Quillen Model Categories
Model
Martin-Löf Type Theory with Identity Types

Samuel Mimram

CEA

Séminaire MeASI au CIRM

Disclaimer

Ideas are not from me (Awodey & Warren, Voevodsky, . . .),
errors are mine.

λ-calculus

- Introduction rule:

$$\frac{\Gamma, x : A \vdash f : B}{\Gamma \vdash \lambda x.f : A \to B}$$

λ-calculus

- Introduction rule:

$$\frac{\Gamma, x : A \vdash f : B}{\Gamma \vdash \lambda x.f : A \to B}$$

- Elimination rule:

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash g : A}{\Gamma \vdash fg : B}$$

λ-calculus

- Introduction rule:

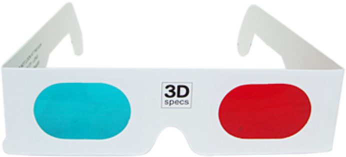
$$\frac{\Gamma, x : A \vdash f : B}{\Gamma \vdash \lambda x.f : A \to B}$$

- Elimination rule:

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash g : A}{\Gamma \vdash fg : B}$$

- Conversion rule:

$$\frac{\Gamma, x : A \vdash f : B \qquad \Gamma \vdash g : A}{\Gamma \vdash (\lambda x.f)g = f[g/x] : B}$$



Now with dependent types.

Dependent types

```
Array.make :  int -> array
```

Dependent types

```
Array.make :  n:int -> n array
```

Dependent types

$$\texttt{Array.make} \;:\; \Pi_{n:\texttt{int}}.\texttt{array}(n)$$

Dependent types

$$\texttt{Array.make} \;:\; \Pi_{n:\texttt{int}}.\texttt{array}(n)$$

- Type formation rule:

$$\frac{\vdash n : \texttt{int}}{\vdash \texttt{array}(n) : \mathsf{type}}$$

$$\texttt{Array.make} \; : \; \Pi_{n:\texttt{int}}.\texttt{array}(n)$$

- Type formation rule:

$$\frac{\vdash n : \texttt{int}}{\vdash \texttt{array}(n) : \text{type}}$$

- Introduction rules:

$$\frac{}{\Gamma \vdash [] : \texttt{array}(0)} \qquad \frac{\Gamma \vdash k : \texttt{int} \qquad \Gamma, n : \texttt{int} \vdash a : \texttt{array}(n)}{\Gamma, n : \texttt{int} \vdash (k :: a) : \texttt{array}(n+1)}$$

Products (and sums)

- Type formation rule:

$$\frac{x : A \vdash B(x) : \mathsf{type}}{\vdash \Pi_{x:A}.B(x) : \mathsf{type}}$$

Products (and sums)

- Type formation rule:

$$\frac{x : A \vdash B(x) : \mathsf{type}}{\vdash \Pi_{x:A}.B(x) : \mathsf{type}}$$

- Introduction rule:

$$\frac{x : A \vdash f(x) : B(x)}{\vdash \lambda_{x:A}.f(x) : \Pi_{x:A}.B(x)}$$

Products (and sums)

- Type formation rule:

$$\frac{x : A \vdash B(x) : \text{type}}{\vdash \Pi_{x:A}.B(x) : \text{type}}$$

- Introduction rule:

$$\frac{x : A \vdash f(x) : B(x)}{\vdash \lambda_{x:A}.f(x) : \Pi_{x:A}.B(x)}$$

- Elimination rule:

$$\frac{\vdash g : \Pi_{x:A}.B(x) \qquad \vdash x : A}{\vdash ga : B(a)}$$

Products (and sums)

- Type formation rule:

$$\frac{x : A \vdash B(x) : \text{type}}{\vdash \Pi_{x:A}.B(x) : \text{type}}$$

- Introduction rule:

$$\frac{x : A \vdash f(x) : B(x)}{\vdash \lambda_{x:A}.f(x) : \Pi_{x:A}.B(x)}$$

- Elimination rule:

$$\frac{\vdash g : \Pi_{x:A}.B(x) \qquad \vdash x : A}{\vdash ga : B(a)}$$

- Conversion rule:

$$\frac{x : A \vdash f(x) : B(x) \qquad \vdash a : A}{\vdash (\lambda_{x:A}.f(x))a = f(a) : B(a)}$$

Remark

The usual arrow type $A \to B$ is recovered as

$$\Pi_{x:A}.B$$

where x does not occur in B.

