Type Theory

Lecture 3: Martin Löf Type Theory

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

Full Dependent Types

- LF's dependent types are refinements of simple types.
- Martin-Löf Type Theory has full-fledged dependent types.
- In particular, a type can be defined by case distinction and recursion on a value.

$$\begin{array}{lcl} \mathbb{R}^{(\_)} & : & \mathbb{N} \to \mathsf{type} \\ \mathbb{R}^{0} & = & \top \\ \mathbb{R}^{n+1} & = & \mathbb{R} \times \mathbb{R}^{n} \end{array}$$

  There is no erasure of $\mathbb{R}^n$ to a simple type.
- Dependent types are sometimes used in linear algebra:

$$\begin{array}{lcl} \mathsf{inner} & : & (n : \mathbb{N}) \to \mathbb{R}^{n} \times \mathbb{R}^{n} \to \mathbb{R} \\ \mathsf{mmult} & : & (n\ m\ l : \mathbb{N}) \to \mathbb{R}^{n,m} \times \mathbb{R}^{m,l} \to \mathbb{R}^{n,l} \end{array}$$

Martin-Löf Type Theory

- Martin-Löf Type Theory (MLTT) is understood as an open system.
- One can add new types given by
  - formation
  - introduction and elimination
  - computation (β)
  - extensionality (η)
- We will formulate it with two judgements:

1. $\Gamma \vdash M : A$ "in context Γ, expression M has type A"
   Established by formation ($A = s$), introduction, and elimination.
2. $\Gamma \vdash M = M' : A$ "inhabitants M and M' of A are definitionally equal"
   Established by computation (β) and extensionality (η).

Basic Rules

- Hypotheses.

$$\frac{(x{:}A) \in \Gamma}{\Gamma \vdash x : A}\ \text{hyp}$$

- Conversion.

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

- Equivalence rules for judgemental equality.

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M = M : A}\ \text{refl} \qquad \frac{\Gamma \vdash M = M' : A}{\Gamma \vdash M' = M : A}\ \text{sym}$$

$$\frac{\Gamma \vdash M_1 = M_2 : A \qquad \Gamma \vdash M_2 = M_3 : A}{\Gamma \vdash M_1 = M_3 : A}\ \text{trans}$$

Dependent function type revisited

- Formation.

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

- Introduction.

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

- Elimination.

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

- Computation.

$$\frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x.M)\,N = M[N/x] : B[N/x]}\ \Pi\beta$$

Dependent function type revisited

- Extensionality (note $x \notin \mathsf{FV}(M)$).

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

- Compatibility rules

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

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

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

Dependent pairs (strong Σ-type)

- Formation.

$$\frac{\Gamma \vdash A : \mathsf{type} \qquad \Gamma, x{:}A \vdash B : \mathsf{type}}{\Gamma \vdash (x : A) \times B : \mathsf{type}}\ \Sigma\mathrm{F}$$

- Introduction.

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B[M/x]}{\Gamma \vdash \langle M,\ N\rangle : (x : A) \times B}\ \Sigma\mathrm{I}$$

- Elimination.

$$\frac{\Gamma \vdash M : (x : A) \times B}{\Gamma \vdash \mathsf{fst}\, M : A}\ \Sigma\mathrm{E}_1 \qquad \frac{\Gamma \vdash M : (x : A) \times B}{\Gamma \vdash \mathsf{snd}\, M : B[\mathsf{fst}\, M/x]}\ \Sigma\mathrm{E}_2$$

- Computation.

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \mathsf{fst}\, \langle M,\ N\rangle = M : A}\ \Sigma\beta_1 \quad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \mathsf{snd}\, \langle M,\ N\rangle = N : B}\ \Sigma\beta_2$$

Interpretations of the Σ type

- Non-dependent: cartesian product $A \times B$.

	B *prop*	B *type*
A *prop*	conjunction $A \wedge B$	mix
A *type*	mix	pair

- Dependent pair type $(x : A) \times B$.

	B *prop*	B *type*
A *prop*	proof-relevant conjunction	proof-rel. dep. pair
A *type*	existential quant. $\exists x{:}A.\, B$	dependent pair$\Sigma x{:}A.\, B$

Σ type: examples

- Lists from vectors: $\mathsf{List}\,\mathbb{R} = (n : \mathbb{N}) \times \mathbb{R}^n$
- Positive numbers: $\mathsf{Pos} = (n : \mathbb{N}) \times (n \geq 1)$
- *Exercises:*
  1. Assuming divisibility $\mathsf{Divides} : \mathbb{N} \to \mathbb{N} \to \mathsf{type}$, define the type of prime numbers.
  2. Define the type of lists of length $< n$.
  3. Define the type of monotone functions on $\mathbb{N}$.

