Lecture 3: Typed Lambda Calculus and Curry-Howard

H. Geuvers

Radboud University
Nijmegen, NL

21st Estonian Winter School in Computer Science
Winter 2016

Outline

Typed λ calculus as a basis for logic

λ-term	:	type
M	:	A
program	:	data type
proof	:	formula
program	:	(full) specification

Aim:

- Type Theory as an integrated system for proving and programming.
- Type Theory as a basis for proof assistants and interactive theorem proving.

Simple type theory

Simplest system: $\lambda\!\!\rightarrow$ or simple type theory, STT. Just arrow types

$$\mathsf{Typ} := \mathsf{TVar} \mid (\mathsf{Typ} \rightarrow \mathsf{Typ})$$

- Examples: $(\alpha \rightarrow \beta) \rightarrow \alpha$, $(\alpha \rightarrow \beta) \rightarrow ((\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \gamma))$
- Brackets associate to the right and outside brackets are omitted:
  $(\alpha \rightarrow \beta) \rightarrow (\beta \rightarrow \gamma) \rightarrow \alpha \rightarrow \gamma$
- Types are denoted by $A, B, \ldots$.

Simple type theory à la Church

Formulation with contexts to declare the free variables:

$$x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$$

is a context, usually denoted by Γ.
Derivation rules of $\lambda\!\!\rightarrow$ (à la Church):

$$\frac{x{:}A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B} \qquad \frac{\Gamma, x{:}A \vdash P : B}{\Gamma \vdash \lambda x{:}A.P : A \to B}$$

$\Gamma \vdash_{\lambda\!\!\rightarrow} M : A$ if there is a derivation using these rules with conclusion $\Gamma \vdash M : A$

Examples

$$\vdash \lambda x : A.\lambda y : B.x \quad : \quad A \to B \to A$$

$$\vdash \lambda x : A \to B.\lambda y : B \to C.\lambda z : A.y\,(x\,z) \quad : \quad (A \to B) \to (B \to C) \to A \to C$$

$$\vdash \lambda x : A.\lambda y : (B \to A) \to A.y(\lambda z : B.x) \quad : \quad A \to ((B \to A) \to A) \to A$$

Not for every type there is a closed term of that type:

$$(A \to A) \to A \text{ is not inhabited}$$

That is: there is no term M such that

$$\vdash M : (A \to A) \to A.$$

Typed Terms versus Type Assignment

- With typed terms also called typing à la Church, we have terms with type information in the λ-abstraction

$$\lambda x : A.x \;\; : \;\; A \to A$$

  - Terms have unique types,
  - The type is directly computed from the type info in the variables.

- With typed assignment also called typing à la Curry, we assign types to untyped λ-terms

$$\lambda x.x : A \to A$$

  - Terms do not have unique types,
  - A principal type can be computed using unification.

Church vs. Curry typing

- The Curry formulation is especially interesting for programming: you want to write as little type information as possible; let the compiler infer the types for you.
- The Church formulation is especially interesting for proof checking: terms are created interactively; type structure is so intricate that type inference is undecidable (if you start from an untyped term).
  [This lecture]

Formulas-as-Types (Curry, Howard)

Recall: there are two readings of a judgement $M : A$

1. term as algorithm/program, type as specification:
   M is a function of type A
2. type as a proposition, term as its proof:
   M is a proof of the proposition A

- There is a one-to-one correspondence:

  typable terms in $\lambda\!\!\rightarrow \simeq$ derivations in minimal proposition logic

- $x_1 : B_1, x_2 : B_2, \ldots, x_n : B_n \vdash M : A$ can be read as
  M is a proof of A from the assumptions $B_1, B_2, \ldots, B_n$.

