

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  $\lambda \rightarrow$  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
  - $\lambda$ -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  $\Gamma$ , 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

One type system : One logic

Logical rules  $\sim$  type theoretic rules

One type system : Many logics

Shallow encoding

Logical rules  $\sim$  context declarations

## Direct 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  $\{ \supset, A \}$ )
3. Untyped  **$\lambda$ -calculus**

## Minimal propositional logic

Fix the **signature** (context) of minimal propositional logic.

**prop** : type

**imp** : prop  $\rightarrow$  prop  $\rightarrow$  prop

Notation:

$A \supset B$  for **imp**  $AB$

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.

**prop** : type

**imp** : prop  $\rightarrow$  prop  $\rightarrow$  prop

Notation:

$A \supset B$  for **imp**  $AB$

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

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

**T** : prop  $\rightarrow$  type.

Intended meaning of **T**  $d$  is 'the type of proofs of  $d$ '.

We interpret ' $d$  is valid' by '**T**  $d$  is inhabited'.

To derive  $\top_d$  we also encode the **logical derivation rules**

$$\text{imp\_intr} : \Pi p, q : \text{prop.} (\top_d \leftarrow \top q) \rightarrow \top (d \supset q),$$

$$\text{imp\_el} : \Pi p, q : \text{prop.} \top (d \supset q) \rightarrow \top_d \leftarrow \top q.$$

New phenomenon:  **$\Pi$ -type**:

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

**imp\_intr** takes two (names of) **propositions**  $d$  and  $q$  and a term  $f : \top_d \leftarrow \top q$  and returns a term of type  $\top (d \supset q)$

Indeed  $A \supset A$ , becomes valid:

$$\text{imp\_intr } A A (\lambda x:T A. x) : \top (A \supset A)$$

Define

$\Sigma\text{PROP}$  to be the signature for minimal proposition logic,  $\text{PROP}$ .

Desired **properties** of the encoding:

- **Adequacy** (**soundness**) of the encoding:

$\vdash_{\text{PROP}} A \Leftrightarrow \Sigma\text{PROP}, a_1:\text{prop}, \dots, a_n:\text{prop} \vdash p : T A$  for some  $p$ .

$\{a_1, \dots, a_n\}$  is the set of proposition variables in  $A$ .  
Proof by induction on the derivation of  $\vdash_{\text{PROP}} A$ .

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

Minimal predicate logic over one domain  $A$  (just  $\subset$  and  $\forall$ )  
 Signature:

prop : type,

$A$  : type,

$c$  :  $A$ ,

$f$  :  $A \rightarrow A$ ,

$R$  :  $A \rightarrow A \rightarrow \text{prop}$ ,

$\subset$  :  $\text{prop} \rightarrow \text{prop} \rightarrow \text{prop}$ ,

imp\_intr :  $\Pi d, b : \text{prop}. (T d \rightarrow T b) \rightarrow T(d \subset b)$ ,

imp\_el :  $\Pi d, b : \text{prop}. T(d \subset b) \rightarrow T d \rightarrow T b$ .

Now encode  $\forall$ :

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

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

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

Signature:  $\Sigma_{\text{PRED}}$

prop : type,

$A$  : type,

⋮

$\supset$  : prop  $\rightarrow$  prop  $\rightarrow$  prop,

imp\_intr :  $\prod p, q : \text{prop} . \top \rightarrow p \rightarrow \top \rightarrow q$  ( $b \supset d$ ),

imp\_elim :  $\prod p, q : \text{prop} . \top \rightarrow p \rightarrow (b \supset d) \rightarrow \top \rightarrow q$ .

Now encode  $\forall$ :

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

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

Universal quantification is translated as follows.

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

Intro and elim rules for  $\forall$ :

$\text{forall} : (A \rightarrow \text{prop}) \rightarrow \text{prop},$   
 $\text{forall\_intr} : \Pi P : A \rightarrow \text{prop}. (\Pi x : A. T(Px)) \rightarrow T(\text{forall } P),$   
 $\text{forall\_elim} : \Pi P : A \rightarrow \text{prop}. T(\text{forall } P) \rightarrow \Pi x : A. T(Px).$

The proof of

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

is now mirrored by the proof-term

$\text{forall\_intr } [] ( \lambda z : A. \text{forall\_intr } [] [] ( \lambda h : T (\forall x, y : A. Rxy) .$   
 $\text{forall\_elim } [] [] (hz) ) )$

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

$\text{forall } (\lambda z : A. \text{forall } (\lambda x : A. (\text{forall } (\lambda y : A. Rxy) ) ) ) (Rzz)$

Again one can prove **adequacy**

$\vdash_{\text{PRET}} \varphi \Leftrightarrow \Sigma_{\text{PRET}, x_1:A, \dots, x_n:A} \vdash p : T\varphi$ , for some  $p$ ,  
where  $\{x_1, \dots, x_n\}$  is the set of free variables in  $\varphi$ .