Booleans

- Formation

$$\frac{}{\Gamma \vdash \mathsf{Bool} : \mathsf{type}}\ \mathsf{BoolF}$$

- Introduction

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}}\ \mathsf{BoolI}_1 \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}}\ \mathsf{BoolI}_2$$

- Elimination

$$\frac{\Gamma \vdash M : Bool \qquad \begin{array}{c}\Gamma, x{:}\mathsf{Bool} \vdash C : s \\ \Gamma \vdash N : C[\mathsf{true}/x]\end{array} \qquad \Gamma \vdash O : C[\mathsf{false}/x]}{\Gamma \vdash \mathsf{if}_{x.C}\ M\ \mathsf{then}\ N\ \mathsf{else}\ O : C[M/x]}\ \mathsf{BoolE}$$

- Computation

$$\frac{\Gamma, x{:}\mathsf{Bool} \vdash C : s \qquad \Gamma \vdash N : C[\mathsf{true}/x] \qquad \Gamma \vdash O : C[\mathsf{false}/x]}{\begin{array}{c}\Gamma \vdash \mathsf{if}_{x.C}\ \mathsf{true}\ \mathsf{then}\ N\ \mathsf{else}\ O = N : C[\mathsf{true}/x] \\ \Gamma \vdash \mathsf{if}_{x.C}\ \mathsf{false}\ \mathsf{then}\ N\ \mathsf{else}\ O = O : C[\mathsf{false}/x]\end{array}}\ \mathsf{Bool}\beta$$

Booleans (ctd.)

- Extensionality: Every boolean is either true or false [1, 2].
- Open problem: adding Bool extensionality to MLTT.
- Type-checking will become complicated but powerful:

$$f : \mathsf{Bool} \to \mathsf{Bool},\ x : \mathsf{Bool} \ \vdash \ f\,(f\,(f\,x)) = f\,x \ : \ \mathsf{Bool}$$

  would hold *definitionally*.

- *Exercise: add the compatibility rules for* if_then_else_*!*
- Programming with the booleans:

$$\begin{array}{lcl} \mathsf{not} & : & \mathsf{Bool} \to \mathsf{Bool} \\ \mathsf{not} & = & \lambda x.\ \mathsf{if\_}_{.\mathsf{Bool}}\ x\ \mathsf{then\ false\ else\ true} \end{array}$$

- *Exercise: Define other boolean functions, like exclusive-or* xor*!*

Defining the disjoint union

- Disjoint union $A + B$ is definable using if-then-else with types!

$$
\begin{array}{lcl}
\_ + \_ & : & \mathsf{type} \to \mathsf{type} \to \mathsf{type} \\
A + B & = & (x : \mathsf{Bool}) \times \mathsf{if}\_{.\mathsf{type}}\ x\ \mathsf{then}\ A\ \mathsf{else}\ B
\end{array}
$$

- Here we eliminate a value $x : \mathsf{Bool}$ to produce a type.
- This is called a large elimination (aka strong elimination).

$$
\begin{array}{lcl}
\mathsf{inl} & : & (A\ B : \mathsf{type}) \to A + B \to A \\
\mathsf{inl} & = & \lambda A.\lambda B.\lambda a.\, \langle \mathsf{true},\ a \rangle \\
\mathsf{inr} & : & (A\ B : \mathsf{type}) \to A + B \to B \\
\mathsf{inr} & = & \lambda A.\lambda B.\lambda b.\, \langle \mathsf{false},\ b \rangle
\end{array}
$$

- *Exercise: Define*

$$
\mathsf{case} : (A\ B\ C : \mathsf{type}) \to A + B \to (A \to C) \to (B \to C) \to C
$$

and check its computation laws!