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{[A \to B \to C]^3 \;\; [A]^1}{B \to C} \qquad \dfrac{[A \to B]^2 \;\; [A]^1}{B}}{C}\,1}{(A \to B) \to A \to C}\,2}{(A \to B \to C) \to (A \to B) \to A \to C}\,3$$

$\simeq$

$$\lambda x{:}A \to B \to C.\lambda y{:}A \to B.\lambda z{:}A.x\ z\ (y\ z)$$
$$: (A \to B \to C) \to (A \to B) \to A \to C$$

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{[x : A \to B \to C]^3 \;\; [z : A]^1}{x\,z : B \to C} \qquad \dfrac{[y : A \to B]^2 \;\; [z : A]^1}{y\,z : B}}{x\,z\,(y\,z) : C}\,1}{\lambda z{:}A.x\,z\,(y\,z) : A \to C}\,2}{\lambda y{:}A \to B.\lambda z{:}A.x\,z\,(y\,z) : (A \to B) \to A \to C}\,3$$

$$\lambda x{:}A \to B \to C.\lambda y{:}A \to B.\lambda z{:}A.x\,z\,(y\,z) : (A \to B \to C) \to (A \to B) \to A \to C$$

Exercise: Give the derivation that corresponds to

$$\lambda x{:}C \to E.\lambda y{:}(C \to E) \to E.y(\lambda z.y\,x) :$$
$$(C \to E) \to ((C \to E) \to E) \to E$$

Typed Combinatory Logic

We have seen Combinatory Logic with the axioms for **I**, **K** and **S**.
We now know their typed definition in $\lambda\!\!\rightarrow$:

$$\begin{array}{rcll} \mathbf{I} & := & \lambda x:A.x & : \quad A \to A \\ \mathbf{K} & := & \lambda x:A.\lambda y:B.x & : \quad A \to B \to A \\ \mathbf{S} & := & \lambda x{:}A \to B \to C.\lambda y{:}A \to B.\lambda z{:}A.x\,z\,(y\,z) & \\ & & : \quad (A \to B \to C) \to (A \to B) \to A \to C & \end{array}$$

- The three axiom schemes $A \to A$, $A \to B \to A$ and $(A \to B \to C) \to (A \to B) \to A \to C$ together with the derivation rule Modus Ponens is exactly Hilbert style minimal proposition logic.
- The typed CL terms are exactly the derivations in this logic.
- Modus Ponens corresponds with Application in CL

Exercise: Show that the scheme $A \to A$ is derivable.
Cast in CL terminology: **I** can be defined in terms of **S** and **K**. To be precise: $\mathbf{I} = \mathbf{S}\,\mathbf{K}\,\mathbf{K}$.

Computation = Cut-elimination

- β-reduction: $(\lambda x{:}A.M)P \to_\beta M[x := P]$

Cut-elimination in minimal logic = β-reduction in $\lambda{\to}$.

$$\dfrac{\dfrac{\begin{array}{c}[A]^1\\ \mathcal{D}_1\\ B\end{array}}{A \to B}\,1 \qquad \dfrac{\mathcal{D}_2}{A}}{B} \quad \longrightarrow \quad \begin{array}{c}\mathcal{D}_2\\ A\\ \mathcal{D}_1\\ B\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}[x : A]^1\\ \mathcal{D}_1\\ M : B\end{array}}{\lambda x{:}A.M : A \to B}\,1 \qquad \dfrac{\mathcal{D}_2}{P : A}}{(\lambda x{:}A.M)P : B} \quad \longrightarrow_\beta \quad \begin{array}{c}\mathcal{D}_2\\ P : A\\ \mathcal{D}_1\\ M[x := P] : B\end{array}$$

Example

Proof of $A \to A \to B, (A \to B) \to A \vdash B$ with a cut.

$$\dfrac{\dfrac{[A]^1 \quad \dfrac{[A]^1 \quad A \to A \to B}{A \to B}}{B} \qquad \dfrac{(A \to B) \to A \qquad \dfrac{[A]^1 \quad A \to A \to B}{A \to B}}{A}}{B}$$

It contains a cut: a $\to$-i directly followed by an $\to$-e.

