CM0859 – MT5009 Type Theory
Martin-Löf's Type Theory

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2025-2

Preliminaries

Textbook

Mimram ([2000] 2025). Program = Proof.

Other references

Martin-Löf (1975). An Intuitionistic Theory of Types: Predicate Part.
Martin-Löf (1985). Constructive Mathematics and Computer Programming.
Rijke (2022). Introduction to Homotopy Type Theory.

Conventions

- The numbers and page numbers assigned to chapters, examples, exercises, figures, quotes, sections and theorems on these slides correspond to the numbers assigned in the textbook.
- The words "term" and "element" are synonymous. Both words are used interchangeably in these slides.

Introduction

- Martin-Löf Type Theory (MLTT) is also called Constructive Type Theory or Intuitionistic Type Theory.
- MLTT is a foundational system for constructive mathematics.
- MLTT is a dependent type theory.
- MLTT is the basis of some proof assistants.
- There are various versions of MLTT proposed by Martin-Löf and by other.
- MLTT is an open theory.

Dependent Types

- Primitive types

  Char, Bool, Nat, Int, ...

- Compound types

  Nat $\to$ Bool (non-dependent function type)
  Bool $\times$ Char (non-dependent product type)

- Types parameterised by types

  List Char, List Int, Maybe Bool, Either String Int, ...

- Dependent types: Types parameterised by terms (elements)

  Vec Char 2 (vector of characters of length 2)
  $\Pi(n : \mathsf{Nat}).\, \mathsf{Vec\ Char\ n}$ (dependent function type)

Inference Rules

Definition

MLTT is a formal system defined by **inference rules** of the form

$$\frac{J_1 \quad \dots \quad J_n}{J} \text{ rule name}$$

where J is the **conclusion**, $J_1, \dots, J_n$ are the **premises** (or **hypotheses**) and $J, J_1, \dots, J_n$ are judgements.

Core Dependent Type Theory

Our presentation of MLTT starts by introducing a core dependent type theory (CDTT) which only contains dependent function types.

Expressions

Definition

Expressions of CDTT are defined by the following grammar:

$$
\begin{aligned}
e, e' ::=\ & x && \text{(variable)}\\
\mid\ & e\, e' && \text{(application)}\\
\mid\ & \lambda x^{e}.e' && \text{(λ-abstraction)}\\
\mid\ & \Pi(x : e).e' && \text{(dependent function type)}\\
\mid\ & \mathsf{Type} && \text{(the type of small types)}
\end{aligned}
$$

Expressions

Definition

Expressions of CDTT are defined by the following grammar:

$$
\begin{array}{rcll}
e, e' & ::= & x & \text{(variable)} \\
& \mid & e\, e' & \text{(application)} \\
& \mid & \lambda x^{e}.e' & \text{(λ-abstraction)} \\
& \mid & \Pi(x : e).e' & \text{(dependent function type)} \\
& \mid & \mathsf{Type} & \text{(the type of small types)}
\end{array}
$$

Convention

We shall write t, u and A, B for expressions thought of as terms and as types, respectively.

Expressions

Definition

In the expressions $\lambda x^A.t$ and $\Pi(x : A).B$ the variable x is bounded. The set of **free variables** of an expression e, denoted $\mathrm{FV}(e)$, is recursively defined by:

$$
\begin{aligned}
\mathrm{FV}(x) &= \{x\}, \\
\mathrm{FV}(t\,u) &= \mathrm{FV}(t) \cup \mathrm{FV}(u), \\
\mathrm{FV}(\lambda x^A.y) &= \mathrm{FV}(A) \cup (\mathrm{FV}(t) - \{x\}), \\
\mathrm{FV}(\Pi(x : A).B) &= \mathrm{FV}(A) \cup (\mathrm{FV}(B) - \{x\}), \\
\mathrm{FV}(\mathsf{Type}) &= \emptyset.
\end{aligned}
$$