Natural numbers and induction

- Formation.

$$\frac{}{\Gamma \vdash \mathbb{N} : \mathsf{type}}\ \mathbb{N}\mathrm{F}$$

- Introduction.

$$\frac{}{\Gamma \vdash \mathsf{zero} : \mathbb{N}}\ \mathbb{N}\mathrm{I}_1 \qquad \frac{\Gamma \vdash M : \mathbb{N}}{\Gamma \vdash \mathsf{suc}\, M : \mathbb{N}}\ \mathbb{N}\mathrm{I}_2$$

- Elimination.

$$\frac{\begin{array}{rl} \Gamma, x{:}\mathbb{N} \vdash & C : s \\ \Gamma \vdash & M_0 : C[\mathsf{zero}/x] \\ \Gamma, y{:}\mathbb{N}, ih : C[y/x] \vdash & M_1 : C[\mathsf{suc}\, y/x] \\ \Gamma \vdash & N : \mathbb{N} \end{array}}{\Gamma \vdash \mathsf{rec}\mathbb{N}_{x.C}(M_0,\ y.ih.M_1,\ N) : C[N/x]}\ \mathbb{N}\mathrm{E}$$

Natural numbers: computation

$$\frac{\begin{array}{r} \Gamma, x{:}\mathbb{N} \vdash C : s \\ \Gamma \vdash M_0 : C[\mathsf{zero}/x] \\ \Gamma, y{:}\mathbb{N}, ih : C[y/x] \vdash M_1 : C[\mathsf{suc}\, y/x] \end{array}}{\Gamma \vdash \mathsf{rec}\mathbb{N}_{x.C}(M_0,\ y.ih.M_1,\ \mathsf{zero}) = M_0 : C[\mathsf{zero}/x]}\ \mathbb{N}\beta_1$$

$$\frac{\begin{array}{r} \Gamma, x{:}\mathbb{N} \vdash C : s \\ \Gamma \vdash M_0 : C[\mathsf{zero}/x] \\ \Gamma, y{:}\mathbb{N}, ih : C[y/x] \vdash M_1 : C[\mathsf{suc}\, y/x] \\ \Gamma \vdash N : \mathbb{N} \end{array}}{\begin{array}{l} \Gamma \vdash \mathsf{rec}\mathbb{N}_{x.C}(M_0,\ y.ih.M_1,\ \mathsf{suc}\, N) \\ \quad = M_1[N/y, \mathsf{rec}\mathbb{N}_{x.C}(M_0,\ y.ih.M_1,\ N)/ih] : C[\mathsf{suc}\, N/x] \end{array}}\ \mathbb{N}\beta_2$$

Programming with natural numbers

- Elimination for $\mathbb{N}$ is higher-order primitive recursion.
- Predecessor and addition:

$$\begin{array}{lcl} \mathsf{pred} & : & \mathbb{N} \to \mathbb{N} \\ \mathsf{pred} & = & \lambda n.\ \mathsf{rec}\mathbb{N}\_\_.\mathbb{N}(\mathsf{zero},\ y.\_.y,\ n) \end{array}$$

$$\begin{array}{lcl} \mathsf{plus} & : & \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\ \mathsf{plus} & = & \lambda n.\lambda m.\ \mathsf{rec}\mathbb{N}\_\_.\mathbb{N}(m,\ \_.z.\mathsf{suc}\,z,\ n) \end{array}$$

- *Exercise: define multiplication and subtraction!*

Identity type

- Formation

$$\frac{\Gamma \vdash A : \mathsf{type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathsf{Id}_A(M, N) : \mathsf{type}}\ \mathsf{IdF}$$

- Introduction

$$\frac{\Gamma \vdash M = N : A}{\Gamma \vdash \mathsf{refl} : \mathsf{Id}_A(M, N)}\ \mathsf{IdI}$$

- Elimination (substitution) and computation

$$\frac{\begin{array}{c}\Gamma \vdash A : \mathsf{type} \qquad \Gamma, x{:}A \vdash C : \mathsf{type} \\ \Gamma \vdash M : A \qquad \Gamma \vdash N : A \qquad \Gamma \vdash P : \mathsf{Id}_A(M, N) \\ \Gamma \vdash O : C[M/x]\end{array}}{\begin{array}{c}\Gamma \vdash \mathsf{subst}_{A,x.C}(M,\ N,\ P,\ O) : C[N/x] \\ \Gamma \vdash \mathsf{subst}_{A,x.C}(M,\ M,\ \mathsf{refl},\ O) = O : C[M/x]\end{array}}\ \mathsf{IdE}^-$$

