Syntax and Semantics of Dependent Type Theories

Nicola Gambino

University of Leeds

HTT-UF Summer School
Leeds, 24th June 2019

Overview

1. The syntax of dependent type theories
2. Models of dependent type theories
3. Coherence issues

Part 1:
The syntax of dependent type theories

First-order theories vs dependent type theories (I)

Usual steps to set up a first-order theory:

(1) fix a language L
(2) define inductively the set of well-formed terms
(3) define inductively the set of well-formed formulas
(4) give the axioms for the theory
(5) define inductively the set of theorems of the theory.

Note. No vicious circle: each step depends only on the previous ones (e.g. the axioms of a theory do not modify the set of well-formed terms).

First-order theories vs dependent type theories (II)

Problem. This approach does not work for dependent type theories:

- deduction rules of a dependent type theory specify how the well-formed term expressions are built, e.g.

$$\frac{a : A}{\mathsf{inl}(A, B, a) : A + B}$$

- the sets of terms and of types cannot be defined independently, e.g.

$$\mathsf{Id}(A, a, b)\,, \quad \mathsf{nil}(A)$$

Setting up a dependent type theory

Solution (Martin-Löf, Aczel, . . .)

(1) fix a signature (*a notion that will be defined in the next slides*),
(2) define inductively the set of raw expressions,
(3) give the deduction rules of the dependent type theory,
(4) define inductively the set of theorems of the theory,
(5) the theorems isolate the well-formed expressions.

We will illustrate this in general and in one example.

Signatures (I)

Recall that a signature for an algebraic theory (e.g. theory of groups) consists of

- a set of operations,
- an assignment of an arity to each operation.

Here, an arity is just a natural number (the number of arguments of the operation).

For dependent type theories, we need more a complex notion of arity, to account for **variable binding** operations, e.g. λ , Π, Σ.

Signatures (II)

Definition. An **arity** is a tuple of the form

$$\big((n_1, \varepsilon_1), \ldots, (n_k, \varepsilon_k), \varepsilon\big)\,,$$

where $k \in \mathbb{N}$, $n_1, \ldots, n_k \in \mathbb{N}$ and $\varepsilon_1, \ldots, \varepsilon_k, \varepsilon$ are either 0 or 1.

Notation. When $k = 0$, we will write (ε).

Idea.

- An operation of arity as above takes k arguments and binds n_i variables in the i-th argument.
- The $\varepsilon_1, \ldots, \varepsilon_k, \varepsilon$ keep track of the distinction between term expressions (0-expressions) and type expressions (1-expressions).

Definition. A **signature** Σ is a set of pairs (s, α) where α is an arity. If $(s, \alpha) \in \Sigma$, we call s a **symbol of arity** α.

Raw expressions

Fix

- an infinite set of variables $\{x_0, x_1, \ldots\}$,
- a signature Σ.

Define the sets of 0-expressions and 1-expressions by the rules:

(i) every variable is a 0-expression,

(ii) if s has arity $((n_1, \varepsilon_1), \ldots, (n_k, \varepsilon_k), \varepsilon)$ and M_i is an ε_i-expression and $\vec{x}_i$ is a vector of n_i distinct variables for $i = 1, \ldots, k$, then

$$s\big((\vec{x}_1)M_1, \ldots, (\vec{x}_k)M_k\big)$$

is an ε-expression.

Example

The signature for the type theory $\mathbf{ML_1}$ includes the symbols

- Nat of arity (1)
- succ of arity $((0,0),0)$
- nil of arity $((0,1),0)$
- λ of arity $((0,1),(1,1),(1,0),0)$

Thus, we have that

- Nat is a 1-expression
- succ(x) is a 0-expression
- nil(Nat) is a 0-expression
- $\lambda(\text{Nat},(x)\text{Nat},(x)x)$ is a 0-expression, usually written $(\lambda x : \text{Nat})x$.

Judgements

A **judgement** has one of the following four forms:

- $A : \mathrm{type}$
- $A = B : \mathrm{type}$
- $a : A$
- $a = b : A$

Here, A, B stand for 1-expressions and a, b for 0-expressions.

Contexts and hypothetical judgements

A **context** Γ is a sequence of the form

$$\Gamma = (x_0 : A_0 \, , x_1 : A_1 \, , \ldots , x_n : A_n)$$

where

- $x_0, \ldots, x_n$ are distinct variables
- $A_1, \ldots, A_n$ are 1-expressions.

A **hypothetical judgement** has the form

$$\Gamma \vdash J$$

where Γ is a context and J is a judgement.