Substitution

Definition

The result of substituting an expression u for every free occurrence of a variable x in an expression e, denoted by $e\,[\,x := u\,]$, is defined recursively by:

$$
\begin{aligned}
x\,[\,x := u\,] &= u; && \\
y\,[\,x := u\,] &= y, && \text{if } x \neq y; \\
(t\,t')\,[\,x := u\,] &= (t\,[\,x := u\,])\,(t'\,[\,x := u\,]); && \\
(\lambda y^{A}.t)\,[\,x := u\,] &= \lambda y^{A\,[\,x := u\,]}.\,t\,[\,x := u\,], && \text{with } y \notin \mathrm{FV}(u) \cup \{x\}; \\
(\Pi(y : A).B)\,[\,x := u\,] &= \Pi(y : A\,[\,x := u\,]).\,B\,[\,x := u\,], && \text{with } y \notin \mathrm{FV}(u) \cup \{x\}; \\
\mathsf{Type}\,[\,x := u\,] &= \mathsf{Type}. &&
\end{aligned}
$$

Substitution

Definition

The result of substituting an expression u for every free occurrence of a variable x in an expression e, denoted by $e\,[\,x := u\,]$, is defined recursively by:[†]

$$
\begin{aligned}
x\,[\,x := u\,] &= u; && \\
y\,[\,x := u\,] &= y, && \text{if } x \neq y; \\
(t\,t')\,[\,x := u\,] &= (t\,[\,x := u\,])\,(t'\,[\,x := u\,]); && \\
(\lambda y^{A}.t)\,[\,x := u\,] &= \lambda y^{A\,[\,x:=u\,]}.\,t\,[\,x := u\,], && \text{with } y \notin \mathrm{FV}(u) \cup \{x\}; \\
(\Pi(y : A).B)\,[\,x := u\,] &= \Pi(y : A\,[\,x := u\,]).\,B\,[\,x := u\,], && \text{with } y \notin \mathrm{FV}(u) \cup \{x\}; \\
\mathsf{Type}\,[\,x := u\,] &= \mathsf{Type}. &&
\end{aligned}
$$

[†] In the textbook the substitution $e\,[\,x := u\,]$ is denoted by $e\,[\,u/x\,]$.

Contexts

Definition

A **context** Γ is finite list of **variable declarations**

$$\Gamma = x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n,$$

where the x_i are variables and the A_i are expressions.

Contexts

Definition

A **context** Γ is finite list of **variable declarations**

$$\Gamma = x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n,$$

where the x_i are variables and the A_i are expressions.

Observation

Note that the expression A_i can depend of variables $x_1, x_2, \ldots, x_{i-1}$.

Contexts

Definition

A **context** Γ is finite list of **variable declarations**

$$\Gamma = x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n,$$

where the x_i are variables and the A_i are expressions.

Observation

Note that the expression A_i can depend of variables $x_1, x_2, \ldots, x_{i-1}$.

Notation

Sometimes the empty context is written $\emptyset$.

Definitional Equality

Description

Definitional equality (or **judgemental equality** or **conversion**) in MLTT is a notion of equality where two expressions are considered equal if they are syntactically identical or can be transformed into one another through computation (β-equality, η-equality, etc). This relation shall be denoted by '$\doteq$'.

Definitional Equality

Description

Definitional equality (or **judgemental equality** or **conversion**) in MLTT is a notion of equality where two expressions are considered equal if they are syntactically identical or can be transformed into one another through computation (β-equality, η-equality, etc). This relation shall be denoted by '$\doteq$'.[†]

[†] The textbook denotes the definitional equality by '$=$'. We follow the notation in (Rijke 2022).

Definitional Equality

Features

- The definitional equality is an equivalence relation.
- We can substitute types by equal ones.
- We need to add congruence rules for each new type.

Judgements

Definition

There are three **judgements** in CDTT.

(i) $\vdash \Gamma \; \mathrm{ctx}$ (Γ is a **well-formed context**)

(ii) $\Gamma \vdash t : A$ (t is a expression of **type** A in the context Γ)

(iii) $\Gamma \vdash t \doteq u : A$ (t and u are **definitionally equal** expressions of type A in context Γ)

Judgements

Definition

There are three **judgements** in CDTT.