Example proof with term information

$$\dfrac{\dfrac{[y : A]^1 \quad \dfrac{[y : A]^1 \quad p : A \to A \to B}{p\,y : A \to B}}{p\,y\,y : B}}{\lambda y{:}A.p\,y\,y : A \to B} \qquad \dfrac{\dfrac{[x : A]^1 \quad \dfrac{[x : A]^1 \quad p : A \to A}{p\,x : A \to B}}{p\,x\,x : B}}{\lambda x{:}A.p\,x\,x : A \to B} \quad q : (A \to B) \to A}{(\lambda y{:}A.p\,y\,y)(q(\lambda x{:}A.p\,x\,x)) : B}$$

Term contains a β-redex: $(\lambda x{:}A.p\,x\,x)\,(q(\lambda x{:}A.p\,x\,x))$

Extension with other connectives

Adding product types $\times$ to $\lambda\!\!\rightarrow$. (Proposition logic with conjunction $\wedge$.)

$$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : B} \qquad \frac{\Gamma \vdash P : A \;\; \Gamma \vdash Q : B}{\Gamma \vdash \langle P, Q \rangle : A \times B}$$

With reduction rules

$$\begin{array}{lcl} \pi_1 \langle P, Q \rangle & \rightarrow & P \\ \pi_2 \langle P, Q \rangle & \rightarrow & Q \end{array}$$

Similar rules can be given for sum-types $A + B$, corresponding to disjunction $A \vee B$.

Extension to predicate logic

- First order language: domain D, with variables $x, y, z : D$ and possibly functions over D, e.g. $f : D \to D$, $g : D \to D \to D$.
- Rules for $\forall x{:}D.\phi$ and $\exists x{:}D.\phi$.
- 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).
- Formulas and domain are both types. What is the type of a predicate or relation?
- A predicate P is a map from D to the collection of types, $*$
- $P : D \to *$ for P a predicate and $R : D \to D \to *$ for R a binary relation on D.
- We will have to make this more precise ...

Idea of extending to $\forall$

Term rules for the $\forall$-quantifier in predicate logic.

$$\frac{\Gamma \vdash M : \forall x{:}D.A}{\Gamma \vdash M\,t : A[x := t]} \text{ if } t : D \qquad \frac{\Gamma \vdash M : A}{\Gamma \vdash \lambda x{:}D.M : \forall x{:}D.A} \; x \text{ not free in } \Gamma$$

With the usual β-reduction rule

$$(\lambda x{:}D.M)t \to M[x := t]$$

.

This conforms with cut-elimination (or "detour elimination") on logical derivations.

Example

Deriving irreflexivity from anti-symmetry

$$\begin{aligned} \text{AntiSym}\, R &:= \forall x, y{:}D.(Rxy) \to (Ryx) \to \bot \\ \text{Irrefl}\, R &:= \forall x{:}D.(Rxx) \to \bot \end{aligned}$$

Derivation in predicate logic:

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall x, y{:}D.R\,x\,y \to R\,y\,x \to \bot}{\forall y{:}D.R\,x\,y \to R\,y\,x \to \bot}}{R\,x\,x \to R\,x\,x \to \bot} \qquad [R\,x\,x]^1}{R\,x\,x \to \bot} \qquad [R\,x\,x]^1}{\dfrac{\dfrac{\bot}{R\,x\,x \to \bot}\,1}{\forall x{:}D.R\,x\,x \to \bot}}$$

Example derivation in type theory, with terms

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{H : \forall x, y{:}D.R\,x\,y \to R\,y\,x \to \bot}{H\,x : \forall y{:}D.R\,x\,y \to R\,y\,x \to \bot}}{H\,x\,x : R\,x\,x \to R\,x\,x \to \bot} \qquad [H' : R\,x\,x]^1}{H\,x\,x\,H' : R\,x\,x \to \bot} \qquad [H' : R\,x\,x]^1}{H\,x\,x\,H'\,H' : \bot}}{\lambda H'{:}(R\,x\,x).H\,x\,x\,H'\,H' : R\,x\,x \to \bot}\,1}{\lambda x{:}A.\lambda H'{:}(R\,x\,x).H\,x\,x\,H'\,H' : \forall x{:}D.R\,x\,x \to \bot}$$

Dependent Type Theory

- We have seen informally "dependent types at work" in the predicate logic example.
- Now: the rules

With dependent types:

- everything depends on everything
- we can't first define the types, then the terms
- two universes: $*$ and $\Box$
- $*$ is the universe of types
- We can't have $* : *$, so we have another universe: $* : \Box$.

NB The Coq system uses "Set" and "Prop" for what I call $*$ and "Type" for what I call $\Box$.