**Faithfulness** can be proved as well.

## Untyped $\lambda$ -calculus Signature $\Sigma_{\text{lambda}}$ :

$D$  : type;  
 app :  $D \rightarrow (D \rightarrow D)$ ;  
 abs :  $(D \rightarrow D) \rightarrow D$ .

**Encoding** of  $\lambda$ -terms as terms of type  $D$ .

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

$[x] = x$ ;  
 $[PQ] = \text{app } [P] [Q]$ ;  
 $[\lambda x.P] = \text{abs } (\lambda x:D.[P])$ .

**Examples:**  $[\lambda x.x] = \text{abs } (\lambda x:D.\text{app } x x)$   
 $[(\lambda x.x)(\lambda y.y)] = \text{app } (\text{abs } (\lambda x:D.\text{app } x x)) (\text{abs } (\lambda y:D.y))$ .

Introducing  $\beta$ -equality in  $\Sigma^{\text{lambda}}$  :

$\text{eq} : D \rightarrow D \rightarrow \text{type}$ .

Notation  $P = Q$  for  $\text{eq } P \ Q$ .

**Rules** for proving equalities.

**refl** :  $\Pi x : D. x = x$ ,

**sym** :  $\Pi x, y : D. x = y \rightarrow y = x$ ,

**trans** :  $\Pi x, y, z : D. x = y \rightarrow y = z \rightarrow x = z$ ,

**mon** :  $\Pi x, x', z, z' : D. x = x' \rightarrow z = z' \rightarrow \text{app } z' \ (\text{app } x \ d) = \text{app } z' \ (\text{app } x' \ d)$ ,

**xi** :  $\Pi f, g : D \rightarrow D. \Pi x : D. (f \ x) = (g \ x) \rightarrow \text{abs } f = \text{abs } g$ ,

**beta** :  $\Pi f : D \rightarrow D. \Pi x : D. \text{app} (\text{abs } f) \ x = (f \ x)$ .

**Adequacy:**

$$P =_{\beta} Q \Leftrightarrow \Sigma^{\text{lambda}, x_1:D, \dots, x_n:D} p \vdash P = [Q], \text{ for some } p.$$

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

**Faithfulness** also holds.

Logical Framework, LF, or  $\lambda P$   
Derive judgements of the form

$$\Gamma \vdash M : B$$

- $\Gamma$  is a context

- $M$  and  $B$  are terms

taken from the set of pseudoterms

$$T ::= \text{Var} \mid \text{type} \mid \text{kind} \mid TT \mid \lambda x:T.T \mid \Pi x:T.T, ,$$

Auxiliary judgement

$$\Gamma \vdash$$

denoting that  $\Gamma$  is a correct context.

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

$$\frac{\Gamma \vdash A : s \quad \text{(base)}}{\emptyset \vdash} \quad \text{(ctxt)} \quad \frac{\Gamma, x:A \vdash}{\Gamma \vdash A : s} \quad \text{if } x \text{ not in } \Gamma \quad \text{(ax)} \quad \frac{\Gamma \vdash \text{type} : \text{kind}}{\Gamma \vdash}$$

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

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

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

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

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

- The contexts  $\Sigma_{\text{PROP}}$ ,  $\Sigma_{\text{PRED}}$  and  $\Sigma_{\text{lambda}}$  are well-formed.
- The  $\Pi$  rule allows to form two forms of function types.

$$(II) \frac{\Gamma \vdash A \vdash B : s \quad \Gamma \vdash A : \text{type}}{\Gamma, x:A \vdash B : s}$$

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

## Properties of $\lambda P$ .

- **Uniqueness of types**  
If  $\Gamma \vdash M : \sigma$  and  $\Gamma \vdash M : \tau$ , then  $\sigma = \tau$ .

