

Type Theory

Lecture 2: Dependent Types

Andreas Abel

Department of Computer Science and Engineering
Chalmers and Gothenburg University

Type Theory – Course CM0859 (2017-1)
Universidad EAFIT, Medellin, Colombia
6-10 March 2017

Contents

- 1 Typed Predicate Logic
 - Formation rules
 - Proof rules
- 2 Dependent Type Theory
 - Expressions and judgements
 - Dependent function type
- 3 The Logical Framework
 - A type of types
 - Type conversion

Typed Predicate Logic

- Propositional logic: only atomic statements like *Socrates is a human*.
- Predicate logic gives a finer structure by decomposing this into a predicate *is a human* applied to an individual *Socrates*
- and can express universal statements such as *all humans are mortal*.
- Untyped vs. typed predicate logic:

untyped: $\forall x. \text{Human}(x) \Rightarrow \text{Mortal}(x)$

typed: $\forall x:\text{Human}. \text{Mortal}^{\text{Human}} x$

- Untyped = untyped: a single type for all individuals/objects.
- Typed: objects are *a priori* sorted into different types.

Typed Predicate Logic (ctd.)

- $\text{Mortal}^{\text{Human}}$ is a predicate that only applies to objects of type **Human**.
- $\forall y:\text{Dog}. \text{Mortal}^{\text{Human}} y$ is an *ill-formed* proposition.
- What is a type, what a predicate is up to design.

$$\forall z:\text{LifeForm}. \text{Human}^{\text{LifeForm}} z \Rightarrow \text{Mortal}^{\text{LifeForm}} z$$

- Spoiler: Type theory will give us means to turn predicate into types:

$$\text{HumanLifeForm} = \Sigma z:\text{LifeForm}. \text{Human}^{\text{LifeForm}} z$$

Formulæ of typed predicate logic

- We extend the grammar for propositions:

$$\begin{array}{l}
 A, B, C ::= A \Rightarrow B \mid A \wedge B \mid A \vee B \mid \top \mid \perp \\
 \quad \mid P^T t \\
 \quad \mid \forall x:T. A \\
 \quad \mid \exists x:T. A
 \end{array}
 \begin{array}{l}
 \text{propositional connectives} \\
 \text{atoms} \\
 \text{universal quantification} \\
 \text{existential quantification}
 \end{array}$$

- Typing context Δ maps variables x to types T .
- Judgement $\Delta \vdash A \text{ prop}$ characterizes well-formed propositions.

Formation rules

- Atoms:

$$\frac{\Delta \vdash t : T}{\Delta \vdash P^T t \text{ prop}} \text{ atomF}$$

- Quantifiers:

$$\frac{\Delta, x:T \vdash A \text{ prop}}{\Delta \vdash \forall x:T. A \text{ prop}} \forall F \qquad \frac{\Delta, x:T \vdash A \text{ prop}}{\Delta \vdash \exists x:T. A \text{ prop}} \exists F$$

- Propositional connectives:

$$\frac{\Delta \vdash A \text{ prop} \quad \Delta \vdash B \text{ prop}}{\Delta \vdash A * B \text{ prop}} *F \quad (* \in \{\Rightarrow, \wedge, \vee\})$$

$$\frac{}{\Delta \vdash \top \text{ prop}} \top F \qquad \frac{}{\Delta \vdash \perp \text{ prop}} \perp F$$

Examples

- Well-formed formulæ:
 - $\forall x:\mathbb{N}. \forall y:\mathbb{N}. (\leq^{\mathbb{N} \times \mathbb{N}} \langle x, y \rangle) \vee (\leq^{\mathbb{N} \times \mathbb{N}} \langle y, x \rangle)$
 - $\forall f:\text{Bool} \rightarrow \text{Bool}. \forall x:\text{Bool}. \equiv^{\text{Bool} \times \text{Bool}} \langle f (f (f x)), f x \rangle$
- Ill-formed formulæ:
 - $\not\vdash \forall f:\mathbb{N} \rightarrow \mathbb{N}. \exists r:\mathbb{R}. \text{Even}^{\mathbb{N}}(f r) \text{ prop}$
 - $\not\vdash \exists x:\mathbb{N}. \leq^{\mathbb{N} \times \mathbb{N}} \langle x, y \rangle \text{ prop}$

Proof rules

- Judgement $\Gamma \vdash_{\Delta} A \text{ true}$. (Both Γ and A are scoped in Δ .)
- Universal quantification:

$$\frac{\Gamma \vdash_{\Delta, x:T} A \text{ true}}{\Gamma \vdash_{\Delta} \forall x:T. A \text{ true}} \forall I \qquad \frac{\Gamma \vdash_{\Delta} \forall x:T. A \text{ true} \quad \Delta \vdash t : T}{\Gamma \vdash_{\Delta} A[t/x] \text{ true}} \forall E$$