First order Dependent Type theory, λP

Derive judgements of the form

$$\Gamma \vdash M : B$$

- Γ is a context

$$x_1 : B_1, x_2 : B_2, \ldots, x_n : B_n$$

- M and B are terms
  taken from the set of pseudoterms

$$\mathsf{T} ::= \mathsf{Var} \mid * \mid \Box \mid (\mathsf{T}\ \mathsf{T}) \mid (\lambda x{:}\mathsf{T}.\mathsf{T}) \mid \Pi x{:}\mathsf{T}.\mathsf{T}$$

Auxiliary judgement

$$\Gamma \vdash$$

denoting that Γ is a correct context.

Derivation rules of λP

s ranges over $\{*, \Box\}$.

$$\text{(base)}\ \emptyset \vdash \qquad \text{(ctxt)}\ \frac{\Gamma \vdash A : \mathbf{s}}{\Gamma, x{:}A \vdash}\ \text{if } x \text{ not in } \Gamma \qquad \text{(ax)}\ \frac{\Gamma \vdash}{\Gamma \vdash * : \Box}$$

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

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

$$\text{(conv)}\ \frac{\Gamma \vdash M : B \quad \Gamma \vdash A : \mathbf{s}}{\Gamma \vdash M : A}\ A =_{\beta\eta} B$$

Notation: write $A \to B$ for $\Pi x{:}A.B$ if $x \notin \mathrm{FV}(B)$.

The use of the Π-type

- The Π rule allows to form two forms of function types.

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

$$\Pi x{:}A.B \simeq \{f \mid \forall a : A(f\ a : B[x := a])\}$$

Write $A \to B$ if $x \notin \mathrm{FV}(B)$

- With $\mathbf{s} = *$, we can form $\mathsf{D} \to \mathsf{D}$ and $\Pi x{:}\mathsf{D}.x = x$, etc.
- With $\mathbf{s} = \Box$, we can form $\mathsf{D} \to \mathsf{D} \to *$ and $\mathsf{D} \to *$.

Representation of PRED (minimal predicate logic) into λP

Represent both the domains of the logic and the formulas as types.

$$\begin{array}{lcl} \mathsf{A} & : & *, \\ P & : & \mathsf{A} \to *, \\ R & : & \mathsf{A} \to \mathsf{A} \to *, \end{array}$$

Now implication is represented as $\to$ and $\forall$ is represented as Π:

$$\forall x{:}A.P\,x \mapsto \Pi x{:}A.P\,x$$

Intro and elim rules are just λ-abstraction and application

Example

$$\begin{array}{rcl} \mathsf{A} : *, R : \mathsf{A} \to \mathsf{A} \to * & \vdash & \lambda z{:}A.\lambda h{:}(\Pi x, y{:}A.R\,x\,y).h\,z\,z \\ & : & \Pi z{:}A.(\Pi x, y{:}A.R\,x\,y) \to R\,z\,z \end{array}$$

This term is a proof of $\forall z{:}A.(\forall x, y{:}A.R(x, y)) \to R(z, z)$
Exercise: Find terms of the following types (NB $\to$ binds strongest)

$$(\Pi x{:}A.P\,x \to Q\,x) \to (\Pi x{:}A.P\,x) \to \Pi x{:}A.Q\,x$$

and

$$(\Pi x{:}A.P\,x \to \Pi z.R\,z\,z) \to (\Pi x{:}A.P\,x) \to \Pi z{:}A.R\,z\,z).$$

Also write down the contexts in which these terms are typed.

Direct embedding of logic in type theory

For $\lambda\!\!\rightarrow$ and λP we have seen

Direct representations of logic in type theory.

- Connectives each have a counterpart in the type theory:
  implication $\sim$ $\rightarrow$-type
  universal quantification $\sim$ $\forall$-type
- Logical rules have their direct counterpart in type theory
  λ-abstraction $\sim$ $\rightarrow$-introduction
  application $\sim$ $\rightarrow$- elimination λ-abstraction $\sim$ $\forall$-introduction
  application $\sim$ $\forall$-elimination
- Context declares signature, local varibales and assumptions.

LF embedding of logic in type theory

Second way of interpreting logic in type theory De Bruijn:

Logical framework encoding of logic in type theory.

- Type theory used as a meta system for encoding ones own logic.
- Choose an appropriate context Γ_L, in which the logic L (including its proof rules) is declared.
- Context used as a signature for the logic.
- Use the type system as the 'meta' calculus for dealing with substitution and binding.

