Programming
Research
Laboratory

FLOLAC '16

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

$$\dfrac{\dfrac{\dfrac{}{\mathtt{A},\ \mathtt{A} \to \mathtt{B} \vdash \mathtt{A} \to \mathtt{B}}\,\text{(assum)} \qquad \dfrac{}{\mathtt{A},\ \mathtt{A} \to \mathtt{B} \vdash \mathtt{A}}\,\text{(assum)}}{\dfrac{\mathtt{A},\ \mathtt{A} \to \mathtt{B} \vdash \mathtt{B}}{\mathtt{A} \vdash (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\,(\to\text{I})}\,(\to\text{E})}{\vdash \mathtt{A} \to (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\,(\to\text{I})$$

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

Annotated derivation

$$\dfrac{\dfrac{\dfrac{}{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{y} : \mathtt{A} \to \mathtt{B}}\ (\mathsf{var}) \qquad \dfrac{}{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{x} : \mathtt{A}}\ (\mathsf{var})}{\dfrac{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{y}\ \mathtt{x} : \mathtt{B}}{\mathtt{x} : \mathtt{A} \vdash \lambda\, \mathtt{y}.\ \mathtt{y}\ \mathtt{x} : (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\ (\mathsf{abs})}\ (\mathsf{app})}{\vdash \lambda\, \mathtt{x}.\ \lambda\, \mathtt{y}.\ \mathtt{y}\ \mathtt{x} : \mathtt{A} \to (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\ (\mathsf{abs})$$

This is a typing derivation for the λ-term $\lambda\, \mathtt{x}.\ \lambda\, \mathtt{y}.\ \mathtt{y}\ \mathtt{x}$!

Simply typed λ-calculus (à la Curry)

Let the set of *types* be the implicational fragment of $\textsc{Prop}$, 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 $\Gamma \vdash t : \tau$.

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

$$\frac{\Gamma, v : \sigma \vdash t : \tau}{\Gamma \vdash \lambda v.\ t : \sigma \to \tau}\,(\mathsf{abs}) \qquad \frac{\Gamma \vdash t : \sigma \to \tau \qquad \Gamma \vdash s : \sigma}{\Gamma \vdash t\, s : \tau}\,(\mathsf{app})$$

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 $\textsc{Prop}$, 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 $\mathbb{N} \to \mathbb{N}$).

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 $\mathcal{U}_0$, $\mathcal{U}_1$, ..., 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:

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

- Introduction:

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

- Elimination:

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

Function types (implication)

- Formation:

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A \to B : \mathcal{U}} \ (\to\mathsf{F})$$

- Introduction:

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

- Elimination:

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\, a : B} \ (\to\mathsf{E})$$

Coproduct types (disjunction)

- Formation:

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

- Introduction:

$$\frac{\Gamma \;\vdash\; a : A}{\Gamma \;\vdash\; \texttt{left}\; a : A + B}\;(+\mathsf{IL}) \qquad \frac{\Gamma \;\vdash\; b : B}{\Gamma \;\vdash\; \texttt{right}\; b : A + B}\;(+\mathsf{IR})$$

- Elimination:

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

Empty type (falsity)

- Formation:

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

- Introduction: none
- Elimination:

$$\frac{\Gamma \vdash b : \bot}{\Gamma \vdash \texttt{absurd}\ b : A}\ (\bot\mathsf{E})$$

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 $A \to \mathcal{U}$ — a *family of sets* indexed by the domain A. For example:

$$\vdash \lambda x.\ \text{"if } x \text{ is zero then } \bot \text{ else } \top\text{"} : \mathbb{N} \to \mathcal{U}$$

Dependent product types (universal quantification)

- Formation:

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

- Introduction:

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

- Elimination:

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

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

$$\Gamma \ \vdash \ \_ : (\Pi(x : A)\ \Pi(y : B)\ C\,x\,y) \to \Pi(y : B)\ \Pi(x : A)\ C\,x\,y$$

Dependent sum types (existential quantification)

- Formation:

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

- Introduction:

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

- Elimination:

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

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

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

Exercises

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

$$\Gamma \ \vdash \ \_ : (\Pi(x:A)\ \ B\,x \times C\,x) \leftrightarrow (\Pi(y:A)\ \ B\,y) \times (\Pi(z:A)\ \ C\,z)$$

$$\Gamma \ \vdash \ \_ : (\Sigma(x:A)\ \ B\,x + C\,x) \leftrightarrow (\Sigma(y:A)\ \ B\,y) + (\Sigma(z:A)\ \ C\,z)$$

What about

$$\Gamma \ \vdash \ \_ : (\Pi(x:A)\ \ B\,x + C\,x) \leftrightarrow (\Pi(y:A)\ \ B\,y) + (\Pi(z:A)\ \ C\,z)$$

$$\Gamma \ \vdash \ \_ : (\Sigma(x:A)\ \ B\,x \times C\,x) \leftrightarrow (\Sigma(y:A)\ \ B\,y) \times (\Sigma(z:A)\ \ C\,z)$$

?

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

$$\Gamma \ \vdash \ \_ : (\Pi(x:A)\ \Sigma(y:B)\ \ R\,x\,y) \to \Sigma(f:A \to B)\ \Pi(z:A)\ \ R\,z\,(f\,z)$$

Computation

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

$\Gamma\ \vdash\ \_ : (\Pi(p : \Sigma(x : A)\ B\,x)\ C\,(\texttt{fst}\ p)) \to \Pi(y : A)\ (B\,y \to C\,y)$

... 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 $C\,(\texttt{fst}\ (y,\,b))$ rather than $C\,y$.

We need to incorporate computation into typing.

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:

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

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

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

$$\frac{\Gamma \ \vdash\ a : A \qquad \Gamma \ \vdash\ f : A \to C \qquad \Gamma \ \vdash\ g : B \to C}{\Gamma \ \vdash\ \mathtt{case}\ (\mathtt{left}\ a)\ f\, g = f\, a \ \in\ C}\ (+\mathrm{CL})$$

$$\frac{\Gamma \ \vdash\ b : B \qquad \Gamma \ \vdash\ f : A \to C \qquad \Gamma \ \vdash\ g : B \to C}{\Gamma \ \vdash\ \mathtt{case}\ (\mathtt{right}\ b)\ f\, g = g\, b \ \in\ C}\ (+\mathrm{CR})$$

More computation rules

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

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

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

Congruence rules

We need a congruence rule for each constant we introduce:

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

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

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

$$\frac{\Gamma \ \vdash \ f = f' \ \in \ A \to B \qquad \Gamma \ \vdash \ a = a' \ \in \ A}{\Gamma \ \vdash \ f\, a = f'\, a' \ \in \ B}$$

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

Equivalence rules

Judgemental equality is an equivalence relation.

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

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

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

Conversion rule

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

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

Exercise. Finish deriving

$$\Gamma \vdash \_ : (\Pi(p : \Sigma\, A\, B)\ C\,(\mathtt{fst}\ p)) \to \Pi(x : A)\ (B\, x \to C\, x)$$

(where $\Gamma := A : \mathcal{U}, B : A \to \mathcal{U}, C : A \to \mathcal{U}$).

More congruence rules

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

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

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

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

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

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 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:

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

Exercise. Find a proof term such that

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

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.