Identity types

- Type formation rule:

$$\frac{\vdash a : A \qquad \vdash b : A}{\vdash \mathsf{Id}_A(a, b) : \mathsf{type}}$$

Identity types

- Type formation rule:

$$\frac{\vdash a : A \qquad \vdash b : A}{\vdash \mathsf{Id}_A(a, b) : \mathsf{type}}$$

- Introduction rule:

$$\frac{\vdash a : A}{\vdash r_A(a) : \mathsf{Id}_A(a, a)}$$

Identity types

- Type formation rule:

$$\frac{\vdash a : A \qquad \vdash b : A}{\vdash \mathsf{Id}_A(a, b) : \mathsf{type}}$$

- Introduction rule:

$$\frac{\vdash a : A}{\vdash r_A(a) : \mathsf{Id}_A(a, a)}$$

- Elimination rule:

$$\frac{\begin{array}{c} x : A, y : A, z : \mathsf{Id}_A(x, y) \vdash D(x, y, z) : \mathsf{type} \\ \vdash p : \mathsf{Id}_A(a, b) \qquad x : A \vdash d(x) : D(x, x, r_A(x)) \end{array}}{\vdash J_{A,D}(d, a, b, p) : D(a, b, p)}$$

Identity types

- Type formation rule:

$$\frac{\vdash a : A \qquad \vdash b : A}{\vdash \mathsf{Id}_A(a, b) : \mathsf{type}}$$

- Introduction rule:

$$\frac{\vdash a : A}{\vdash r_A(a) : \mathsf{Id}_A(a, a)}$$

- Elimination rule:

$$\frac{\begin{array}{c} x : A, y : A, z : \mathsf{Id}_A(x, y) \vdash D(x, y, z) : \mathsf{type} \\ \vdash p : \mathsf{Id}_A(a, b) \qquad x : A \vdash d(x) : D(x, x, r_A(x)) \end{array}}{\vdash J_{A,D}(d, a, b, p) : D(a, b, p)}$$

- Conversion rule:

$$\frac{\begin{array}{c} x : A, y : A, z : \mathsf{Id}_A(x, y) \vdash D(x, y, z) : \mathsf{type} \\ \vdash a : A \qquad x : A \vdash d(x) : D(x, x, r_A(x)) \end{array}}{\vdash J_{A,D}(d, a, a, r_A(a)) = d(a) : D(a, a, r_A(a))}$$

Categories

A category $\mathcal{C}$ consists of

- objects: $\mathrm{Ob}(\mathcal{C})$
- morphisms: $\forall A, B \in \mathrm{Ob}(\mathcal{C}), \quad \mathrm{Hom}(A, B)$
- compositions:

$$\frac{f : A \to B \qquad g : B \to C}{g \circ f : A \to C}$$

- identities:

$$\forall A \in \mathrm{Ob}(\mathcal{C}), \quad \mathrm{id}_A : A \to A$$

such that

- composition is associative:

$$h \circ (g \circ f) = (h \circ g) \circ f$$

- admits identities as neutral elements

$$\mathrm{id} \circ f = f = f \circ \mathrm{id}$$

The category **Set**

The category **Set** has

- objects: sets
- morphisms: functions $f : A \to B$
- with usual composition and identities

Modeling programming languages

From a programming language, we can build a category Π whose

- objects: types
- morphisms: programs $\pi : A \to B$ modulo cut-elimination
- composition: usual composition of programs

Modeling programming languages

From a programming language, we can build a category Π whose

- objects: types
- morphisms: programs $\pi : A \to B$ modulo cut-elimination
- composition: usual composition of programs