Deduction rules

A **deduction rule** has the form

$$\frac{\Gamma_1 \vdash J_1 \qquad \cdots \qquad \Gamma_n \vdash J_n}{\Gamma \vdash J}$$

where $\Gamma_1 \vdash J_1, \ldots, \Gamma_n \vdash J_n, \Gamma \vdash J$ are hypothetical judgements.

A **dependent type theory** is given by specifying

- a signature,
- a set of deduction rules.

Derivability is defined in the standard way.

Well-formed expressions

Definition. Let T be a dependent type theory over a signature Σ.

- If $\Gamma \vdash A : \text{type}$ is derivable, then we say that A is a **well-formed type** in context Γ.
- If $\Gamma \vdash a : A$ is derivable, then we say that a is a **well-formed element** of type A in context Γ.
- If $\Gamma \vdash A = B : \text{type}$, then we say that A and B are **judgementally equal well-formed types** in context Γ.
- If $\Gamma \vdash a = b : A$, then we say that a and b are **judgementally equal well-formed elements** of type A in context Γ.

The dependent type theory $\mathbf{ML_1}$

$\mathbf{ML_1}$ has the following forms of type:

$$0\,,\quad 1\,,\quad \mathsf{Bool}\,,\quad \mathsf{Nat}$$

$$\mathsf{List}(A)\,,\quad A + B$$

$$\mathsf{Id}(A, a, b)$$

$$(\Sigma x : A)B\,,\quad (\Pi x : A)B$$

$$\mathsf{U}$$

The signature of $\mathbf{ML}_1$ (I)

Symbol	*Arity*
$0, 1, \mathsf{Bool}, \mathsf{Nat}, \mathsf{U}$	(1)
List	$((0,1),1)$
$+$	$((0,1),(0,1),1)$
Id	$((0,1),(0,0),(0,0),1)$
Π	$((0,1),(1,1),1)$
Σ	$((0,1),(1,1),1)$
T	$((0,0),1)$

Notation. We write

- $A + B$ for $+(A, B)$
- $(\Pi x : A)B$ for $\Pi(A, (x)B)$
- $(\Sigma x : A)B$ for $\Sigma(A, (x)B)$.

The signature of $\mathbf{ML}_1$ (II)

The other symbols of the signature:

$*$	refl
unitrec	J
true	λ
false	app
boolrec	pair
0	split
succ	$\mathsf{Bool}^{\mathsf{U}}$
natrec	$\mathsf{Nat}^{\mathsf{U}}$
nil	0^{U}
cons	$\mathsf{List}^{\mathsf{U}}$
listrec	$+^{\mathsf{U}}$
inl	Id^{U}
inr	Π^{U}
case	Σ^{U}

The deduction rules of $\mathbf{ML_1}$

(1) General rules

(2) Rules for the forms of type of $\mathbf{ML_1}$. For each one, we have

- formation rules
- introduction rules
- elimination rules
- computation rules

General deduction rules

Standard rules regarding context formation, substitution, equality.

Examples.

$$\frac{\Gamma, \Delta \vdash J \qquad \Gamma \vdash A : \text{type}}{\Gamma, x : A, \Delta \vdash J}\, x \notin \mathrm{FV}(\Gamma) \cup \mathrm{FV}(\Delta)$$

$$\frac{x : A, \Gamma \vdash J \qquad a : A}{\Gamma[a/x] \vdash J[a/x]}$$

$$\frac{a : A \quad A = B : \text{type}}{a : B}$$

Notation. $\Gamma[a/x]$ and $J[a/x]$ denote the substitution of a for x in Γ and J, respectively.

The empty type

Formation rule.

$$0 : \text{type}$$

No introduction rules.

Elimination rule.

$$\frac{c : 0 \quad u : 0 \vdash E : \text{type}}{\text{emptyrec}(c, (u)E) : E[c/u]}$$

No computation rules.

