Type Theory

Lecture 2: Dependent Types

Andreas Abel

Department of Computer Science and Engineering
Chalmers and Gothenburg University

ESSLLI 2016
28th European Summer School in Logic, Language, and Information
unibz, Bozen/Bolzano, Italy
15-19 August 2016

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:

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

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

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

Typed Predicate Logic (ctd.)

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

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

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

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

Formulæ of typed predicate logic

- We extend the grammar for propositions:

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

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

Formation rules

- Atoms:

$$\frac{\Delta \vdash t : T}{\Delta \vdash P^T\, t\ \mathit{prop}}\ \mathrm{atomF}$$

- Quantifiers:

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

- Propositional connectives:

$$\frac{\Delta \vdash A\ \mathit{prop} \qquad \Delta \vdash B\ \mathit{prop}}{\Delta \vdash A \star B\ \mathit{prop}}\ \star\mathrm{F}\ (\star \in \{\Rightarrow, \wedge, \vee\})$$

$$\frac{}{\Delta \vdash \top\ \mathit{prop}}\ \top\mathrm{F} \qquad \frac{}{\Delta \vdash \bot\ \mathit{prop}}\ \bot\mathrm{F}$$

Examples

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

Proof rules

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

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

- Existential quantification:

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

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

Proof examples

- Use of $\exists$I:

$$\frac{\vdash 0 : \mathbb{N} \qquad \vdash \mathsf{Even}^{\mathbb{N}} 0\ \mathit{true}}{\vdash \exists x{:}\mathbb{N}.\ \mathsf{Even}^{\mathbb{N}} x}\ \exists\mathsf{I}$$

- $\Gamma_1 := (\exists x{:}S.\ \top\ \mathit{true},\ \forall y{:}S.\ A\ \mathit{true})$. Prove $\Gamma_1 \vdash \exists z{:}S.\ A\ \mathit{true}$!
- Exercise: Show $A\ \mathit{true},\ \exists x{:}T.\ B\ \mathit{true} \vdash \exists x{:}T.\ A \wedge B\ \mathit{true}$!
- Exercise: Show $\forall x{:}T.\ A \Rightarrow B\ \mathit{true},\ A\ \mathit{true} \vdash \forall x{:}T.\ B\ \mathit{true}$! (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:

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

- Existential quantification:

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

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

- New reduction:

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

Strong existentials

- Proof terms would allow us to extract the witness!

$$\frac{\Gamma \vdash_\Delta M : \exists x{:}T.\ A}{\Delta \vdash \mathsf{fst}\, M : T}\ \exists E_1 \qquad \frac{\Gamma \vdash_\Delta M : \exists x{:}T.\ A}{\Gamma \vdash_\Delta \mathsf{snd}\, M : A[\mathsf{fst}\, M/x]}\ \exists E_2$$

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

Dependent Type Theory

- Interpret propositions as "sets" of their proofs.
- Rather:

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

- 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*.

$$
\begin{array}{lll}
A, B, C, & & \\
M, N, O ::= & c & \text{constants} \\
& \mid x \mid \lambda x.M \mid M\,N & \text{lambda-calculus} \\
& \mid (x : A) \to B & \text{dependent function type}
\end{array}
$$

- Expressions are sorted into terms and types by judgements:

$\Gamma \vdash A\ \mathit{type}$ *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.

$$\frac{\Gamma \vdash A\ type \qquad \Gamma, x{:}A \vdash B\ type}{\Gamma \vdash (x : A) \to B\ type}\ \Pi\mathsf{F}$$

- Introduction.

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

- Elimination.

$$\frac{\Gamma \vdash M : (x : A) \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B[N/x]}\ \Pi\mathsf{E}$$

Dependent function type: examples

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

$$\mathsf{div} : (x : \mathbb{N}) \to (y : \mathbb{N}) \to (p : (y > 0)) \to \mathbb{N}$$

Function type interpretations

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

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

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

	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.,

$$\mathbb{N}\ \mathit{type}$$

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

$$\begin{array}{lcl} \mathsf{zero} & : & \mathbb{N} \\ \mathsf{suc} & : & \mathbb{N} \to \mathbb{N} \end{array}$$

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

$$\begin{array}{lcl} \mathsf{app} & : & \mathsf{Symbol} \to \mathsf{ArgList} \to \mathsf{Tm} \\ \mathsf{nil} & : & \mathsf{ArgList} \\ \mathsf{cons} & : & \mathsf{Tm} \to \mathsf{ArgList} \to \mathsf{ArgList} \end{array}$$