Identity type: more elimination rules

- Full elimination (J)

$$\frac{\begin{array}{c}\Gamma \vdash A : \mathsf{type} \qquad \Gamma, x{:}A, y{:}A, p : \mathsf{Id}_A(x,y) \vdash C : \mathsf{type} \\ \Gamma \vdash M : A \qquad \Gamma \vdash N : A \qquad \Gamma \vdash P : \mathsf{Id}_A(M,N) \\ \Gamma, z{:}A \vdash O : C[z/x, z/y, \mathsf{refl}/p]\end{array}}{\begin{array}{l}\Gamma \vdash \mathsf{J}_{A,x.y.p.C}(M,\ N,\ P,\ z.O) : C[M/x, N/y, P/p] \\ \Gamma \vdash \mathsf{J}_{A,x.y.p.C}(M,\ M,\ \mathsf{refl},\ z.O) = O[M/x] : C[M/x, M/y, \mathsf{refl}/p]\end{array}}\ \mathsf{IdE}$$

- Uniquess of identity proofs (Streicher's K axiom) [7]

$$\frac{\begin{array}{c}\Gamma \vdash A : \mathsf{type} \qquad \Gamma, x{:}A, p : \mathsf{Id}_A(x,x) \vdash C : \mathsf{type} \\ \Gamma \vdash M : A \qquad \Gamma \vdash P : \mathsf{Id}_A(M,M) \\ \Gamma, z{:}A \vdash O : C[z/x, \mathsf{refl}/p]\end{array}}{\begin{array}{l}\Gamma \vdash \mathsf{K}_{A,x.p.C}(M,\ P,\ z.O) : C[M/x, P/p] \\ \Gamma \vdash \mathsf{K}_{A,x.p.C}(M,\ \mathsf{refl},\ z.O) = O[M/x] : C[M/x, \mathsf{refl}/p]\end{array}}\ \mathsf{IdE}$$

Digression: groupoid interpretation

- Each type A can be interpreted as a category.
- $\mathsf{Id}_A(M, N)$ is the set of morphims between objects $M, N : A$.
- refl is the identity morphims.
- Transitivity is morphism composition.
- Symmetry makes the category into a groupoid [3].
- Adding K makes groupoid trivial: $\mathsf{Id}_A(M, N)$ has at most one inhabitant.

Digression: Homotopy Type Theory

- Started by Field's medallist Vladimir Voevodsky.
- Drop axiom K.
- Interpret $\mathsf{Id}_A(M, N)$ as path from M to N.
- Pathes form a groupoid.
- Groupoid laws form again a groupoid... ω-groupoid.

Equality proofs

- *Exercise: for the identity type, prove:*
  - subst is definable from J
  - symmetry is definable with subst
  - transitivity is definable with subst
  - if you are courageous: K implies uniqueness of identity proofs (UIP): $\mathsf{Id}_{\mathsf{Id}_A(M,N)}(P, Q)$ is inhabited.
- *Exercise: show that* IdI *is equivalent to:*

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{refl} : \mathsf{Id}_A(M, M)}\ \mathsf{IdI}'$$

Agda

- For the rest, we use Agda!
- More user-friendly interface to Type Theory:
    - User-definable data types
    - Function definitions by pattern matching
    - Termination checking

References I

Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Philip J. Scott.
Normalization by evaluation for typed lambda calculus with coproducts.
In *LICS'01*, pages 303–310. IEEE CS Press, 2001.

Thorsten Altenkirch and Tarmo Uustalu.
Normalization by evaluation for $\lambda^{\rightarrow 2}$.
In *FLOPS'04*, volume 2998 of *LNCS*, pages 260–275. Springer, 2004.

Martin Hofmann and Thomas Streicher.
The groupoid model refutes uniqueness of identity proofs.
In *LICS'94*, pages 208–212. IEEE CS Press, 1994.

References II

Per Martin-Löf.
An intuitionistic theory of types: Predicative part.
In *Logic Colloquium '73*, pages 73–118. North-Holland, 1975.

Bengt Nordström, Kent Petersson, and Jan M. Smith.
Programming in Martin Löf's Type Theory: An Introduction.
Clarendon Press, Oxford, 1990.

Ulf Norell.
Towards a Practical Programming Language Based on Dependent Type Theory.
PhD thesis, Chalmers, Göteborg, Sweden, 2007.

Thomas Streicher.
Investigations into Intensional Type Theory, 1993.
Habilitation thesis, Ludwig-Maximilians-University Munich.