



Martin-Löf type theory

Propositions as types, proofs as programs

11 July 2016

柯向上

(日本) 国立情報学研究所

hsiang-shang@nii.ac.jp

Annotated derivation

$$\begin{array}{c}
 \frac{}{A, A \rightarrow B \vdash A \rightarrow B} \text{ (assum)} \quad \frac{}{A, A \rightarrow B \vdash A} \text{ (assum)} \\
 \hline
 \frac{}{A, A \rightarrow B \vdash B} \text{ (}\rightarrow\text{E)} \\
 \hline
 \frac{}{A \vdash (A \rightarrow B) \rightarrow B} \text{ (}\rightarrow\text{I)} \\
 \hline
 \vdash A \rightarrow (A \rightarrow B) \rightarrow B \text{ (}\rightarrow\text{I)}
 \end{array}$$

Handwritten red annotations: A red line connects the two top assumptions. The label (→E) is circled in red. The label (→I) is crossed out with a red 'X'. The label (→I) at the bottom is also crossed out with a red 'X'.

- Label elements in contexts with (distinct) names.
- Represent (assum) by the name of the assumption used.
- Represent (\rightarrow E) by juxtaposing the representations of its two sub-derivations.
- Represent (\rightarrow I) by prefixing $\lambda v.$ to the representation of its sub-derivation, where v is the name of the new assumption.

Annotated derivation

$$\frac{\frac{\frac{x : A, y : A \rightarrow B \vdash y : A \rightarrow B}{x : A, y : A \rightarrow B \vdash y x : B} \text{ (app)} \quad \frac{}{x : A, y : A \rightarrow B \vdash x : A} \text{ (var)}}{x : A \vdash \lambda y. y x : (A \rightarrow B) \rightarrow B} \text{ (abs)} \quad \frac{}{\vdash \lambda x. \lambda y. y x : A \rightarrow (A \rightarrow B) \rightarrow B} \text{ (abs)}$$

This is a **typing derivation** for the λ -term $\lambda x. \lambda y. y x$!

Simply typed λ -calculus (à la Curry)

Let the set of *types* be the implicational fragment of PROP, i.e., the subset of the propositional language generated by variables and implication only.

A λ -term t is said to *have type τ under context Γ* exactly when, using the following rules, there is a typing derivation of $\Gamma \vdash t : \tau$.

$$\frac{}{\Gamma \vdash v : \tau} \text{ (var) } \quad \text{if } (v : \tau) \in \Gamma$$

$$\frac{\Gamma, v : \sigma \vdash t : \tau}{\Gamma \vdash \lambda v. t : \sigma \rightarrow \tau} \text{ (abs) } \quad \frac{\Gamma \vdash t : \sigma \rightarrow \tau \quad \Gamma \vdash s : \sigma}{\Gamma \vdash t s : \tau} \text{ (app)}$$

Curry–Howard correspondence

Deduction systems and programming calculi can be put in correspondence — a corresponding pair of a deduction system and a programming calculus can be regarded as logical and computational interpretations of essentially the same set of syntactic objects.

Slogan: *Propositions are types. Proofs are programs.*

Natural deduction for full propositional logic corresponds to simply typed λ -calculus with constants: Defining the set of types to be PROP , the derivations in natural deduction (the proofs) correspond exactly to the well-typed λ -terms (the programs).

Unifying logic and computation

Martin-Löf's *intuitionistic type theory* was designed in the '70s to serve as a foundation for *intuitionistic mathematics*. It is simultaneously

- a computationally meaningful higher-order logic system and
- a very expressively typed functional programming language.

The dependently typed programming language Agda is theoretically based on MLTT.

Sets

Activities within type theory consist of construction of elements of various *sets* (which we regard as synonymous with “types”).

- Note that element construction includes proving logical propositions (when we use sets as propositions) and carrying out general mathematical constructions (e.g., constructing functions of type $\mathbb{N} \rightarrow \mathbb{N}$).

Specification of sets is thus the central part of type theory.

Set of sets

We assume that there is a set of sets named \mathcal{U} (for “universe”), so when we write down $\Gamma \vdash A : \mathcal{U}$, this states that A is a set under the assumptions in Γ .

Rules of type theory are formulated such that whenever $\Gamma \vdash t : A$ it is also the case that $\Gamma \vdash A : \mathcal{U}$.

Remark. Can we postulate $\mathcal{U} : \mathcal{U}$? The answer was shown by Girard to be no, because $\mathcal{U} : \mathcal{U}$ leads to inconsistency.

We thus need to introduce a *predicative* hierarchy of universes $\mathcal{U}_0, \mathcal{U}_1, \dots$, up to infinity, and postulate $\mathcal{U}_i : \mathcal{U}_{i+1}$.

In practice, however, we can forget about indexing and just assume $\mathcal{U} : \mathcal{U}$, because there is an algorithm for inferring the indices.