(i) $\vdash \Gamma \ \mathrm{ctx}$ (Γ is a **well-formed context**)[†]

(ii) $\Gamma \vdash t : A$ (t is a expression of **type** A in the context Γ)

(iii) $\Gamma \vdash t \doteq u : A$ (t and u are **definitionally equal** expressions of type A in context Γ)

[†] In the textbook the contexts are defined using sequents. We use a different notation for avoiding them. The judgement $\Gamma \vdash$ in the textbook is replaced by $\vdash \Gamma \ \mathrm{ctx}$.

Types and Terms

Types and terms

> *"There is no syntactic distinction between terms and types: both are expressions. The logic will however allow us to distinguish between the two. An expression A for which $\Gamma \vdash A : \mathsf{Type}$ is derivable for some context Γ is called a **type**. An expression t for which $\Gamma \vdash t : A$ is derivable for some context Γ and type A is called a **term**." (p. 364)*

Rules for Contexts

Definition

The **well-formed contexts** are defined by the following rules:

$$\frac{}{\vdash \emptyset \ \mathrm{ctx},} \qquad \frac{\Gamma \vdash A : \mathsf{Type}}{\vdash \Gamma, x : A \ \mathrm{ctx}.}$$

Rules for Contexts

Definition

The **well-formed contexts** are defined by the following rules:[†]

$$\frac{}{\vdash \emptyset \ \mathrm{ctx},} \qquad \frac{\Gamma \vdash A : \mathsf{Type}}{\vdash \Gamma, x : A \ \mathrm{ctx}.}$$

[†]The rules in the textbook uses sequents. We use different rules for avoiding them.

Definitional Equality

Equivalence relation

The definitional equality is a relation of equivalence on terms.

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \doteq t : A,}\ \text{refl}$$

$$\frac{\Gamma \vdash t \doteq u : A}{\Gamma \vdash u \doteq t : A,}\ \text{sym}$$

$$\frac{\Gamma \vdash t \doteq u : A \qquad \Gamma \vdash u \doteq v : A}{\Gamma \vdash t \doteq v : A.}\ \text{trans}$$

Definitional Equality

Substitution of equal types

We can substitute a type by an equal one.

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \doteq B : \mathsf{Type}}{\Gamma \vdash t : B,}$$

$$\frac{\Gamma \vdash t \doteq u : A \qquad \Gamma \vdash \Gamma \vdash A \doteq B : \mathsf{Type}}{\Gamma \vdash t \doteq u : A.}$$

Terms and Rules for Type Constructors

Description

For each new type constructor we shall need:

- Three constructions for expressions
  1. (1) a constructor for the type,
  2. (2) a constructor for the terms of this type,
  3. (3) an eliminator for the terms of this type.
- Six inference rules
  1. (1) formation: construct a type with the type constructor,
  2. (2) introduction: construct a term of the type,
  3. (3) elimination: use a term of the type,
  4. (4) computation: β-reduces a term of the type,
  5. (5) uniqueness: express a uniqueness property of the constructed terms, which corresponds to an η-equivalence rule,
  6. (6) congruence: express that definitional equality is compatible with the term constructors.

Dependent Function Types (Π-types)

Dependent Function Types (Π-types)

- Formation rule and associated congruence rule

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

Dependent Function Types (Π-types)

- Formation rule and associated congruence rule

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

$$\frac{\Gamma \vdash A \doteq A' : \mathsf{Type} \qquad \Gamma, x : A \vdash B \doteq B' : \mathsf{Type}}{\Gamma \vdash \Pi(x : A).B \doteq \Pi(x : A').B' : \mathsf{Type}.}\ \Pi\mathsf{F}^{\doteq}$$

Dependent Function Types (Π-types)

- Introduction rule and associated congruence rule

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

Dependent Function Types (Π-types)