- Existential quantification:

$$\frac{\Delta \vdash t : T \quad \Gamma \vdash_{\Delta} A[t/x] \text{ true}}{\Gamma \vdash_{\Delta} \exists x:T. A \text{ true}} \exists I$$

$$\frac{\Gamma \vdash_{\Delta} \exists x:T. A \text{ true} \quad \Gamma, A \text{ true} \vdash_{\Delta, x:T} C \text{ true}}{\Gamma \vdash_{\Delta} C \text{ true}} \exists E$$

Proof examples

- Use of \exists :

$$\frac{\vdash 0 : \mathbb{N} \quad \vdash \text{Even}^{\mathbb{N}} 0 \text{ true}}{\vdash \exists x : \mathbb{N}. \text{Even}^{\mathbb{N}} x} \exists\text{I}$$

- $\Gamma_1 := (\exists x : S. \top \text{ true}, \forall y : S. A \text{ true})$. Prove $\Gamma_1 \vdash \exists z : S. A \text{ true}$!
- Exercise: Show $A \text{ true}, \exists x : T. B \text{ true} \vdash \exists x : T. A \wedge B \text{ true}$!
- Exercise: Show $\forall x : T. A \Rightarrow B \text{ true}, A \text{ true} \vdash \forall x : T. B \text{ true}$!
(Here, A does not depend on x .)

Proof terms

- Judgement $\Gamma \vdash_{\Delta} M : A$. (All of Γ , M , and A are scoped in Δ .)
- Universal quantification:

$$\frac{\Gamma \vdash_{\Delta, x:T} M : A}{\Gamma \vdash_{\Delta} \lambda x. M : \forall x:T. A} \forall I \qquad \frac{\Gamma \vdash_{\Delta} M : \forall x:T. A \quad \Delta \vdash t : T}{\Gamma \vdash_{\Delta} M t : A[t/x]} \forall E$$

- Existential quantification:

$$\frac{\Delta \vdash t : T \quad \Gamma \vdash_{\Delta} M : A[t/x]}{\Gamma \vdash_{\Delta} \langle t, M \rangle : \exists x:T. A} \exists I$$

$$\frac{\Gamma \vdash_{\Delta} M : \exists x:T. A \quad \Gamma, y:A \vdash_{\Delta, x:T} N : C}{\Gamma \vdash_{\Delta} \text{let } \langle x, y \rangle = M \text{ in } N : C} \exists E$$

- New reduction:

$$\text{let } \langle x, y \rangle = \langle t, M \rangle \text{ in } N \longrightarrow_{\beta} N[t/x, M/y]$$

Strong existentials

- Proof terms would allow us to extract the witness!

$$\frac{\Gamma \vdash_{\Delta} M : \exists x:T. A}{\Delta \vdash \text{fst } M : T} \exists E_1$$

$$\frac{\Gamma \vdash_{\Delta} M : \exists x:T. A}{\Gamma \vdash_{\Delta} \text{snd } M : A[\text{fst } M/x]} \exists E_2$$

- However, this would make proving and programming (typing) interdependent.
- Why not? ;-)

Dependent Type Theory

- Interpret propositions as “sets” of their proofs.
- Rather:

$$\begin{array}{lcl} \text{proposition} & = & \text{type} \\ \text{proof of proposition} & = & \text{inhabitant of type} \end{array}$$

- Abolish “set” as a primitive notion.
- Instead: **types** and **predicates**. Example:
 - set of natural numbers \rightarrow type of natural numbers \mathbb{N}
 - set of primes \rightarrow predicate **Prime** on \mathbb{N}
- Unify types T and propositions A .
- Unify programs/objects t and proof terms M .

Expressions of Dependent Type Theory

- We no longer distinguish between terms and types *a priori*.
- There is a single grammar of *expressions*.

$A, B, C,$

$M, N, O ::= c$

constants

$| x | \lambda x.M | M N$

lambda-calculus

$| (x : A) \rightarrow B$

dependent function type

- Expressions are sorted into terms and types by judgements:

$\Gamma \vdash A \text{ type}$ *in context Γ , expression A is a well-formed type*

$\Gamma \vdash M : A$ *in context Γ , expression M has type A*

Dependent function type

- Formation.

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma, x:A \vdash B \text{ type}}{\Gamma \vdash (x : A) \rightarrow B \text{ type}} \quad \Pi F$$

- Introduction.