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

$$\begin{array}{lcl} \mathsf{Top},\ \mathsf{Bot} & : & \mathsf{Form} \\ \mathsf{And},\ \mathsf{Or},\ \mathsf{Imp} & : & \mathsf{Form} \to \mathsf{Form} \to \mathsf{Form} \\ \mathsf{Equal} & : & \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Form} \\ \mathsf{Forall},\ \mathsf{Exists} & : & (\mathsf{Tm} \to \mathsf{Form}) \to \mathsf{Form} \end{array}$$

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

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

- 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

$$\begin{array}{lcl} \mathsf{Even} & : & \mathbb{N} \to \mathsf{type} \\ \mathsf{ezero} & : & \mathsf{Even}\ \mathsf{zero} \\ \mathsf{esuc} & : & (x : \mathbb{N}) \to \mathsf{Even}\ x \to \mathsf{Even}\ (\mathsf{suc}\,(\mathsf{suc}\,x)) \end{array}$$

- What should be the rules for type?

Is type a type?

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

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

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

$$\frac{}{\Gamma \vdash \mathsf{type} : \mathsf{kind}}\ \mathsf{typeF} \qquad \frac{\Gamma \vdash A : \mathsf{type} \qquad \Gamma, x{:}A \vdash B : \mathsf{kind}}{\Gamma \vdash (x : A) \to B : \mathsf{kind}}\ \Pi\mathsf{F}'$$

- The second rule allows us to form types of predicates like $\mathbb{N} \to \mathsf{type}$.
- "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

$$\mathsf{Prf} : \mathsf{Form} \to \mathsf{type}$$

  $\mathsf{Prf}\ A$ is the type of proofs of formula A.
- Proof rules for conjunction:

$$\begin{array}{ll}
\mathsf{prfAndI} & : (a : \mathsf{Form}) \to (b : \mathsf{Form}) \to \mathsf{Prf}\ a \to \mathsf{Prf}\ b \to \mathsf{Prf}\ (\mathsf{And}\ a\ b) \\
\mathsf{prfAndEL} & : (a : \mathsf{Form}) \to (b : \mathsf{Form}) \to \mathsf{Prf}\ (\mathsf{And}\ a\ b) \to \mathsf{Prf}\ a \\
\mathsf{prfAndER} & : (a : \mathsf{Form}) \to (b : \mathsf{Form}) \to \mathsf{Prf}\ (\mathsf{And}\ a\ b) \to \mathsf{Prf}\ b
\end{array}$$

LF example: representing derivations

- Proof rules for implication:

$$\begin{aligned}\mathsf{prfImpI} &: (a\ b : \mathsf{Form}) \to (\mathsf{Prf}\ a \to \mathsf{Prf}\ b) \to \mathit{Prf}\ (\mathsf{Imp}\ a\ b)\\ \mathsf{prfImpE} &: (a\ b : \mathsf{Form}) \to \mathsf{Prf}\ (\mathsf{Imp}\ a\ b) \to \mathsf{Prf}\ a \to \mathsf{Prf}\ b\end{aligned}$$

- Proof rules for universal quantification:

$$\begin{aligned}\mathsf{prfAllI} &: (p : \mathsf{Tm} \to \mathsf{Form}) \to ((x : \mathsf{Tm}) \to \mathsf{Prf}\ (p\ x)) \to \mathsf{Prf}\ (\mathsf{Forall}\ p)\\ \mathsf{prfAllE} &: (p : \mathsf{Tm} \to \mathsf{Form}) \to \mathsf{Prf}\ (\mathsf{Forall}\ p) \to (t : \mathsf{Tm}) \to \mathsf{Prf}\ (p\ t)\end{aligned}$$

The need for type conversion

$$\mathsf{prfAllE} \;:\; (p : \mathsf{Tm} \to \mathsf{Form}) \to \mathsf{Prf}\,(\mathsf{Forall}\,p) \to (t : \mathsf{Tm}) \to \mathsf{Prf}\,(p\,t)$$

- We would expect

$$\mathsf{prfAllE}\,(\lambda t.\,\mathsf{Equal}\,t\,t)\,u \;:\; \mathsf{Prf}\,(\mathsf{Equal}\,u\,u).$$

- By ΠE, we only get

$$\mathsf{prfAllE}\,(\lambda t.\,\mathsf{Equal}\,t\,t)\,u \;:\; \mathsf{Prf}\,((\lambda t.\,\mathsf{Equal}\,t\,t)\,u)$$

- 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:

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

- We close with the infamous type conversion rule:

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

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

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

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

Summary: LF rules

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

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

$$\frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x.M : (x : A) \to B}\ \Pi\text{I} \qquad \frac{\Gamma \vdash M : (x : A) \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B[N/x]}\ \Pi\text{E}$$

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

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.

$$\begin{array}{lcl} [\![(x : A) \to B]\!] & = & [\![A]\!] \to [\![B]\!] \\ [\![c\, M_1 \ldots M_n]\!] & = & c \end{array}$$

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.