Direct and LF embedding

	proof	formula
direct embedding	$\lambda x{:}A.x$	$A \to A$
LF embedding	$\mathsf{imp\_intr}\ A\ A\ \lambda x{:}T\ A.x$	$T(A \Rightarrow A)$

- Direct representation: One type system : One logic, Logical rules $\sim$ type theoretic rules
- LF encoding One type system : Many logics, Logical rules $\sim$ context declarations

Examples of the Deep embedding

The encoding of logics in a logical framework is shown by three examples:

1. Minimal proposition logic
2. Minimal predicate logic (just $\{\Rightarrow, \forall\}$)
3. Untyped λ-calculus

Minimal propositional logic

Fix the signature (context) of minimal propositional logic.

$$\begin{aligned} \textbf{prop} &: * \\ \mathsf{imp} &: \textbf{prop} \to \textbf{prop} \to \textbf{prop} \end{aligned}$$

Notation:

$$A \Rightarrow B \text{ for } \mathsf{imp}\, A\, B$$

The type **prop** is the type of 'names' of propositions.
NB : A term of type **prop**can not be inhabited (proved), as it is not a type.
We 'lift' a name p : **prop** to the type of its proofs by introducing the following map:

$$\mathsf{T} \quad : \quad \textbf{prop} \to *.$$

Intended meaning of $\mathsf{T}p$ is 'the type of proofs of p'.
We interpret 'p is valid' by '$\mathsf{T}p$ is inhabited'.

Encoding of derivations

To derive $\mathsf{T}p$ we also encode the logical derivation rules

$$\begin{aligned} \text{imp\_intr} &: \Pi p, q : \text{prop}.(\mathsf{T}p \to \mathsf{T}q) \to \mathsf{T}(p \Rightarrow q), \\ \text{imp\_el} &: \Pi p, q : \text{prop}.\mathsf{T}(p \Rightarrow q) \to \mathsf{T}p \to \mathsf{T}q. \end{aligned}$$

New phenomenon: Π-type:

$$\Pi x{:}A.B(x) \simeq \text{the type of functions } f \text{ such that } f\ a : B(a) \text{ for all } a{:}A$$

imp_intr takes two (names of) propositions p and q and a term $f : \mathsf{T}\,p \to \mathsf{T}\,q$ and returns a term of type $\mathsf{T}(p \Rightarrow q)$
Indeed $A \Rightarrow A$, becomes valid:

$$\text{imp\_intr}A\,A(\lambda x{:}\mathsf{T}\,A.x) : \mathsf{T}(A \Rightarrow A)$$

Exercise: Construct a term of type $\mathsf{T}(A \Rightarrow (B \Rightarrow A))$

Signature of PROP in LF

To encode proposition logic in LF we need a context (signature) Σ_{PROP}:

$$
\begin{array}{rcl}
\textbf{prop} & : & * \\
\Rightarrow & : & \textbf{prop} \to \textbf{prop} \to \textbf{prop} \\
\mathsf{T} & : & \textbf{prop} \to * \\
\text{imp\_intr} & : & (A, B : \textbf{prop})(\mathsf{T}\, A \to \mathsf{T}\, B) \to \mathsf{T}(A \Rightarrow B) \\
\text{imp\_el} & : & (A, B : \textbf{prop})\mathsf{T}(A \Rightarrow B) \to \mathsf{T}\, A \to \mathsf{T}\, B.
\end{array}
$$

Desired properties of the encoding:

- Adequacy (soundness) of the encoding:

  $\vdash_{\text{PROP}} A \implies \Sigma_{\text{PROP}}, a_1{:}\text{prop}, \ldots, a_n{:}\text{prop} \vdash p : \mathsf{T}\, A$ for some

  $\{a, \ldots, a_n\}$ is the set of proposition variables in A.
- Faithfulness (or completeness) is the converse. It also holds, but more involved to prove.