Definition

A **model** of the programming language is a functor

$$F : \Pi \to \mathcal{C}$$

Models of simply typed λ-calculus

Take the category with

- objects: types

$$A \quad ::= \quad X \quad | \quad A \Rightarrow B \quad | \quad A \times B$$

- morphisms $A \to B$: λ-terms $f : A \Rightarrow B$

Models of simply typed λ-calculus

Take the category with

- objects: types

$$A \quad ::= \quad X \quad | \quad A \Rightarrow B \quad | \quad A \times B$$

- morphisms $A \to B$: λ-terms $f : A \Rightarrow B$

Example:

$$\lambda x.\lambda y.x : A \to (B \Rightarrow A)$$

Models of simply typed λ-calculus

Take the category with

- objects: types

$$A \quad ::= \quad X \quad | \quad A \Rightarrow B \quad | \quad A \times B$$

- morphisms $A \to B$: λ-terms $f : A \Rightarrow B$

Exercise: give a model of this language into **Set**.

Models of simply typed λ-calculus

Take the category with

- objects: types

$$A \quad ::= \quad X \quad | \quad A \Rightarrow B \quad | \quad A \times B$$

- morphisms $A \to B$: λ-terms $f : A \Rightarrow B$

More generally, it can be modeled in any cartesian closed category.

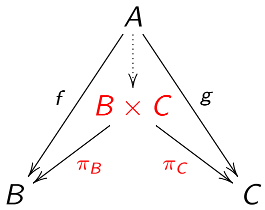
Cartesian closed categories

Definition

A **cartesian closed category** is a category which has

- *products*:

$$\forall f : A \to B, g : A \to C,$$



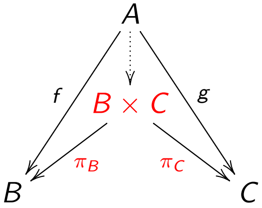
Cartesian closed categories

Definition

A **cartesian closed category** is a category which has

- *products*:

$$\forall f : A \to B, g : A \to C,$$



- a *terminal object* 1:

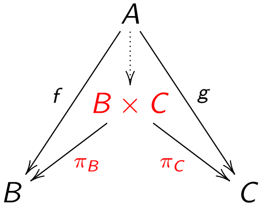
$$\forall A, \qquad A \dashrightarrow 1$$

Definition

A **cartesian closed category** is a category which has

- *products*:

$$\forall f : A \to B, g : A \to C,$$



- a *terminal object* 1:

$$\forall A, \qquad A \dashrightarrow 1$$

- which is *closed*:

$$\frac{A \times B \to C}{A \to (B \Rightarrow C)}$$

A model of Martin-Löf type theory

The traditional models of Martin-Löf type theory are given by

Definition

A **locally cartesian closed category** is a category $\mathcal{C}$ in which for every object A the slice category $\mathcal{C}/A$ is cartesian closed.

A model of Martin-Löf type theory

The traditional models of Martin-Löf type theory are given by

Definition

A **locally cartesian closed category** is a category $\mathcal{C}$ in which for every object A the slice category $\mathcal{C}/A$ is cartesian closed.

Theorem

An LCCC is a category with pullbacks in which for every $f : A \to B$, the base change functor $f^* : \mathcal{C}/B \to \mathcal{C}/A$ has a right adjoint $\Pi_f : \mathcal{C}/A \to \mathcal{C}/B$.

A model of Martin-Löf type theory

The traditional models of Martin-Löf type theory are given by

Definition

A **locally cartesian closed category** is a category $\mathcal{C}$ in which for every object A the slice category $\mathcal{C}/A$ is cartesian closed.

Theorem

An LCCC is a category with pullbacks in which for every $f : A \to B$, the base change functor $f^* : \mathcal{C}/B \to \mathcal{C}/A$ has a right adjoint $\Pi_f : \mathcal{C}/A \to \mathcal{C}/B$.

Example

$$\frac{\Gamma, x : A \vdash B(x) : \text{type}}{\Gamma \vdash \Pi_{x:A}.B(x) : \text{type}}$$

Problem

Every LCCC is also a model of MLTT with the rule of *extensionality*:

$$\frac{\vdash p : \mathsf{Id}_A(a, b)}{\vdash a = b : A}$$

...and type checking is indecidable in extensional MLTT!

We explain here that Quillen model categories model identity types in Martin-Löf type theory:

$$F : \mathcal{M} \to \mathcal{Q}$$

Which provides non-extensional models.