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

- Elimination.

$$\frac{\Gamma \vdash M : (x : A) \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]} \quad \Pi E$$

Dependent function type: examples

- $(x : \mathbb{N}) \rightarrow \mathbb{N}$: non-dependent function type $\mathbb{N} \rightarrow \mathbb{N}$
- $(x : \mathbb{N}) \rightarrow \text{Vec } A \ n$: properly dependent function type
- $(p : (x \geq 2)) \rightarrow x^2 > x$: implication $x \geq 2 \Rightarrow x^2 > x$
- $(x : \mathbb{N}) \rightarrow x \geq 0$: universal quantification $\forall x:\mathbb{N}. x \geq 0$.
- $(p : (x > 0)) \rightarrow \mathbb{N}$: conditional value.

$$\text{div} : (x : \mathbb{N}) \rightarrow (y : \mathbb{N}) \rightarrow (p : (y > 0)) \rightarrow \mathbb{N}$$

Function type interpretations

- Non-dependent function type $A \rightarrow B$.

	B prop	B type
A prop	implication $A \Rightarrow B$	conditional value
A type	void universal quant. $\forall _ : A. B$	function

- Dependent-function type $(x : A) \rightarrow B$.

	B prop	B type
A prop	proof-relevant implication	proof-rel. cond. value
A type	universal quant. $\forall x:A. B$	dependent function

The Logical Framework

- The Logical Framework (LF) is a minimal dependently typed lambda calculus.
- It is used to represent programming languages and logics.
- A mature implementation is Twelf (Pfenning, Schürmann).

The Logical Framework: representing trees

- In LF, abstract syntax (e. g., unary numbers) can be represented by
 - adding new type constant(s), e. g.,

$$\mathbb{N} \text{ type}$$

- adding new term constants targeting the new type constant(s), e. g.,

$$\begin{aligned} \text{zero} & : \mathbb{N} \\ \text{suc} & : \mathbb{N} \rightarrow \mathbb{N} \end{aligned}$$

- Grammar of first order terms $f(\vec{t})$:
Three new types **Symbol**, **Tm**, **ArgList**, and constructors

$$\begin{aligned} \text{app} & : \text{Symbol} \rightarrow \text{ArgList} \rightarrow \text{Tm} \\ \text{nil} & : \text{ArgList} \\ \text{cons} & : \text{Tm} \rightarrow \text{ArgList} \rightarrow \text{ArgList} \end{aligned}$$

The Logical Framework: representing binders

- Trees with binders are represented by LF binders (HOAS).
- E. g., first-order logical formulæ: New type `Form` *type*, plus

$$\begin{array}{ll}
 \text{Top, Bot} & : \text{Form} \\
 \text{And, Or, Imp} & : \text{Form} \rightarrow \text{Form} \rightarrow \text{Form} \\
 \text{Equal} & : \text{Tm} \rightarrow \text{Tm} \rightarrow \text{Form} \\
 \text{Forall, Exists} & : (\text{Tm} \rightarrow \text{Form}) \rightarrow \text{Form}
 \end{array}$$

- Given a symbol $f : \text{Symbol}$, we represent the formula $\exists x. x \equiv f(x)$ by the expression:

$$\text{Exists } \lambda x. \text{Equal } x \text{ (app } f \text{ (cons } x \text{ nil))}$$

- So far, we have not used dependent types.

The Logical Framework: representing predicates

- Besides trees, a specification language need predicates/relations.
- How to represent, e. g., the predicate `Even` on \mathbb{N} ?
- `Even n` should be a type that is inhabited iff n is even.
- `Even` is a function from \mathbb{N} to types.
- With a constant `type`, a type of types, we can write

`Even` : $\mathbb{N} \rightarrow \text{type}$

`ezero` : `Even zero`

`esuc` : $(x : \mathbb{N}) \rightarrow \text{Even } x \rightarrow \text{Even } (\text{suc } (\text{suc } x))$

- What should be the rules for `type`?

Is type a type?

- We drop judgement A *type* in favor of $A : \text{type}$.
- Tentative rules for *type*:

$$\frac{}{\Gamma \vdash \text{type} : \text{type}} \text{typeF}$$

- This rule is **inconsistent!** Girard's paradox [2].
- We introduce a universe *kind* inhabited by *type*.

$$\frac{}{\Gamma \vdash \text{type} : \text{kind}} \text{typeF} \quad \frac{\Gamma \vdash A : \text{type} \quad \Gamma, x:A \vdash B : \text{kind}}{\Gamma \vdash (x : A) \rightarrow B : \text{kind}} \Pi F'$$

- The second rule allows us to form types of predicates like $\mathbb{N} \rightarrow \text{type}$.
- “Predicate” is an interpretation, the technical term is *type family*.

LF example: representing derivations

- LF allows us to represent *judgements as types*.
- Provability of a FO formula can be stated with type family

$$\text{Prf} : \text{Form} \rightarrow \text{type}$$

$\text{Prf } A$ is the type of proofs of formula A .

