- *dependent* types:
  - array `int[n]`
  - bitfield of a given length

```
printf("%d\n", i);
```

`printf "%d\n"` : ‘arguments type for `"%d\n"`’ $\to \ldots$

$\downarrow$

`int`

program correctness

typing: ensure that the program makes sense

dependent types: more expressive

evolution of programming languages:

- imperative/object-oriented programming
  Fortran, Algol, C, C++, Java
- functional programming
  Lisp, ML, Haskell
- dependently typed functional programming
  Epigram, Agda, Coq

Curry-Howard for predicate logic

BHK interpretation:

proof of $A \to B$: *function* that maps proofs of A to proofs of B

proof of $\forall x \in D.\, P[x]$: *function* that maps elements of D to $\underbrace{\text{proofs of } P[x]}_{\color{red}\text{dependent type!}}$ proofs of $P[x]$

type of proofs of $P[x]$ depends on parameter x

dependent functions

dependent lists

$$\text{vec } n \quad = \quad \text{type of } \textit{vectors} \text{ of length } n$$

of natural numbers

$$\text{zeroes } n \quad = \quad \text{the vector of zeroes of length } n$$

$$\begin{aligned}
\text{zeroes } 0 &= \langle\rangle \\
\text{zeroes } 1 &= \langle 0 \rangle \\
\text{zeroes } 2 &= \langle 0, 0 \rangle \\
\text{zeroes } 3 &= \langle 0, 0, 0 \rangle \\
\ldots &
\end{aligned}$$

type of zeroes?

$$\text{zeroes} \quad : \quad \text{nat} \longrightarrow \text{vec}\ \underset{?}{\underset{\uparrow}{n}}$$

n in output type depends on input argument:

$$\text{zeroes} \quad : \quad \Pi n : \text{nat}.\, \text{vec}\ n$$

dependent product

$=$ *dependent* function type

$$\boxed{\Pi x : A.\, B}$$

x can occur here

non-dependent function type now becomes abbreviation:

$$A \to B \quad := \quad \Pi x : A.\, B$$

if x does not occur in B

λP

λP

extend $\lambda\!\rightarrow$ with dependent types

- syntax
  - $\left.\begin{array}{l}\blacksquare\ \text{\sout{terms}} \\ \blacksquare\ \text{\sout{types}}\end{array}\right\}$ become unified!
  - contexts
  - judgments

- *rules*

  7 rules instead of 3

terms

grammar of pseudo-terms

both object and type variables!

$$M \quad ::= \quad x \mid (MM) \mid (\lambda x : M.\, M) \mid (\Pi x : M.\, M) \mid * \mid \Box$$

$M, N, A, B, \ldots$

sorts:

$$s \quad ::= \quad * \mid \Box$$

$s, s', s_1, s_2, \ldots$

never left of ':'
only all by itself right of ':'

rules

- $\lambda\to$

  3 rules, one for every term constructor

  $$x \mid MM \mid \lambda x : M.\, M$$

- λP

  $5 + 2 = 7$ rules: one for every term constructor

  $$x \mid MM \mid \lambda x : M.\, M \mid \Pi x : M.\, M \mid *$$

  but:

  - *two* for typing variables from a context
  - *conversion* rule for $=_\beta$ on dependent type arguments

judgments

order of variables in context now matters:

$$\underbrace{x_1 : A_1, \ldots, x_n : A_n}_{\text{no longer set but list}} \vdash M : B$$

x_i can occur in A_j if $i < j$

rule 1: start rule

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

no $\lambda\!\rightarrow$ counterpart

only λP rule without antecedents

also called *axiom* rule

rule 2: application

$\lambda{\rightarrow}$:

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

λP:

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

rule 3: abstraction

$\lambda{\rightarrow}$:

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

λP:

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

$\Gamma \vdash (\Pi x : A.\, B) : s$ establishes that $(\Pi x : A.\, B)$ is allowed

A should not be $*$!

rule 4: product

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

no $\lambda\!\!\rightarrow$ counterpart

$\lambda\!\!\rightarrow$ types are always correct

two product rules

type of dependent functions:

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

type of dependent *types*:

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

$$\begin{aligned} \mathsf{vec} &: \mathsf{nat} \to * \\ \mathsf{vec} &: \color{red}{\Pi n : \mathsf{nat}.\, *} \end{aligned}$$

$$\frac{\dfrac{\vdots}{\mathsf{nat} : * \vdash \mathsf{nat} : *} \qquad \dfrac{\vdots}{\mathsf{nat} : *,\ n : \mathsf{nat} \vdash * : \Box}}{\mathsf{nat} : * \vdash (\Pi n : \mathsf{nat}.\, *) : \Box}$$

rules 5 and 6: variable and weakening

$\lambda{\rightarrow}$:

$$\frac{}{\Gamma \vdash x : A} \quad x : A \in \Gamma$$

λP:

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

typing a variable from a context

$$
\dfrac{\dfrac{\dfrac{\vdots}{x : A \vdash B : *}}{x : A, y : B \vdash y : B} \qquad \dfrac{\vdots}{x : A, y : B \vdash C : *}}{x : A, y : B, z : C \vdash y : B}
$$

conversion

dependent append

$$\text{append } 2\ 3\ \langle 1,2\rangle\ \langle 3,4,5\rangle = \langle 1,2,3,4,5\rangle$$

$$\begin{aligned} \text{append} &: \Pi n : \text{nat}.\, \Pi m : \text{nat}.\, \text{vec}\ n \to \text{vec}\ m \to \text{vec}\ (\text{plus}\ n\ m) \\ \text{plus} &: \text{nat} \to \text{nat} \to \text{nat} \end{aligned}$$

$$\begin{aligned} \text{append } 2\ 3\ \langle 1,2\rangle\ \langle 3,4,5\rangle &: \text{vec}\ (\text{plus}\ 2\ 3) \\ \langle 1,2,3,4,5\rangle &: \text{vec}\ 5 \end{aligned}$$

$$\text{plus}\ 2\ 3 \neq 5$$

$$\text{plus}\ 2\ 3 =_{\beta\delta\iota\zeta} 5$$

rule 7: conversion rule

λP:

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

conversion
typing rule

$$\Gamma \vdash M : A \quad \& \quad A \twoheadrightarrow_\beta A' \quad \Rightarrow \quad \Gamma \vdash M : A'$$

subject reduction
property

$$\Gamma \vdash M : A \quad \& \quad M \twoheadrightarrow_\beta M' \quad \Rightarrow \quad \Gamma \vdash M' : A$$

all λP rules together

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

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

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

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

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

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

pure type systems

PTS s

PTS = pure type system

framework for defining type systems

exactly like λP, *but:*

- start rules:

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

- product rules:

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

PTS parameters

λP:

$$\begin{array}{rcl} \mathcal{S} & = & \{*, \Box\} \\ \mathcal{A} & = & \{(*, \Box)\} \\ \mathcal{R} & = & \{\underbrace{(*, *)}_{(*,*,*)}, \underbrace{(*, \Box)}_{(*,\Box,\Box)}\} \end{array}$$

PTS given by

1. $\mathcal{S}$: *sorts*
2. $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$: axioms
3. $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$: rules

   (s_1, s_2, s_2) is written as (s_1, s_2)

$\lambda *$

$$\begin{array}{rcl} \mathcal{S} & = & \{*\} \\ \mathcal{A} & = & \{(*, *)\} \\ \mathcal{R} & = & \{(*, *)\} \end{array}$$

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

'star in star'
Per Martin-Löf, 1971

inconsistent: every type is inhabited

Girard's paradox $\leadsto$ *subject of last lecture!*
Jean-Yves Girard, 1972

λU^-

$$\begin{array}{rcl} \mathcal{S} & = & \{*, \Box, \triangle\} \\ \mathcal{A} & = & \{(*, \Box),\ (\Box, \triangle)\} \\ \mathcal{R} & = & \{(*, *),\ (\Box, *),\ (\Box, \Box),\ (\triangle, \Box)\} \end{array}$$

Hurkens' paradox

(λl:Πp:(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗.(Πv:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.(λs:ΠX:□.(((X→∗)→∗)→X)
→(X→∗)→∗.s(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X
→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xX
f))))t))vp→pv)→p((λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X
→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))(λp:(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗.Πw:ΠX:□.(((X→∗)→∗)→X)→(X→
∗)→∗.(λs:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.s(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→∗)→∗)→X)→
(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.
(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))t))wp→pw)).l(λy:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.(Πp:(ΠX:□.(((X→∗)→
∗)→X)→(X→∗)→∗)→∗.(λs:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.s(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.
(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.
λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))t))yp→p((λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→
∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))((λs:ΠX:□.(((X→∗)→∗)→X)
→(X→∗)→∗.s(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X
→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xX
f))))t))y)))→(ΠP: ∗ .P))(λz:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.λH:(λs:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.s(ΠX:□.
(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→
∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))t))z(λy:ΠX:□.
(((X→∗)→∗)→X)→(X→∗)→∗.(Πp:(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗.(λs:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.s
(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X→∗)→∗)→X)
→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))t))yp→p
((λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X
→∗)→∗.p(f(xXf))))((λs:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.s(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→∗)
→∗)→X)→(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t
(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))t))y)))→(ΠP: ∗ .P)).λl:Πp:(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→
∗.(λs:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.s(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→
∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→
∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))t))zp→p((λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→
X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))((λs:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.s(ΠX:□.(((X
→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)
→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))t))z)).l(λy:ΠX:□.
(((X→∗)→∗)→X)→(X→∗)→∗.(Πp:(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗.(λs:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.s
(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X→∗)→∗)→X)
→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))t))yp→p
((λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.t(λx:ΠX:□.(((X→∗)→∗)→X)→(X
→∗)→∗.p(f(xXf))))((λs:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.s(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)(λt:((ΠX:□.(((X→
∗)→∗)→X)→(X→∗)→∗)→∗)→∗.(λt:((ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗)→∗)→∗.λX:□.λf:((X→∗)→∗)→X.λp:X→∗.
t(λx:ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗.p(f(xXf))))t))y)))→(ΠP: ∗ .P))H(λp:(ΠX:□.(((X→∗)→∗)→X)→(X→∗)→∗