Set specification

To specify each set, we first give three kinds of rules:

- *Formation rule* — what constitute the name of the set.
- *Introduction rule(s)* — how to construct (canonical) elements of the set.
- *Elimination rule(s)* — how to deconstruct elements of the set and transform them to elements of some other sets.

The fourth kind of rules will be introduced later today.

Cartesian product types (conjunction)

- Formation:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A \times B : \mathcal{U}} (\times F)$$

- Introduction:

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B} (\times I)$$

- Elimination:

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \text{fst } p : A} (\times EL) \qquad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \text{snd } p : B} (\times ER)$$

$$\lambda p. (\text{snd } p, \text{fst } p)$$

Function types (implication)

- Formation:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A \rightarrow B : \mathcal{U}} (\rightarrow F)$$

- Introduction:

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

- Elimination:

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f a : B} (\rightarrow E)$$

Coprodut types (disjunction)

- Formation:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A + B : \mathcal{U}} (+F)$$

- Introduction:

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \text{left } a : A + B} (+IL) \quad \frac{\Gamma \vdash b : B}{\Gamma \vdash \text{right } b : A + B} (+IR)$$

- Elimination:

$$\frac{\Gamma \vdash s : A + B \quad \Gamma, x : A \vdash l : C \quad \Gamma, y : B \vdash r : C}{\Gamma \vdash \text{case } s \text{ of } \{\text{left } x. l; \text{right } y. r\} : C} (+E)$$

data Either a b = Left a | Right b

switch :: Either a b → Either b a

switch (Left x) = Right x

switch (Right y) = Left y

switch = λe → case e of

Left x → Right x

Right y → Left y

Empty type (falsity)

- Formation:

$$\frac{}{\Gamma \vdash \perp : \mathcal{U}} (\perp F)$$

- Introduction: none

- Elimination:

$$\frac{\Gamma \vdash b : \perp}{\Gamma \vdash \text{absurd } b : A} (\perp E)$$

Exercise. Which programs correspond to the proofs you constructed last Thursday?

Indexed families of sets as predicates

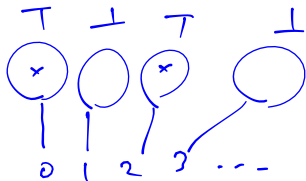
Mathematical statements usually involve predicates and universal/existential quantification.

For example: “For every $x : \mathbb{N}$, if x is not zero, then there exists $y : \mathbb{N}$ such that x is equal to $1 + y$.”

In type theory, a predicate on A can be thought of as having type $A \rightarrow \mathcal{U}$ — a *family of sets* indexed by the domain A . For example:

$\vdash \lambda x. \text{“if } x \text{ is zero then } \perp \text{ else } \top\text{”} : \mathbb{N} \rightarrow \mathcal{U}$

$\text{Even} :: \mathbb{N} \rightarrow \mathcal{U}$



Dependent product types (universal quantification)

- Formation:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash \Pi(x:A) B : \mathcal{U}} \text{ (IIF)}$$

Handwritten notes: \mathbb{N} above A , $\text{Even } x$ above B , and $\Pi(x:\mathbb{N}) \text{ Even } x$ to the right of the rule.

- Introduction:

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : \Pi(x:A) B} \text{ (III)}$$

- Elimination:

$$\frac{\Gamma \vdash f : \Pi(x:A) B \quad \Gamma \vdash a : A}{\Gamma \vdash f a : B[a/x]} \text{ (IIE)}$$

Handwritten notes: \mathbb{N} above A , $\text{Even } x$ above B , f above f , 42 above a , and $\text{Even } 42$ above $B[a/x]$.

Exercise. Let $\Gamma := A : \mathcal{U}, B : \mathcal{U}, C : A \rightarrow B \rightarrow \mathcal{U}$. Derive

$$\Gamma \vdash \neg : (\Pi(x:A) \Pi(y:B) C x y) \rightarrow \Pi(y:B) \Pi(x:A) C x y$$

Handwritten note: A red circle around the \neg symbol.

Dependent sum types (existential quantification)

- Formation:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash \Sigma(x:A) B : \mathcal{U}} \quad (\Sigma F)$$

- Introduction:

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a, b) : \Sigma(x:A) B} \quad (\Sigma I)$$

- Elimination:

$$\frac{\Gamma \vdash p : \Sigma(x:A) B}{\Gamma \vdash \text{fst } p : A} \quad (\Sigma EL) \quad \frac{\Gamma \vdash p : \Sigma(x:A) B}{\Gamma \vdash \text{snd } p : B[\text{fst } p/x]} \quad (\Sigma ER)$$