The unit type (I)

Formation rule.

$$1 : \text{type}$$

Introduction rules.

$$* : 1$$

The unit type (II)

Elimination rules.

$$\frac{c : 1 \quad u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[*/u]}{\mathsf{unitrec}(c, (u)E, d) : E[c/u]}$$

Computation rule.

$$\frac{u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[*/u]}{\mathsf{unitrec}(*, (u)E, d) = d : E[*/u]}$$

The type of Boolean truth values (I)

Formation rule.

$$\mathsf{Bool} : \mathsf{type}$$

Introduction rules.

$$\mathsf{true} : \mathsf{Bool} \qquad \mathsf{false} : \mathsf{Bool}$$

The type of Boolean truth values (II)

Elimination rules.

$$\frac{c : \mathsf{Bool} \quad u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[\mathsf{true}/u] \quad e : E[\mathsf{false}/u]}{\mathsf{boolrec}(c, (u)E, d, e) : E[c/u]}$$

Computation rules.

$$\frac{u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[\mathsf{true}/u] \quad e : E[\mathsf{false}/u]}{\mathsf{boolrec}(\mathsf{true}, (u)E, d, e) = d : E[\mathsf{true}/u]}$$

$$\frac{u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[\mathsf{true}/u] \quad e : E[\mathsf{false}/u]}{\mathsf{boolrec}(\mathsf{false}, (u)E, d, e) = e : E[\mathsf{false}/u]}$$

The type of natural numbers (I)

Formation rule.

$$\mathsf{Nat} : \mathsf{type}$$

Introduction rules.

$$0 : \mathsf{Nat} \qquad \frac{n : \mathsf{Nat}}{\mathsf{succ}(n) : \mathsf{Nat}}$$

The type of natural numbers (II)

Elimination rule.

$$\frac{c : \mathsf{Nat} \quad u : \mathsf{Nat} \vdash E : \mathsf{type} \quad d : E[0/u] \quad x : \mathsf{Nat}, y : E[x/u] \vdash e : E[\mathsf{succ}(x)/u]}{\mathsf{natrec}(c, (u)E, d, (x, y)e) : E[c/u]}$$

Computation rules.

$$\frac{u : \mathsf{Nat} \vdash E : \mathsf{type} \quad d : E[0/u] \quad x : \mathsf{Nat}, y : E[x/u] \vdash e : E[\mathsf{succ}(x)/u]}{\mathsf{natrec}(0, (u)E, d, (x, y)e) = d : E[0/u]}$$

$$\frac{c : \mathsf{Nat} \quad u : \mathsf{Nat} \vdash E : \mathsf{type} \quad d : E[0/u] \quad x : \mathsf{Nat}, y : E[x/u] \vdash e : E[\mathsf{succ}(x)/u]}{\begin{array}{l}\mathsf{natrec}(\mathsf{succ}(c), (u)E, d, (x, y)e) \\ \quad = e[c/x, \mathsf{natrec}(c, (u)E, d, (x, y)e)/y] : E[\mathsf{succ}(c)/u]\end{array}}$$

List types (I)

Formation rule.

$$\frac{A : \text{type}}{\text{List}(A) : \text{type}}$$

Introduction rules.

$$\frac{A : \text{type}}{\text{nil}(A) : \text{List}(A)} \qquad \frac{a : A \qquad \ell : \text{List}(A)}{\text{cons}(A, a, \ell) : \text{List}(A)}$$

Notation. We will write nil and $\text{cons}(a, \ell)$ for $\text{nil}(A)$ and $\text{cons}(A, a, \ell)$.

List types (II)

Elimination rule.

$$\frac{\begin{array}{l}\ell : \mathsf{List}(A)\\ u : \mathsf{List}(A) \vdash E : \mathsf{type}\\ d : E[\mathsf{nil}/u]\\ x : A, y : \mathsf{List}(A), z : E[y/u] \vdash e : E[\mathsf{cons}(x,y)/u]\end{array}}{\mathsf{listrec}(\ell, (u)E, d, (x,y,z)e) : E[\ell/u]}$$

Computation rules.

$$\frac{\begin{array}{l}u : \mathsf{List}(A) \vdash E : \mathsf{type}\\ d : E[\mathsf{nil}/u]\\ x : \mathsf{List}(A), y : A, z : E[y/u] \vdash e : E[\mathsf{cons}(x,y)/u]\end{array}}{\mathsf{listrec}(\mathsf{nil}, (u)E, d, (x,y,z)e) = d : E[\mathsf{nil}/u]}$$

$$\frac{\begin{array}{l}\ell : \mathsf{List}(A)\\ a : A\\ u : \mathsf{List}(A) \vdash E : \mathsf{type}\\ d : E[\mathsf{nil}/u]\\ x : \mathsf{List}(A), y : A, z : E[y/u] \vdash e : E[\mathsf{cons}(x,y)/u]\end{array}}{\begin{array}{l}\mathsf{listrec}(\mathsf{cons}(a,\ell), (u)E, d, (x,y,z)e) =\\ \quad e[\ell/x, a/y, \mathsf{listrec}(\ell, (u)E, d, (x,y,z)e/u] : E[\mathsf{cons}(a,\ell)/u]\end{array}}$$

Sum types (I)

Formation rule.

$$\frac{A : \text{type} \quad B : \text{type}}{A + B : \text{type}}$$

Introduction rules.

$$\frac{a : A}{\text{inl}(A, B, a) : A + B} \qquad \frac{b : B}{\text{inr}(A, B, b) : A + B}$$

Notation. We write $\text{inl}(a)$, $\text{inr}(b)$ for $\text{inl}(A, B, a)$ and $\text{inr}(A, B, b)$.