$\lambda{\rightarrow}$

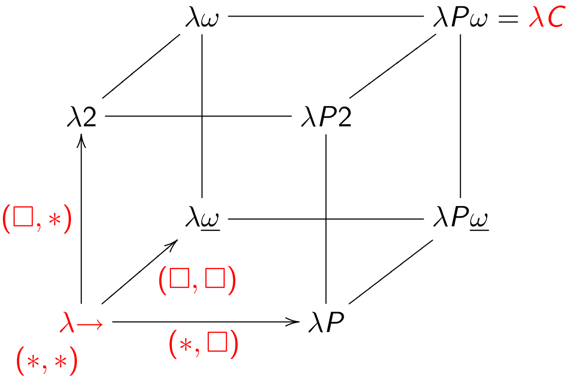
$$
\begin{aligned}
\mathcal{S} &= \{*, \Box\} \\
\mathcal{A} &= \{(*, \Box)\} \\
\mathcal{R} &= \{(*, *)\}
\end{aligned}
$$

PTS equivalent to system from last week!

$$
\frac{\frac{}{x : a \vdash x : a}}{\vdash (\lambda x : a.\, x) : a \rightarrow a}
\qquad
\frac{\frac{\frac{\frac{}{* : \Box}}{a : * \vdash a : *}}{a : *, x : a \vdash x : a} \quad \frac{\frac{\frac{}{* : \Box}}{a : * \vdash a : *} \quad \frac{\frac{\frac{}{* : \Box}}{a : * \vdash a : *} \quad \frac{\frac{}{* : \Box}}{a : * \vdash a : *}}{a : *, x : a \vdash a : *}}{a : * \vdash a \rightarrow a : *}}{a : * \vdash (\lambda x : a.\, x) : a \rightarrow a}
$$

Barendregt cube

= lambda cube



calculus of constructions

$$\mathcal{S} = \{*, \Box\}$$
$$\mathcal{A} = \{(*, \Box)\}$$

$\lambda\!\to$	$\mathcal{R} = \{(*, *)\}$	
λP	$\mathcal{R} = \{(*, *), (*, \Box)\}$	dependent types
$\lambda 2$	$\mathcal{R} = \{(*, *), \quad (\Box, *)\}$	polymorphism
λC	$\mathcal{R} = \{(*, *), (*, \Box), (\Box, *), (\Box, \Box)\}$	both

λC = CC = Calculus of Constructions

Thierry Coquand, 1985

Curry-Howard

Curry-Howard isomorphism for predicate logic

as always:

proofs	$\leftrightarrow$	terms
introduction rules	$\leftrightarrow$	lambda abstraction
elimination rules	$\leftrightarrow$	function application
assumption rule	$\leftrightarrow$	variable

minimal predicate logic

$$\frac{}{\Gamma \vdash A} \; A \in \Gamma$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to I \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \to E$$

$$\frac{\Gamma \vdash A[x]}{\underset{\uparrow}{\Gamma} \vdash \forall x \in D.\, A[x]} \forall I \qquad \frac{\Gamma \vdash \forall x \in D.\, A[x]}{\Gamma \vdash A[t]} \forall E$$

only if x not free in Γ

Curry-Howard for implication

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to I \qquad \frac{\Gamma, H : A \vdash M : B \qquad \ldots}{\Gamma \vdash \lambda H : A.\, M : A \to B}$$

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

Curry-Howard for universal quantification

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x \in D.\, A}\,\forall I \qquad \frac{\Gamma, x : D \vdash M : A \qquad \ldots}{\Gamma \vdash \lambda x : D.\, M \;:\; \Pi x : D.\, A}$$

$$\frac{\Gamma \vdash \forall x \in D.\, A}{\Gamma \vdash A[x := t]}\,\forall E \qquad \frac{\Gamma \vdash M : \Pi x : D.\, A \qquad \Gamma \vdash t : D}{\Gamma \vdash Mt : A[x := t]}$$

recap

1. dependent types
2. dependent functions
3. λP
4. conversion
5. pure type systems
6. Curry-Howard