Introduction to Type Theory

February 2008

Alpha Lernet Summer School

Piriapolis, Uruguay

Herman Geuvers

Nijmegen & Eindhoven, NL

Lecture 4: Dependent Type Theory, Logical Framework

Dependent Type Theory

We have seen "Dependent Types at Work" and we have seen the history and intuitions behind it.

Now: the rules

With dependent types:
- "everything depends on everything"
- we can't first define the types, then the terms
- two "universes": $\mathbf{type}$ and $\mathbf{kind}$

NB Peter Dybjer and Ana Bove used "Set" for what I call $\mathbf{type}$.

- $\mathbf{type}$ is the universe of types
- We can't have $\mathbf{type} : \mathbf{type}$, so we have another universe:

$\mathbf{type} : \mathbf{kind}$.

First order Dependent Type theory, 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

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

Auxiliary judgement

$$\Gamma \vdash$$

denoting that Γ is a correct context.

Derivation rules of λP

$\mathbf{s}$ 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) $\dfrac{\Gamma, x{:}A \vdash B : \mathbf{s} \;\; \Gamma \vdash A : \mathbf{type}}{\Gamma \vdash \Pi x{:}A.B : \mathbf{s}}$

(λ) $\dfrac{\Gamma, x{:}A \vdash M : B \;\; \Gamma \vdash \Pi x{:}A.B : \mathbf{s}}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}$ (app) $\dfrac{\Gamma \vdash M : \Pi x{:}A.B \;\; \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$

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

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

Representation of PRED (minimal predicate logic) into λP

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

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

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

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

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.

Shallow embedding of logic in type theory

For $\lambda\!\!\rightarrow$, $\lambda 2$ and λP we have seen

Direct representations (shallow embeddings) 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.

Deep embedding of logic in type theory

Second way of interpreting logic in type theory De Bruijn:

> Logical framework encoding or deep 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.

Shallow and Deep embedding

	proof	formula
shallow embedding	$\lambda x{:}A.x$	$A{\rightarrow}A$
deep embedding	imp_intr $A\,A\,\lambda x{:}T\,A.x$	$T(A \Rightarrow A)$

Needed:

$$
\begin{array}{rcl}
\Rightarrow & : & \mathsf{prop}{\rightarrow}\mathsf{prop}{\rightarrow}\mathsf{prop} \\
\mathsf{T} & : & \mathsf{prop}{\rightarrow}\mathbf{type} \\
\mathsf{imp\_intr} & : & (A, B : \mathsf{prop})(\mathsf{T}\,A \rightarrow \mathsf{T}\,B) \rightarrow \mathsf{T}(A \Rightarrow B) \\
\mathsf{imp\_el} & : & (A, B : \mathsf{prop})\mathsf{T}(A \Rightarrow B) \rightarrow \mathsf{T}\,A \rightarrow \mathsf{T}\,B.
\end{array}
$$

Close to a Gödel like encoding of predicate logic in $\mathbb{N}$:

Define a coding fuction $\lceil - \rceil$ for formulas and define $\mathsf{Bew}(\lceil \varphi \rceil, n)$

Shallow and Deep embedding

Direct representation	Deep 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 the logical framework LF – which is just λP.
- Exhibit the properties of LF that make this work.

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 $\{\Rightarrow, \forall\}$)

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:

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

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:

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

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

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

Encoding of derivations

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

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

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 $f : \mathsf{T}\,p \rightarrow \mathsf{T}\,q$ 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

Define

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

Desired properties of the encoding:

- Adequacy (soundness) of the encoding:

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

Signature:

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

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

Signature: Σ_{PRED}

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

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.

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

Intro and elim rules for $\forall$

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

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 $\forall$

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

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

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

Adequacy for minimal predicate logic

Again one can prove adequacy

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

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

Faithfulness can be proved as well.

The use of the Π-type

- The contexts Σ_{PROP} and Σ_{PRED} 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 $\mathsf{D}\to\mathsf{D}$ and $\Pi x{:}\mathsf{D}.x = x$, etc.
  - With $\mathbf{s} = \mathbf{kind}$, we can form $\mathsf{D}\to\mathsf{D}\to\mathbf{type}$ and $\mathsf{prop}\to\mathbf{type}$.

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

$$\Gamma \vdash M : \sigma? \quad \text{TCP}$$
$$\Gamma \vdash M : ? \quad \text{TSP}$$
$$\Gamma \vdash ? : \sigma \quad \text{TIP}$$

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

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

for all Γ and M.

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

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

for all Γ, M and A.

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

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

Soundness and Completeness

Soundness

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

Completeness

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

As a consequence:

$$\mathrm{Type}_\Gamma(M) = \text{‘false’} \;\Rightarrow\; M \text{ is not typable in } \Gamma$$

NB 1. Completeness implies that Type terminates on all well-typed terms. We want that Type terminates on all pseudo terms.

NB 2. 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.

Interesting cases: λ-abstraction and application:

$$\mathrm{Type}_\Gamma(\lambda x{:}A.M) \quad = \quad \text{if } \mathrm{Type}_{\Gamma,x:A}(M) = B$$

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

$$\text{then } \Pi x{:}A.B \text{ else 'false'}$$

$$\text{else 'false'},$$

! Recursive call is not on a smaller term!

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.
Interesting cases: λ-abstraction and application:

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

! Need to decide β-reduction and β-equality!

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