- Introduction rule and associated congruence rule

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

$$\frac{\Gamma \vdash A \doteq A' : \mathsf{Type} \qquad \Gamma, x : A \vdash t \doteq t' : B}{\Gamma \vdash \lambda x^{A}.t \doteq \lambda x^{A'}.t' : \Pi(x : A).B.}\ \Pi\mathsf{I}^{\doteq}$$

Dependent Function Types (Π-types)

- Elimination rule and associated congruence rule

$$\frac{\Gamma \vdash t : \Pi(x : A).B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B\,[\,x := u\,].}\;\Pi\mathsf{E}$$

Dependent Function Types (Π-types)

- Elimination rule and associated congruence rule

$$\frac{\Gamma \vdash t : \Pi(x : A).B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B\,[\,x := u\,].}\ \Pi\mathsf{E}$$

$$\frac{\Gamma \vdash t \doteq t' : \Pi(x : A).B \qquad \Gamma \vdash u \doteq u' : A}{\Gamma \vdash t\,u \doteq t'\,u' : B\,[\,x := u\,].}\ \Pi\mathsf{E}^{\doteq}$$

Dependent Function Types (Π-types)

- Computation rule

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x^{A}.t)\, u \doteq t\,[\,x := u\,] : B\,[\,x := u\,].}\ \Pi\beta, \Pi\mathrm{C}$$

Dependent Function Types (Π-types)

- Uniqueness rule

$$\frac{\Gamma \vdash t : \Pi(x : A).B}{\Gamma \vdash t \doteq \lambda x^A.\, t\, x : \Pi(x : A).B.}\ \Pi\eta, \Pi\mathsf{U}$$

Dependent Function Types (Π-types)

Observations

- The dependent function type $\Pi(x : A).B$ is also denoted
  $\Pi_{(x:A)}B(x)$,
  $(x : A) \to B(x)$ or
  $(x : A) \to B$.

Dependent Function Types (Π-types)

Observations

- The dependent function type $\Pi(x : A).B$ is also denoted
  $\Pi_{(x:A)} B(x)$,
  $(x : A) \to B(x)$ or
  $(x : A) \to B$.
- The non-dependent function type is defined from the dependent function type when the type B does not depend of $x : A$, that is,

$$A \to B :\equiv \Pi(\_ : A).B.$$

Dependent Function Types (Π-types)

Observations

- The dependent function type $\Pi(x : A).B$ is also denoted
  $\Pi_{(x:A)}B(x)$,
  $(x : A) \to B(x)$ or
  $(x : A) \to B$.
- The non-dependent function type is defined from the dependent function type when the type B does not depend of $x : A$, that is,

$$A \to B :\equiv \Pi(\_ : A).B.$$

- Under the propositions-as-types principle the dependent function type $\Pi(x : A).B$ corresponds to the universal quantifier $(\forall x \in A)B$.

References

Per Martin-Löf (1975). An Intuitionistic Theory of Types: Predicate Part. In: Logic Colloquium 1973. Ed. by H. E. Rose and J. C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, pp. 73–118 (cit. on p. 2).

Per Martin-Löf (1982). Constructive Mathematics and Computer Programming. In: Logic, Methodology and Philosophy of Science VI (1979). Ed. by L. J. Cohen et al. Vol. 104. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, pp. 153–175. DOI: 10.1016/S0049-237X(09)70189-2 (cit. on p. 38).

Per Martin-Löf (1985). Constructive Mathematics and Computer Programming. In: Mathematical Logic and Programming Languages. Ed. by C. A. R. Hoare and J. C. Shepherdson. Reprinted from (Martin-Löf 1982) with a short discussion added. Prentice/Hall International, pp. 167–184 (cit. on p. 2).

Samuel Mimram [2000] (21st Oct. 2025). Program = Proof. Independently published. URL: https://www.lix.polytechnique.fr/Labo/Samuel.Mimram/publications/ (cit. on p. 2).

References

Egbert Rijke (2022). Introduction to Homotopy Type Theory. Draft version. URL: https://arxiv.org/abs/2212.11082 (cit. on pp. 2, 16).