Proofs as Programs Summer School
Eugene Oregon June - July 2002

Type Systems

Herman Geuvers
Nijmegen University, NL

Lecture 3: Dependent Type Theory / Logical Framework

For $\lambda\!\!\to$ and $\lambda 2$:

Direct encoding (deep embedding) of logic in type theory.

- Connectives each have a counterpart in the type theory:
  implication $\sim$ arrow type
- logical rules have their direct counterpart in type theory
  λ-abstraction $\sim$ implication introduction
  application $\sim$ implication elimination
- Context declares assumptions

Second way of interpreting logic in type theory De Bruijn:

Logical framework encoding or shallow embedding 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 encoding	Shallow encoding
One type system : One logic	One type system : Many logics
Logical rules $\sim$ type theoretic rules	Logical rules $\sim$ context declarations

Direct encoding	Shallow encoding
One type system : One logic	One type system : Many logics
Logical rules $\sim$ type theoretic rules	Logical rules $\sim$ context declarations

Plan:

- First show examples of logics in a logical framework
- Then define precisely the type theory of the logical framework

Use **type** to denote the universe of types.

Direct encoding	Shallow encoding
One type system : One logic	One type system : Many logics
Logical rules $\sim$ type theoretic rules	Logical rules $\sim$ context declarations

Plan:

- First show examples of logics in a logical framework
- Then define precisely the type theory of the logical framework

Use **type** to denote the universe of types.

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

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

Minimal propositional logic

Fix the signature (context) of minimal propositional logic.

$$\begin{aligned} \mathsf{prop} &: \mathbf{type} \\ \mathsf{imp} &: \mathsf{prop} \to \mathsf{prop} \to \mathsf{prop} \end{aligned}$$

Notation:

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

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.

Minimal propositional logic

Fix the signature (context) of minimal propositional logic.

$$\begin{aligned} \mathsf{prop} &: \mathbf{type} \\ \mathsf{imp} &: \mathsf{prop} \to \mathsf{prop} \to \mathsf{prop} \end{aligned}$$

Notation:

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

The type prop is the type of 'names' of propositions.

We 'lift' a name p : prop to the type of its proofs by introducing the following map:

$$\mathsf{T} : \mathsf{prop} \to \mathbf{type}.$$

Intended meaning of $\mathsf{T}p$ is 'the type of proofs of p'.

We interpret 'p is valid' by '$\mathsf{T}p$ is inhabited'.

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

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

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 $\mathsf{T}(p \supset q)$

Indeed $A \supset A$, becomes valid:

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

Define

Σ_{PROP} to be the signature for minimal proposition logic, PROP.

Desired properties of the encoding:

- Adequacy (soundness) of the encoding:

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

  $\{a, \ldots, a_n\}$ is the set of proposition variables in A.
  Proof by induction on the derivation of $\vdash_{\mathsf{PROP}} A$.

- Faithfulness (or completeness) is the converse. It also holds, but more involved to prove.

Minimal predicate logic over one domain A (just $\supset$ and $\forall$

Signature:

$$\begin{aligned}
\mathsf{prop} &: \mathbf{type}, \\
\mathsf{A} &: \mathbf{type}, \\
\mathsf{c} &: \mathsf{A}, \\
\mathsf{f} &: \mathsf{A}{\rightarrow}\mathsf{A}, \\
\mathsf{R} &: \mathsf{A}{\rightarrow}\mathsf{A}{\rightarrow}\mathsf{prop}, \\
\supset &: \mathsf{prop}{\rightarrow}\mathsf{prop}{\rightarrow}\mathsf{prop}, \\
\mathsf{imp_intr} &: \Pi p, q : \mathsf{prop}.(\mathsf{T}p{\rightarrow}\mathsf{T}q){\rightarrow}\mathsf{T}(p \supset q), \\
\mathsf{imp_el} &: \Pi p, q : \mathsf{prop}.\mathsf{T}(p \supset q){\rightarrow}\mathsf{T}p{\rightarrow}\mathsf{T}q.
\end{aligned}$$

Now encode $\forall$:

$\forall$ takes a P : $\mathsf{A}{\rightarrow}\mathsf{prop}$ and returns a proposition, so:

$$\forall : (\mathsf{A}{\rightarrow}\mathsf{prop}){\rightarrow}\mathsf{prop}$$

Minimal predicate logic over one domain A (just $\supset$ and $\forall$

Signature: Σ_{PRED}

$$
\begin{aligned}
\mathsf{prop} &: \mathbf{type}, \\
\mathsf{A} &: \mathbf{type}, \\
&\vdots \\
\supset &: \mathsf{prop}{\rightarrow}\mathsf{prop}{\rightarrow}\mathsf{prop}, \\
\mathsf{imp\_intr} &: \Pi p, q : \mathsf{prop}.(\mathsf{T}p{\rightarrow}\mathsf{T}q){\rightarrow}\mathsf{T}(p \supset q), \\
\mathsf{imp\_el} &: \Pi p, q : \mathsf{prop}.\mathsf{T}(p \supset q){\rightarrow}\mathsf{T}p{\rightarrow}\mathsf{T}q.
\end{aligned}
$$

Now encode $\forall$:

$\forall$ takes a $P : \mathsf{A}{\rightarrow}\mathsf{prop}$ and returns a proposition, so:

$$\forall : (\mathsf{A}{\rightarrow}\mathsf{prop}){\rightarrow}\mathsf{prop}$$

Universal quantification is translated as follows.

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

Intro and elim rules for $\forall$:

$$\begin{aligned}
\text{forall} &: (\mathsf{A}\to\mathsf{prop})\to\mathsf{prop},\\
\text{forall_intr} &: \Pi P{:}\mathsf{A}\to\mathsf{prop}.(\Pi x{:}\mathsf{A}.\mathsf{T}(Px))\to\mathsf{T}(\text{forall}\,P),\\
\text{forall_elim} &: \Pi P{:}\mathsf{A}\to\mathsf{prop}.\mathsf{T}(\text{forall}\,P)\to\Pi x{:}\mathsf{A}.\mathsf{T}(Px).
\end{aligned}$$

The proof of

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

is now mirrored by the proof-term

$$\begin{aligned}
&\text{forall_intr}[_](\ \lambda z{:}\mathsf{A}.\text{imp_intr}[_][_](\lambda h{:}\mathsf{T}(\forall x,y{:}A.Rxy).\\
&\qquad\qquad \text{forall_elim}[_](\text{forall_elim}[_]hz))\ )
\end{aligned}$$

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

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

Again one can prove adequacy

$\vdash_{\text{PRED}} \varphi \Rightarrow \Sigma_{\text{PRED}}, x_1{:}\mathsf{A}, \ldots, x_n{:}\mathsf{A} \vdash p : T\varphi$, for some p,

where $\{x_1, \ldots, x_n\}$ is the set of free variables in φ.

Faithfulness can be proved as well.

Untyped λ-calculus

Signature Σ_{lambda}:

$$\begin{aligned} \mathsf{D} &: \mathbf{type}; \\ \mathsf{app} &: \mathsf{D}{\to}(\mathsf{D}{\to}\mathsf{D}); \\ \mathsf{abs} &: (\mathsf{D}{\to}\mathsf{D}){\to}\mathsf{D}. \end{aligned}$$

Encoding of λ-terms as terms of type D.

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

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

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 in Σ_{lambda} :

$$\mathsf{eq}{:}\mathsf{D}{\rightarrow}\mathsf{D}{\rightarrow}\mathbf{type}.$$

Notation $P = Q$ for eq $P\ Q$.

Rules for proving equalities.

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

Adequacy:

$P =_\beta Q \Rightarrow \Sigma_{\mathsf{lambda}}, x_1{:}\mathsf{D}, \ldots, x_n{:}\mathsf{D} \vdash p : [P] = [Q]$, for some p.

Here, $x_1, \ldots, x_n$ are the free variables in PQ

Faithfulness also holds.

Logical Framework, LF, or λP

Derive judgements of the form

$$\Gamma \vdash M : B$$

- Γ is a context
- M and B are terms
  taken from the set of pseudoterms

$$\mathsf{T} ::= \mathsf{Var} \,|\, \mathbf{type} \,|\, \mathbf{kind} \,|\, \mathsf{TT} \,|\, \lambda x{:}\mathsf{T}.\mathsf{T} \,|\, \Pi x{:}\mathsf{T}.\mathsf{T},$$

Auxiliary judgement

$$\Gamma \vdash$$

denoting that Γ is a correct context.

Derivation rules of LF. ($\mathbf{s}$ ranges over $\{\mathbf{type}, \mathbf{kind}\}$.)