We explain here that Quillen model categories model identity types in Martin-Löf type theory:

$$F : \mathcal{M} \to \mathcal{Q}$$

Which provides non-extensional models.

The idea here is that identity types behave like homotopies between topological spaces.

Homotopy

A homotopy between two continuous functions $f, g : A \to B$ between topological spaces A and B is a continuous function

$$h : I \times A \to B$$

where $I = [0, 1]$ such that $h(0, x) = f(x)$ and $h(1, x) = g(x)$.

Homotopy

A homotopy between two continuous functions $f, g : A \to B$ between topological spaces A and B is a continuous function

$$h : I \times A \to B$$

where $I = [0, 1]$ such that $h(0, x) = f(x)$ and $h(1, x) = g(x)$.

Two spaces A and B are *homotopy equivalent* when there exists maps $f : A \to B$ and $g : B \to A$ such that

$$g \circ f \sim \mathrm{id}_A \qquad f \circ g \sim \mathrm{id}_B$$

Ex: square $\approx$ circle, coffee mug $\approx$ donut, etc.

Homotopies

Suppose given a topological space T.

- A **path** in T is a continuous function $\pi : I \to T$, where $I = [0, 1]$.

Homotopies

Suppose given a topological space T.

- A **path** in T is a continuous function $\pi : I \to T$, where $I = [0, 1]$.
- An **homotopy** between two paths π and ρ is a continuous function

$$h : I \to (I \Rightarrow T) \qquad \text{such that} \qquad h(0) = \pi \quad \text{and} \quad h(1) = \rho$$

Homotopies

Suppose given a topological space T.

- A **path** in T is a continuous function $\pi : I \to T$, where $I = [0, 1]$.
- An **homotopy** between two paths π and ρ is a continuous function

$$h : I \to (I \Rightarrow T) \qquad \text{such that} \qquad h(0) = \pi \quad \text{and} \quad h(1) = \rho$$

- An **homotopy between homotopies** h and k is a continuous function

$$h : I \to (I \Rightarrow (I \Rightarrow T)) \qquad \text{such that} \qquad h(0) = h \quad \text{and} \quad h(1) = k$$

Homotopies

Suppose given a topological space T.

- A **path** in T is a continuous function $\pi : I \to T$, where $I = [0, 1]$.
- An **homotopy** between two paths π and ρ is a continuous function

$$h : I \to (I \Rightarrow T) \qquad \text{such that} \qquad h(0) = \pi \quad \text{and} \quad h(1) = \rho$$

- An **homotopy between homotopies** h and k is a continuous function

$$h : I \to (I \Rightarrow (I \Rightarrow T)) \qquad \text{such that} \qquad h(0) = h \quad \text{and} \quad h(1) = k$$

- etc.

Modeling MLTT

We interpret

- a type $\vdash A$: type as a topological space
- a term $\vdash x : A$ as a point in A
- a term $p : \mathsf{Id}_A(a, b)$ as a path $a \to b$
- a term $s : \mathsf{Id}_{\mathsf{Id}(a,b)}(p, q)$ as an homotopy $a \underset{q}{\overset{p}{\underset{s\Downarrow}{\rightrightarrows}}} b$
- etc.

Dependent types

As in the case of LCCC we interpret a dependent type

$$x : A \vdash B(x) : \text{type}$$

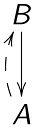
as a continuous map

$$\begin{array}{c} B \\ \downarrow \\ A \end{array}$$

As in the case of LCCC we interpret a dependent type

$$x : A \vdash B(x) : \text{type}$$

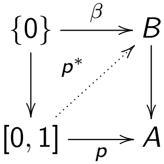
as a continuous map



and a term $x : A \vdash f : B(x)$ as a section of this map.

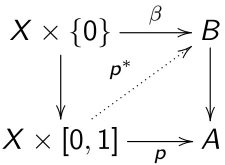
Dependent types and equality

The maps interpreting types should have the *homotopy lifting property*:



Dependent types and equality

The maps interpreting types should have the *homotopy lifting property*:



Maps like this are often called *fibrations*.