- Proof rules for conjunction:

$$\text{prfAndI} \quad : (a : \text{Form}) \rightarrow (b : \text{Form}) \rightarrow \text{Prf } a \rightarrow \text{Prf } b \rightarrow \text{Prf } (\text{And } a \ b)$$
$$\text{prfAndEL} \quad : (a : \text{Form}) \rightarrow (b : \text{Form}) \rightarrow \text{Prf } (\text{And } a \ b) \rightarrow \text{Prf } a$$
$$\text{prfAndER} \quad : (a : \text{Form}) \rightarrow (b : \text{Form}) \rightarrow \text{Prf } (\text{And } a \ b) \rightarrow \text{Prf } b$$

LF example: representing derivations

- Proof rules for implication:

$$\text{prfImpl} : (a\ b : \text{Form}) \rightarrow (\text{Prf } a \rightarrow \text{Prf } b) \rightarrow \text{Prf } (\text{Imp } a\ b)$$
$$\text{prfImpE} : (a\ b : \text{Form}) \rightarrow \text{Prf } (\text{Imp } a\ b) \rightarrow \text{Prf } a \rightarrow \text{Prf } b$$

- Proof rules for universal quantification:

$$\text{prfAllI} : (p : \text{Tm} \rightarrow \text{Form}) \rightarrow ((x : \text{Tm}) \rightarrow \text{Prf } (p\ x)) \rightarrow \text{Prf } (\text{Forall } p)$$
$$\text{prfAllE} : (p : \text{Tm} \rightarrow \text{Form}) \rightarrow \text{Prf } (\text{Forall } p) \rightarrow (t : \text{Tm}) \rightarrow \text{Prf } (p\ t)$$

The need for type conversion

$\text{prfAllE} : (p : \text{Tm} \rightarrow \text{Form}) \rightarrow \text{Prf} (\text{Forall } p) \rightarrow (t : \text{Tm}) \rightarrow \text{Prf} (p \ t)$

- We would expect

$$\text{prfAllE} (\lambda x. \text{Equal } x \ x) \ q \ t : \text{Prf} (\text{Equal } t \ t).$$

- By ΠE , we only get

$$\text{prfAllE} (\lambda x. \text{Equal } x \ x) \ q \ t : \text{Prf} ((\lambda x. \text{Equal } x \ x) \ t)$$

- The types are *β -convertible*.

Completing LF's rules

- Grammar for sorts $s ::= \text{type} \mid \text{kind}$
- This allows us to unify the Π -formation rules:

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

- We close with the infamous type conversion rule:

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A = B : s}{\Gamma \vdash M : B} \text{conv}$$

- For now, judgement $\Gamma \vdash A = B : s$ is defined as

$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : s \quad A =_{\beta\eta} B}{\Gamma \vdash A = B : s}$$

with $=_{\beta\eta}$ the least congruence over β -reduction and η -expansion.

Summary: LF rules

$$\frac{(x:A) \in \Gamma}{\Gamma \vdash x : A} \text{ hyp} \quad \frac{}{\Gamma \vdash \text{type} : \text{kind}} \text{ typeF}$$

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

$$\frac{\Gamma, x:A \vdash M : B}{\Gamma \vdash \lambda x.M : (x : A) \rightarrow B} \text{ PI} \quad \frac{\Gamma \vdash M : (x : A) \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]} \text{ PE}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A = B : s}{\Gamma \vdash M : B} \text{ conv}$$

LF: discussion

- LF allows HOAS (Higher Order Abstract Syntax) encodings of programming languages and logics with binders.
- Adequacy of encodings relies on the parametricity of λ -abstraction.
- In particular, no case distinction. The term

$$\text{Forall } (\lambda x. \text{if } x \text{ then } A \wedge B \text{ else } A \vee B)$$

does not correspond to a formula of predicate logic.

- LF admits *type erasure*: Each well-typed term also has a simple type.
- Normalization can be proven by erasure to simply-typed λ -calculus.

$$\begin{aligned} \llbracket (x : A) \rightarrow B \rrbracket &= \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket \\ \llbracket c M_1 \dots M_n \rrbracket &= c \end{aligned}$$

References



Robert Harper, Furio Honsell, and Gordon Plotkin.

A framework for defining logics.

JACM, 40(1):143–184, 1993.



Antonius J. C. Hurkens.

A simplification of Girard's paradox.

In *TLCA'95*, volume 902 of *LNCS*, pages 266–278. Springer, 1995.



Frank Pfenning.

Logical frameworks.

In *Handbook of Automated Reasoning, volume 2*, chapter 17, pages 1063–1147. Elsevier and MIT Press, 2001.