Exercise. Let $\Gamma := A : \mathcal{U}, B : \mathcal{U}, C : A \rightarrow B \rightarrow \mathcal{U}$. Derive

$$\Gamma \vdash _ : (\Sigma(p : A \times B) C(\text{fst } p)(\text{snd } p)) \rightarrow \Sigma(x:A) \Sigma(y:B) C x y$$

Sketch

$$g : \Sigma(p : A \times B) C(\text{fst } p)(\text{snd } p)$$

$$\text{fst } g : A \times B \quad \text{fst}(\text{fst } g) : A$$

$$\text{snd } g : C(\text{fst}(\text{fst } g))(\text{snd}(\text{fst } g))$$

$$\vdots$$

$$\Gamma \vdash \lambda g. (\text{fst}(\text{fst } g), (\text{snd}(\text{fst } g), \text{snd } g)) : \dots$$

Exercises

Let $\Gamma := A : \mathcal{U}, B : A \rightarrow \mathcal{U}, C : A \rightarrow \mathcal{U}$. Find proof terms such that the following are derivable:

$$\Gamma \vdash _ : (\Pi(x:A) Bx \times Cx) \leftrightarrow (\Pi(y:A) By) \times (\Pi(z:A) Cz)$$

$$\Gamma \vdash _ : (\Sigma(x:A) Bx + Cx) \leftrightarrow (\Sigma(y:A) By) + (\Sigma(z:A) Cz)$$

What about

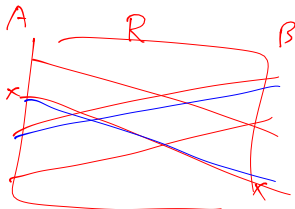
$$\Gamma \vdash _ : (\Pi(x:A) Bx + Cx) \leftrightarrow (\Pi(y:A) By) + (\Pi(z:A) Cz)$$

$$\Gamma \vdash _ : (\Sigma(x:A) Bx \times Cx) \leftrightarrow (\Sigma(y:A) By) \times (\Sigma(z:A) Cz)$$

?

Now let $\Gamma := A : \mathcal{U}, B : \mathcal{U}, R : A \rightarrow B \rightarrow \mathcal{U}$. Prove the *axiom of choice*, i.e., find a proof term for

$$\Gamma \vdash _ : (\Pi(x:A) \Sigma(y:B) Rxy) \rightarrow \Sigma(f:A \rightarrow B) \Pi(z:A) Rzfz$$



Computation

Let $\Gamma := A : \mathcal{U}, B : A \rightarrow \mathcal{U}, C : A \rightarrow \mathcal{U}$. Try to derive

$$\Gamma \vdash _ : (\Pi(p:\Sigma(x:A) B\ x) C(\text{fst } p)) \rightarrow \Pi(y:A) (B\ y \rightarrow C\ y)$$

... and you should notice a problem:

- Intuitively, $\lambda f. \lambda y. \lambda b. f(y, b)$ does the job.
- However, $f(y, b)$ has type $C(\text{fst}(y, b))$ rather than $C y$.

We need to incorporate computation into typing.

$$\frac{\frac{w : C(\text{fst}(y, b)) \quad \frac{y : A \quad b : B \quad y}{\text{fst}(y, b) = y \in A} (\text{EC})}{w : C_y} (\text{conv})$$

III-17

$$\dots \rightarrow \text{IsTrue}(\text{not}(\text{make xs})) \rightarrow \dots$$

Equality judgements

We introduce a new kind of judgement for stating that two terms should be regarded as the same during type-checking:

$$\Gamma \vdash t = u \in A$$

Rules will be formulated such that whenever $\Gamma \vdash t = u \in A$ is derivable, so are $\Gamma \vdash t : A$ and $\Gamma \vdash u : A$.

Computation rules

For each set, (when applicable) we specify additional *computation rules* stating how to eliminate an introductory term.

For example, for product types we have two computation rules:

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \text{fst}(a, b) = a \in A} (\times\text{CL}) \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \text{snd}(a, b) = b \in B} (\times\text{CR})$$

This is the type-theoretic manifestation of *Gentzen's inversion principle* saying that elimination rules should be justified in terms of introduction rules.

More computation rules

$$\frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x. t) a = t[a/x] \in B} (\rightarrow C)$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash f : A \rightarrow C \quad \Gamma \vdash g : B \rightarrow C}{\Gamma \vdash \text{case (left } a) f g = f a \in C} (+CL)$$

$\text{case (left } a) \text{ of } \{\text{left } x. l; \text{right } y. v\} = l[a/x]$

$$\frac{\Gamma \vdash b : B \quad \Gamma \vdash f : A \rightarrow C \quad \Gamma \vdash g : B \rightarrow C}{\Gamma \vdash \text{case (right } b) f g = g b \in C} (+CR)$$