Dependent types and equality

The maps interpreting types should have the *homotopy lifting property*:

$$\begin{array}{ccc} X \times \{0\} & \xrightarrow{\beta} & B \\ \downarrow & \overset{p^*}{\nearrow} & \downarrow \\ X \times [0,1] & \xrightarrow[p]{} & A \end{array}$$

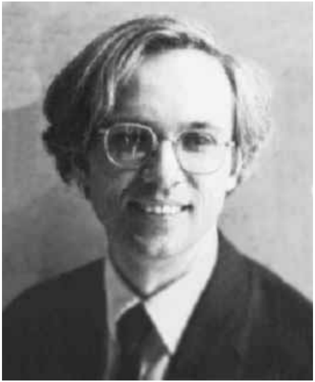
Maps like this are often called *fibrations*.

Ex: the interpretation of $x, y : A \vdash \mathsf{Id}_A(x, y)$ is a map

$$\begin{array}{c} A^I \\ \downarrow \\ A \times A \end{array}$$

Lifting properties

Homotopy is more generally carried on in Quillen model categories.



Lifting properties

Homotopy is more generally carried on in Quillen model categories.

Definition

Given maps $f : A \to B$ and $g : C \to D$, f has the *left lifting property* wrt g when every commutative square

$$\begin{array}{ccc} A & \xrightarrow{h} & C \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[k]{} & D \end{array}$$

Homotopy is more generally carried on in Quillen model categories.

Definition

Given maps $f : A \to B$ and $g : C \to D$, f has the *left lifting property* wrt g when every commutative square

$$\begin{array}{ccc} A & \xrightarrow{h} & C \\ {\scriptstyle f}\downarrow & \overset{l}{\nearrow} & \downarrow{\scriptstyle g} \\ B & \xrightarrow[k]{} & D \end{array}$$

admits a lifting.

Lifting properties

Homotopy is more generally carried on in Quillen model categories.

Definition

Given maps $f : A \to B$ and $g : C \to D$, f has the *left lifting property* wrt g when every commutative square

$$\begin{array}{ccc} A & \xrightarrow{h} & C \\ {\scriptstyle f}\downarrow & \overset{l}{\nearrow} & \downarrow{\scriptstyle g} \\ B & \xrightarrow[k]{} & D \end{array}$$

admits a lifting.

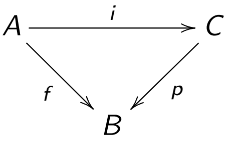
Given a class $\mathfrak{L}$ of maps, we write ${}^{\perp}\mathfrak{L}$ for the class of maps which have LLP wrt every map in $\mathfrak{L}$ (and similarly $\mathfrak{L}^{\perp}$ for RLP).

Weak factorization systems

Definition

A *weak factorization system* $(\mathfrak{L}, \mathfrak{R})$ consists of two classes of maps such that

1. every map $f : A \to B$ factors as

   with $i \in \mathfrak{L}$ and $p \in \mathfrak{R}$
2. $\mathfrak{L}^{\perp} = \mathfrak{R}$ and $\mathfrak{L} = {}^{\perp}\mathfrak{R}$

Model categories

Definition

A *model category* consists of $\mathcal{C}$ together with subcategories

- $\mathfrak{F}$: *fibrations*
- $\mathfrak{C}$: *cofibrations*
- $\mathfrak{W}$: *weak equivalences*

such that

1. three for two
2. both $(\mathfrak{C}, \mathfrak{W} \cap \mathfrak{F})$ and $(\mathfrak{C} \cap \mathfrak{W}, \mathfrak{F})$ are weak factorization systems.

Definition

A *model category* consists of $\mathcal{C}$ together with subcategories

- $\mathfrak{F}$: *fibrations*
- $\mathfrak{C}$: *cofibrations*
- $\mathfrak{W}$: *weak equivalences*

such that

1. three for two
2. both $(\mathfrak{C}, \mathfrak{W} \cap \mathfrak{F})$ and $(\mathfrak{C} \cap \mathfrak{W}, \mathfrak{F})$ are weak factorization systems.