(base) $\emptyset \vdash$ (ctxt) $\dfrac{\Gamma \vdash A : \mathbf{s}}{\Gamma, x{:}A \vdash}$ if x not in Γ (ax) $\dfrac{\Gamma \vdash}{\Gamma \vdash \mathbf{type} : \mathbf{kind}}$

(proj) $\dfrac{\Gamma \vdash}{\Gamma \vdash x : A}$ if $x{:}A \in \Gamma$ (Π) $\dfrac{\Gamma, x{:}A \vdash B : \mathbf{s} \quad \Gamma \vdash A : \mathbf{type}}{\Gamma \vdash \Pi x{:}A.B : \mathbf{s}}$

(λ) $\dfrac{\Gamma, x{:}A \vdash M : B \quad \Gamma \vdash \Pi x{:}A.B : \mathbf{s}}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}$

(app) $\dfrac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B[N/x]}$

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

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

- The contexts Σ_{PROP}, Σ_{PRED} and Σ_{lambda} are well-formed.
- The Π rule allows to form two forms of function types.

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

  – With $\mathbf{s} = \mathbf{type}$, we can form $\mathsf{D}{\rightarrow}\mathsf{D}$ and $\Pi x{:}\mathsf{D}.x = x$, etc.
  – With $\mathbf{s} = \mathbf{kind}$, we can form $\mathsf{D}{\rightarrow}\mathsf{D}{\rightarrow}\mathbf{type}$ and $\mathsf{prop}{\rightarrow}\mathbf{type}$.

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 $M \longrightarrow_{\beta\eta} N$, 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 $\lambda{\rightarrow}$.

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
- 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

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

for all Γ and M.

The type synthesis algorithm $\mathrm{Type}_{-}(-)$ is complete if

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

for all Γ, M and A.

$$
\begin{aligned}
\mathrm{Ok}(<>) &= \text{'true'} \\[1em]
\mathrm{Ok}(\Gamma, x{:}A) &= \mathrm{Type}_\Gamma(A) \in \{\mathbf{type}, \mathbf{kind}\}, \\[1em]
\mathrm{Type}_\Gamma(x) &= \text{if } \mathrm{Ok}(\Gamma) \text{ and } x{:}A \in \Gamma \text{ then } A \text{ else } \text{'false'}, \\[1em]
\mathrm{Type}_\Gamma(\mathbf{type}) &= \text{if } \mathrm{Ok}(\Gamma)\text{then } \mathbf{kind} \text{ else } \text{'false'}, \\[1em]
\mathrm{Type}_\Gamma(MN) &= \text{if } \mathrm{Type}_\Gamma(M) = C \text{ and } \mathrm{Type}_\Gamma(N) = D \\
&\ \text{then} \quad \text{if } C \twoheadrightarrow_\beta \Pi x{:}A.B \text{ and } A =_\beta D \\
&\ \qquad\qquad \text{then } B[N/x] \text{ else } \text{'false'} \\
&\ \text{else} \quad \text{'false'},
\end{aligned}
$$

$$
\begin{array}{rcl}
\mathrm{Type}_\Gamma(\lambda x{:}A.M) & = & \textsf{if } \mathrm{Type}_{\Gamma,x:A}(M) = B \\
 & & \textsf{then} \qquad\qquad \textsf{if } \mathrm{Type}_\Gamma(\Pi x{:}A.B) \in \{\mathbf{type}, \mathbf{kind}\} \\
 & & \qquad\qquad\qquad\; \textsf{then } \Pi x{:}A.B \textsf{ else } \text{`}\textsf{false}\text{'} \\
 & & \textsf{else } \text{`}\textsf{false}\text{'}, \\
\mathrm{Type}_\Gamma(\Pi x{:}A.B) & = & \textsf{if } \mathrm{Type}_\Gamma(A) = \mathbf{type} \textsf{ and } \mathrm{Type}_{\Gamma,x:A}(B) = s \\
 & & \textsf{then } s \textsf{ else } \text{`}\textsf{false}\text{'}
\end{array}
$$

Soundness

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

Completeness

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

This implies that, if $\mathrm{Type}_\Gamma(M) =$ 'false', then M is not typable in Γ.

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.
(Not guaranteed by soundness and completeness)

Interesting cases: λ-abstraction and application:

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

Replace the side condition

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

by

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

Termination

We want $\mathrm{Type}_{\_}(-)$ to terminate on all inputs.
(Not guaranteed by soundness and completeness)

Interesting cases: λ-abstraction and application:

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

For this case, termination follows from the decidability of equality on well-typed terms (using SN and CR).