Sum types (II)

Elimination rule.

$$\frac{c : A + B \quad u : A + B \vdash E : \mathsf{type} \quad x : A \vdash d : E[\mathsf{inl}(x)/u] \quad y : B \vdash e : E[\mathsf{inr}(y)u]}{\mathsf{case}(c, (u)E, (x)d, (y)e) : E[c/u]}$$

Computation rules.

$$\frac{a : A \quad u : A + B \vdash E : \mathsf{type} \quad x : A \vdash d : E[\mathsf{inl}(x)/u] \quad y : B \vdash e : E[\mathsf{inr}(y)u]}{\mathsf{case}(\mathsf{inl}(a), (u)E, (x)d, (y)e) = d[a/x] : E[\mathsf{inl}(a)/u]}$$

$$\frac{b : B \quad u : A + B \vdash E : \mathsf{type} \quad x : A \vdash d : E[\mathsf{inl}(x)/u] \quad y : B \vdash e : E[\mathsf{inr}(y)u]}{\mathsf{case}(\mathsf{inr}(b), (u)E, (x)d, (y)e) = e[b/y] : E[\mathsf{inr}(b)/u]}$$

Identity types (I)

Formation rule.

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

Introduction rule.

$$\frac{a : A}{\mathsf{refl}(A, a) : \mathsf{Id}(A, a, a)}$$

Notation. We write $\mathsf{Id}_A(a, b)$ for $\mathsf{Id}(A, a, b)$, $\mathsf{refl}(a)$ for $\mathsf{refl}(A, a)$.

Identity types (II)

Elimination rule.

$$\frac{p : \mathrm{Id}_A(a,b) \quad x : A, y : A, u : \mathrm{Id}_A(x,y) \vdash E : \mathrm{type} \quad x : A \vdash e : E[x/y, \mathrm{refl}(x)/u]}{\mathrm{J}(a,b,p,(x,y,u)E,(x)e) : E[a/x, b/y, p/u]}$$

Computation rule.

$$\frac{a : A \quad x : A, y : A, u : \mathrm{Id}_A(x,y) \vdash E : \mathrm{type} \quad x : A \vdash e : E[x/y, \mathrm{refl}(x)/u]}{\mathrm{J}(a,a,\mathrm{refl}(a),(x,y,u)E,(x)e) = e[a/x] : E[a/x, a/y, \mathrm{refl}(a)/u]}$$

Note. $E[y/x, \mathrm{refl}(x)/u] = E[x/x, y/x, \mathrm{refl}(x)/u]$.

Dependent pair types (I)

Formation rule.

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

Introduction rule.

$$\frac{a : A \qquad b : B[a/x]}{\text{pair}(A, (x)B, a, b) : \Sigma(x : A)B}$$

Notation. We will write $\text{pair}(a, b)$ for $\text{pair}(A, (x)B, a, b)$.

Special case. If $x \notin \mathrm{FV}(B)$, write $A \times B$ for $\Sigma(x : A)B$.

Dependent pair types (II)

Elimination rule.

$$\frac{c : (\Sigma x : A)B(x) \quad u : (\Sigma x : A)B(x) \vdash E : \mathsf{type} \quad x : A, y : B \vdash d : E[\mathsf{pair}(x, y)/u]}{\mathsf{split}(c, (u)E, (x, y)d) : E[c/u]}$$

Computation rule.

$$\frac{a : A \quad b : B[a/x] \quad u : (\Sigma x : A)B(x) \vdash E : \mathsf{type} \quad x : A, y : B \vdash d : E[\mathsf{pair}(x, y)/u]}{\mathsf{split}(\mathsf{pair}(a, b), (u)E, (x, y)d) = d[a/x, b/y] : E[\mathsf{pair}(a, b)/u]}$$

Dependent function types (I)

Formation rule.

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

Introduction rule.

$$\frac{x : A \vdash b : B}{\lambda(A, (x)B, (x)b) : \Pi(x : A)B}$$

Notation. We write $\lambda x.b$ for $\lambda(A, (x)B, (x)b)$.