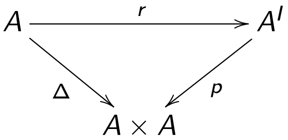
Example

On **Top**:

- generating cofibrations are inclusions $i : \Delta^n \to \Delta^n \times I$,
- fibrations are RLP of generating cofibrations (Serre fibrations),
- weak equivalences are weak homotopy equivalences.

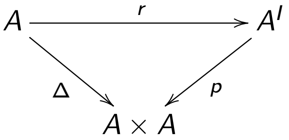
Definition

A *(very good) path object* A^I for an object A consists of a factorization

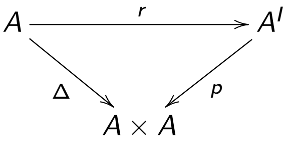


with r acyclic cofibration and p fibration.

Interpretation of MLTT



Interpretation of MLTT

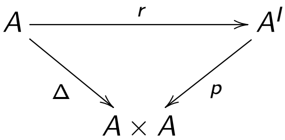


- Type formation rule:

$$\frac{\vdash a : A \qquad \vdash b : A}{\vdash \mathsf{Id}_A(a, b) : \mathsf{type}}$$

Id_A is interpreted as p

Interpretation of MLTT



- Type formation rule:
- Introduction rule:

$$\frac{\vdash a : A}{\vdash r_A(a) : \mathrm{Id}_A(a, a)}$$

r_A is interpreted as r

Interpretation of MLTT

- Type formation rule:
- Introduction rule:
- Elimination rule:

$$\frac{x : A, y : A, z : \mathsf{Id}_A(x, y) \vdash D(x, y, z) : \mathsf{type} \qquad x : A \vdash d(x) : D(x, x, r_A(x))}{x : A, y : A, z : \mathsf{Id}_A(x, y) \vdash J_{A,D}(d, x, y, z) : D(x, y, z)}$$

Interpretation of MLTT

- Type formation rule:
- Introduction rule:
- Elimination rule:

$$\frac{\begin{array}{c} x : A, y : A, z : \mathrm{Id}_A(x, y) \vdash D(x, y, z) : \mathrm{type} \\ x : A \vdash d(x) : D(x, x, r_A(x)) \end{array}}{x : A, y : A, z : \mathrm{Id}_A(x, y) \vdash J_{A,D}(d, x, y, z) : D(x, y, z)}$$

- Conversion rule:

$$\frac{\begin{array}{c} x : A, y : A, z : \mathrm{Id}_A(x, y) \vdash D(x, y, z) : \mathrm{type} \\ x : A \vdash d(x) : D(x, x, r_A(x)) \end{array}}{x : A \vdash J_{A,D}(d, x, x, r_A(x)) = d(x) : D(x, x, r_A(x))}$$

The current state of things

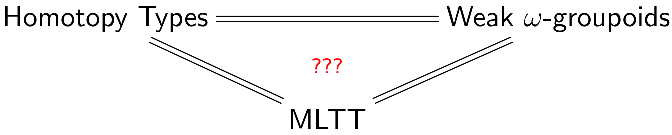
Theorem (Awodey & Warren)

MLTT can be interpreted in any model category.

Theorem (Gambino & Garner)

The interpretation is complete.

The Homotopy Hypothesis



Towards directed algebraic topology?

We could think of a directed variant:

- replace equality by a reduction relation:

$$f \rightsquigarrow g \qquad \Rightarrow \qquad \text{there is a directed path from } f \text{ to } g$$

Towards directed algebraic topology?

We could think of a directed variant:

- replace equality by a reduction relation:

$$f \rightsquigarrow g \qquad \Rightarrow \qquad \text{there is a directed path from } f \text{ to } g$$

- the reduction should be compatible with identity:

$$r : \mathsf{Id}(f, f') \qquad \Rightarrow \qquad \exists g', \quad \exists s : \mathsf{Id}(g, g') \quad \text{and} \quad g \rightsquigarrow g'$$

$$\begin{array}{ccc} f & = \!=\! = & f' \\ \wr\!\downarrow & & \wr\!\downarrow \\ g & = = = & g' \end{array}$$

We can "translate continuously" the directed path $f \rightsquigarrow g$ into the directed path $f' \rightsquigarrow g$