- **Subject Reduction**  
If  $\Gamma \vdash M : \sigma$  and  $M \rightarrow^{\beta\eta} N$ , 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  $\lambda P$  to  $\lambda \rightarrow$ .

## Decidability Questions:

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

For  $\lambda P$ :

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

## Type Checking

Define algorithms  $\text{Ok}(-)$  and  $\text{Type}(-)$  simultaneously:

- $\text{Ok}(-)$  takes a **context** and returns 'true' or 'false'
- $\text{Type}(-)$  takes a **context** and a **term** and returns a **term** or 'false'.

The type synthesis algorithm  $\text{Type}(-)$  is **sound** if

$$\text{Type}(M) = A \Leftrightarrow \Gamma \vdash M : A$$

for all  $\Gamma$  and  $M$ .

The type synthesis algorithm  $\text{Type}(-)$  is **complete** if

$$\Gamma \vdash M : A \Leftrightarrow \text{Type}(M) =_{\beta\eta} A$$

for all  $\Gamma$ ,  $M$  and  $A$ .

$\text{Ok}(\langle \rangle) = \text{'true'}$

$\text{Ok}(\Gamma, x:A) = \text{Type}_{\Gamma}(A) \in \{\text{type}, \text{kind}\}$ ,

$\text{Type}_{\Gamma}(x) = \text{if } \text{Ok}(\Gamma) \text{ and } x:A \in \Gamma \text{ then } A \text{ else 'false'}$ ,

$\text{Type}_{\Gamma}(\text{type}) = \text{if } \text{Ok}(\Gamma) \text{ then kind else 'false'}$ ,

$\text{Type}_{\Gamma}(MN) = \text{if } \text{Type}_{\Gamma}(M) = C \text{ and } \text{Type}_{\Gamma}(N) = D$   
then if  $C \rightarrow_{\beta} \Pi x:A.B$  and  $A =_{\beta} D$   
then  $B[N/x]$  else 'false',  
else 'false',

$\text{Type}_{\Gamma}(\lambda x:A.M) =$  if  $\text{Type}_{\Gamma, x:A}(M) = B$  then  
 if  $\text{Type}_{\Gamma}(\Pi x:A.B) \in \{\text{type}, \text{kind}\}$   
 then  $\Pi x:A.B$  else 'false'  
 else 'false',  
 $\text{Type}_{\Gamma}(\Pi x:A.B) =$  if  $\text{Type}_{\Gamma}(A) = \text{type}$  and  $\text{Type}_{\Gamma, x:A}(B) = s$   
 then  $s$  else 'false'

Soundness

$$\text{Type}_{\mathcal{T}}(M) = A \Leftrightarrow \Gamma \vdash M : A$$

Completeness

$$\Gamma \vdash M : A \Leftrightarrow \text{Type}_{\mathcal{T}}(M) =_{\beta\eta} A$$

This implies that, if  $\text{Type}_{\mathcal{T}}(M) = \text{'false'}$ , then  $M$  is not typable in  $\mathcal{T}$ .

Completeness only makes sense if we have **uniqueness of types** (Otherwise: let  $\text{Type}_{\mathcal{T}}(-)$  generate a **set of possible types**)

## Termination

We want  $\text{Type}(-)$  to **terminate** on all inputs.

(Not guaranteed by **soundness** and **completeness**)

Interesting cases:  $\lambda$ -abstraction and application:

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

then

**if**  $\text{Type}(A.B) \in \{\text{type}, \text{kind}\}$

then  $\text{Type}(A.B)$  else 'false'

else 'false',

Replace the side condition

**if**  $\text{Type}(A.B) \in \{\text{type}, \text{kind}\}$

by

**if**  $\text{Type}(A) \in \{\text{type}\}$

## Termination

We want  $\text{Type}(-)$  to **terminate** on all inputs.

(Not guaranteed by **soundness** and **completeness**)

Interesting cases:  $\lambda$ -abstraction and application:

$\text{Type}_{\text{PT}}(MN) =$  if  $\text{Type}_{\text{PT}}(M) = C$  and  $\text{Type}_{\text{PT}}(N) = D$   
then if  $C \twoheadrightarrow_{\beta} \Pi x:A.B$  and  $A =_{\beta} D$   
then  $B[N/x]$  else 'false'  
else 'false',

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