Minimal predicate logic over one domain A

Signature:

$$
\begin{array}{rcl}
\textbf{prop} & : & *, \\
\mathsf{A} & : & *, \\
\mathsf{T} & : & \textbf{prop} \to * \\
\mathsf{f} & : & \mathsf{A} \to \mathsf{A}, \\
\mathsf{R} & : & \mathsf{A} \to \mathsf{A} \to \textbf{prop}, \\
\Rightarrow & : & \textbf{prop} \to \textbf{prop} \to \textbf{prop}, \\
\mathsf{imp\_intr} & : & \Pi p, q : \textbf{prop}.(\mathsf{T}p \to \mathsf{T}q) \to \mathsf{T}(p \Rightarrow q), \\
\mathsf{imp\_el} & : & \Pi p, q : \textbf{prop}.\mathsf{T}(p \Rightarrow q) \to \mathsf{T}p \to \mathsf{T}q.
\end{array}
$$

Now encode $\forall$: $\forall$ takes a $P : \mathsf{A} \to \textbf{prop}$ and returns a proposition, so we add:

$$\forall : (\mathsf{A} \to \textbf{prop}) \to \textbf{prop}$$

Minimal predicate logic over one domain A

Signature: Σ_{PRED}

$$
\begin{array}{rcl}
\textbf{prop} & : & *, \\
\mathsf{A} & : & *, \\
 & \vdots & \\
\mathsf{imp\_intr} & : & \Pi p, q : \textbf{prop}.(\mathsf{T}p \to \mathsf{T}q) \to \mathsf{T}(p \Rightarrow q), \\
\mathsf{imp\_el} & : & \Pi p, q : \textbf{prop}.\mathsf{T}(p \Rightarrow q) \to \mathsf{T}p \to \mathsf{T}q.
\end{array}
$$

Now encode $\forall$: $\forall$ takes a $P : \mathsf{A} \to \textbf{prop}$ and returns a proposition, so:

$$\forall : (\mathsf{A} \to \textbf{prop}) \to \textbf{prop}$$

Universal quantification is translated as follows.

$$\forall x{:}A.(Px) \mapsto \forall(\lambda x{:}A.(Px))$$

Intro and Elim rules for $\forall$

$$\begin{array}{rcl} \forall & : & (\mathsf{A} \to \mathbf{prop}) \to \mathbf{prop}, \\ \forall\_\mathsf{intr} & : & \Pi P{:}\mathsf{A} \to \mathsf{prop}.(\Pi x{:}\mathsf{A}.\mathsf{T}(Px)) \to \mathsf{T}(\forall P), \\ \forall\_\mathsf{elim} & : & \Pi P{:}\mathsf{A} \to \mathsf{prop}.\mathsf{T}(\forall P) \to \Pi x{:}\mathsf{A}.\mathsf{T}(Px). \end{array}$$

The proof of

$$\forall z{:}A(\forall x, y{:}A.Rxy) \Rightarrow Rzz$$

is now mirrored by the proof-term

$$\begin{array}{rl} \forall\_\mathsf{intr}[\_](& \lambda z{:}\mathsf{A}.\mathsf{imp\_intr}[\_][\_](\lambda h{:}\mathsf{T}(\forall x, y{:}A.Rxy). \\ & \forall\_\mathsf{elim}[\_](\forall\_\mathsf{elim}[\_]hz)z)\ ) \end{array}$$

We have replaced the instantiations of the Π-type by $[\_]$.
This term is of type

$$\mathsf{T}(\forall(\lambda z{:}\mathsf{A}.\mathsf{imp}(\forall(\lambda x{:}\mathsf{A}.(\forall(\lambda y{:}\mathsf{A}.\mathsf{R}xy))))(\mathsf{R}zz)))$$

Intro and Elim rules for $\forall$

$$\begin{array}{rcl} \forall & : & (\mathsf{A} \to \mathbf{prop}) \to \mathbf{prop}, \\ \forall\_\mathsf{intr} & : & \Pi P{:}\mathsf{A} \to \mathsf{prop}.(\Pi x{:}\mathsf{A}.\mathsf{T}(Px)) \to \mathsf{T}(\forall P), \\ \forall\_\mathsf{elim} & : & \Pi P{:}\mathsf{A} \to \mathsf{prop}.\mathsf{T}(\forall P) \to \Pi x{:}\mathsf{A}.\mathsf{T}(Px). \end{array}$$