More computation rules

$$\frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x. t) a = t[a/x] \in B[a/x]} \text{ (}\Pi\text{C)}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[a/x]}{\Gamma \vdash \text{fst}(a, b) = a \in A} \text{ (}\Sigma\text{CL)}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[a/x]}{\Gamma \vdash \text{snd}(a, b) = b \in B[a/x]} \text{ (}\Sigma\text{CR)}$$

Congruence rules

We need a congruence rule for each constant we introduce:

$$\frac{\Gamma \vdash a = a' \in A \quad \Gamma \vdash b = b' \in B}{\Gamma \vdash (a, b) = (a', b') \in A \times B}$$

$$\frac{\Gamma \vdash p = p' \in A \times B}{\Gamma \vdash \text{fst } p = \text{fst } p' \in A} \quad \frac{\Gamma \vdash p = p' \in A \times B}{\Gamma \vdash \text{snd } p = \text{snd } p' \in B}$$

$$\frac{\Gamma, x : A \vdash t = t' \in B}{\Gamma \vdash \lambda x. t = \lambda x. t' \in A \rightarrow B}$$

$$\frac{\Gamma \vdash f = f' \in A \rightarrow B \quad \Gamma \vdash a = a' \in A}{\Gamma \vdash f a = f' a' \in B}$$

... and similar rules for left, right, case, and absurd.

Equivalence rules

Judgemental equality is an equivalence relation.

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t = t \in A}$$

$$\frac{\Gamma \vdash t = u \in A}{\Gamma \vdash u = t \in A}$$

$$\frac{\Gamma \vdash t = u \in A \quad \Gamma \vdash u = v \in A}{\Gamma \vdash t = v \in A}$$

Conversion rule

Once we establish that two sets are judgementally equal, we can transfer terms between the two sets.

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash A = B \in \mathcal{U}}{\Gamma \vdash t : B} \text{ (conv)}$$

Exercise. Finish deriving

$\Gamma \vdash _ : (\Pi(p : \Sigma A B) C(\text{fst } p)) \rightarrow \Pi(x : A) (B x \rightarrow C x)$
(where $\Gamma := A : \mathcal{U}, B : A \rightarrow \mathcal{U}, C : A \rightarrow \mathcal{U}$).

More congruence rules

(We can state congruence rules for dependent products and sums only after we have the conversion rule. Why?)

$$\frac{\Gamma \vdash a = a' \in A \quad \Gamma \vdash b = b' \in B[a/x]}{\Gamma \vdash (a, b) = (a', b') \in \Sigma(x:A) B}$$

$$\frac{\Gamma \vdash p = p' \in \Sigma(x:A) B}{\Gamma \vdash \text{fst } p = \text{fst } p' \in A} \quad \frac{\Gamma \vdash p = p' \in \Sigma(x:A) B}{\Gamma \vdash \text{snd } p = \text{snd } p' \in B[\text{fst } p/x]}$$

$$\frac{\Gamma, x:A \vdash t = t' \in B}{\Gamma \vdash \lambda x. t = \lambda x. t' \in \Pi(x:A) B}$$

$$\frac{\Gamma \vdash f = f' \in \Pi(x:A) B \quad \Gamma \vdash a = a' \in A}{\Gamma \vdash f a = f' a' \in B[a/x]}$$

Decidability of judgemental equality

Our judgemental equality is *decidable*: for any equality judgement we can decide whether it has a derivation or not.

(As a consequence, typechecking is also decidable.)

Decidability is achieved by reducing terms to their *normal forms* and see if the normal forms match.

There are various reduction strategies, and judgemental equality is formulated without reference to any particular reduction strategy — it captures the notion of computation only abstractly.

Canonical vs non-canonical elements

Introduction rules specify *canonical* — or *immediately recognisable* — elements of a set.

A complex construction may not <sup>$A \times B$
 $(\lambda x. b)$</sup> be immediately recognisable as belonging to a set, but as long as we can see that it *computes* to a canonical element, we accept it as a *non-canonical* element of the set.

Remark. It follows that all computations in type theory must terminate, because from a non-canonical proof we should be able to get a canonical one in finite time.

Property (canonicity). If $\vdash t : A$, then $\vdash t = c \in A$ for some canonical element c .

Classical axioms from a type-theoretic perspective

We obtained a classical system NK by adding an inference rule to NJ. The same could also be done for MLTT by introducing a new constant:

$$\frac{\Gamma \vdash X : \mathcal{U}}{\Gamma \vdash \text{LEM } X : X + \neg X} \text{ (LEM)}$$

Exercise. Find a proof term such that

$$A : \mathcal{U}, B : A \rightarrow \mathcal{U} \vdash _ : (\neg \Pi(x : A) \neg B x) \rightarrow \Sigma(y : A) B y$$

is derivable.

We do not know how to formulate computation rules for LEM, however. This breaks canonicity, and the type theory ceases to be computationally meaningful.