The proof of

$$\forall z{:}A(\forall x, y{:}A.Rxy) \Rightarrow Rzz$$

is now mirrored by the proof-term

$$\begin{array}{rl} \forall\_\mathsf{intr}[\_](& \lambda z{:}\mathsf{A}.\mathsf{imp\_intr}[\_][\_](\lambda h{:}\mathsf{T}(\forall x, y{:}A.Rxy). \\ & \forall\_\mathsf{elim}[\_](\forall\_\mathsf{elim}[\_]hz)z)\ ) \end{array}$$

Exercise: Construct a proof-term that mirrors the (obvious) proof of

$$\forall x(P\,x \Rightarrow Q\,x) \Rightarrow \forall x.P\,x \Rightarrow \forall x.Q\,x$$

$$\begin{array}{rcl} \text{Signature } \Sigma_{\text{lambda}}: \quad \mathsf{D} & : & *; \\ \mathsf{app} & : & \mathsf{D} \to (\mathsf{D} \to \mathsf{D}); \\ \mathsf{abs} & : & (\mathsf{D} \to \mathsf{D}) \to \mathsf{D}. \end{array}$$

- A variable x in λ-calculus becomes $x : \mathsf{D}$ in the type system.
- The translation $[-] : \Lambda \to \mathrm{Term}(\mathsf{D})$ is defined as follows.

$$\begin{array}{rcl} [x] & = & x; \\ [PQ] & = & \mathsf{app}\ [P]\ [Q]; \\ [\lambda x.P] & = & \mathsf{abs}\ (\lambda x{:}\mathsf{D}.[P]). \end{array}$$

Examples: $[\lambda x.xx] := \mathsf{abs}(\lambda x{:}\mathsf{D}.\mathsf{app}\, x\, x)$
$[(\lambda x.xx)(\lambda y.y)] := \mathsf{app}(\mathsf{abs}(\lambda x{:}\mathsf{D}.\mathsf{app}\, x\, x))(\mathsf{abs}(\lambda y{:}\mathsf{D}.y))$.

Introducing β-equality

$$\mathsf{eq}{:}\mathsf{D} \to \mathsf{D} \to *.$$

Notation $P = Q$ for eq P Q.
Rules for proving equalities.

$$\begin{array}{rcl}
\mathsf{refl} & : & \Pi x{:}\mathsf{D}.x = x, \\
\mathsf{sym} & : & \Pi x, y{:}\mathsf{D}.x = y \to y = x, \\
\mathsf{trans} & : & \Pi x, y, z{:}\mathsf{D}.x = y \to y = z \to x = z, \\
\mathsf{mon} & : & \Pi x, x', z, z'{:}\mathsf{D}.x = x' \to z = z' \to (\mathsf{app}\ z\ x) = (\mathsf{app}\ z'\ x'), \\
\mathsf{xi} & : & \Pi f, g{:}\mathsf{D} \to \mathsf{D}.(\Pi x{:}\mathsf{D}.(fx) = (gx)) \to (\mathsf{abs}\ f) = (\mathsf{abs}\ g), \\
\mathsf{beta} & : & \Pi f{:}\mathsf{D} \to \mathsf{D}.\Pi x{:}\mathsf{D}.(\mathsf{app}(\mathsf{abs}\ f)x) = (fx).
\end{array}$$

Properties of λP

- Uniqueness of types
  If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma =_{\beta\eta} \tau$.
- Subject Reduction
  If $\Gamma \vdash M : \sigma$ and $M \to_{\beta\eta} N$, then $\Gamma \vdash N : \sigma$.
- Strong Normalization
  If $\Gamma \vdash M : \sigma$, then all $\beta\eta$-reductions from M terminate.

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

Decidability Questions

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

For λP:

- TIP is undecidable
- TCP/TSP: simultaneously with Context checking

Curry-Howard-de Bruijn

logic	$\sim$	**type theory**
formula	$\sim$	type
proof	$\sim$	term
detour elimination	$\sim$	β-reduction
proposition logic	$\sim$	simply typed λ-calculus
predicate logic	$\sim$	dependently typed λ-calculus λP
intuitionistic logic	$\sim$	$\ldots +$ inductive types
higher order logic	$\sim$	$\ldots +$ higher types and polymorphism
classical logic	$\sim$	$\ldots +$ exceptions