Special case. If $x \notin \mathrm{FV}(B)$, write $A \to B$ for $\Pi(x : A)B$.

Dependent function types (II)

Elimination rule.

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

Computation rule.

$$\frac{x : A \vdash b : B \qquad a : A}{\mathsf{app}(\lambda x.b, a) = b[a/x] : B[a/x]}$$

A type universei

Formation rule.

$$\mathsf{U} : \mathsf{type}$$

.

Elimination rule.

$$\frac{A : \mathsf{U}}{\mathsf{T}(A) : \mathsf{type}}$$

This is the so-called formulation à la Tarski.

A type universe (II)

Introduction and computation rules.

$$0^{\mathsf{U}} : \mathsf{U} \qquad \mathsf{T}(0^{\mathsf{U}}) = 0 : \mathsf{type}$$

$$1^{\mathsf{U}} : \mathsf{U} \qquad \mathsf{T}(1^{\mathsf{U}}) = 1 : \mathsf{type}$$

$$\mathsf{Bool}^{\mathsf{U}} : \mathsf{U} \qquad \mathsf{T}(\mathsf{Bool}^{\mathsf{U}}) = \mathsf{Bool} : \mathsf{type}$$

$$\mathsf{Nat}^{\mathsf{U}} : \mathsf{U} \qquad \mathsf{T}(\mathsf{Nat}^{\mathsf{U}}) = \mathsf{Nat} : \mathsf{type}$$

A type universe (III)

Introduction and computation rules.

$$\frac{A : \mathsf{U} \quad B : \mathsf{U}}{A +^{\mathsf{U}} B : \mathsf{U}} \qquad \frac{A : \mathsf{U} \quad B : \mathsf{U}}{\mathsf{T}(A +^{\mathsf{U}} B) = \mathsf{T}(A) + \mathsf{T}(B) : \mathsf{type}}$$

$$\frac{A : \mathsf{U}}{\mathsf{List}^{\mathsf{U}}(A) : \mathsf{U}} \qquad \frac{A : \mathsf{U}}{\mathsf{T}(\mathsf{List}^{\mathsf{U}}(A)) = \mathsf{List}(\mathsf{T}(A)) : \mathsf{type}}$$

A type universe (IV)

Introduction and computation rules.

$$\frac{A : \mathsf{U} \quad a : \mathsf{T}(A) \quad b : \mathsf{T}(A)}{\mathsf{Id}^{\mathsf{U}}(A, a, b) : \mathsf{U}} \qquad \frac{A : \mathsf{U} \quad a : \mathsf{T}(A) \quad b : \mathsf{T}(A)}{\mathsf{T}(\mathsf{Id}^{\mathsf{U}}(A, a, b)) = \mathsf{Id}(\mathsf{T}(A), a, b) : \mathsf{type}}$$

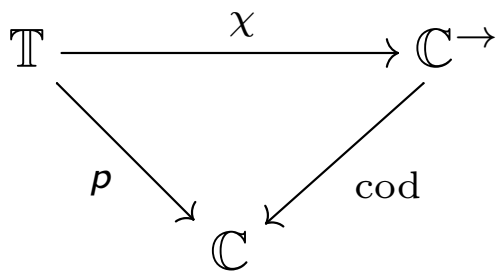
$$\frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\Pi^{\mathsf{U}}(A, (x)B) : \mathsf{U}} \qquad \frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\mathsf{T}(\Pi^{\mathsf{U}}(A, (x)B)) = \Pi(x : \mathsf{T}(A))\mathsf{T}(B) : \mathsf{type}}$$

$$\frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\Sigma^{\mathsf{U}}(A, (x)B) : \mathsf{U}} \qquad \frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\mathsf{T}(\Sigma^{\mathsf{U}}(A, (x)B)) = \Sigma(x : \mathsf{T}(A))\mathsf{T}(B) : \mathsf{type}}$$

Part 2:
Models of dependent type theories

Comprehension categories

Definition. A (split) comprehension category consists of



where

- p is a (split) Grothendieck fibration
- χ sends Cartesian morphisms to pullback squares

Example: the syntactic category (I)

Let **Ctx** be the category of contexts of $\mathbf{ML_1}$, *i.e.*

- objects are contexts, *e.g.*

$$\Gamma = (x_0 : A_0 , , \ldots , x_n : A_n)$$

- morphisms $\sigma : \Delta \to \Gamma$ are substitutions, *i.e.* sequences

$$\sigma = (a_0, \ldots, a_n)$$

  such that

$$\begin{gathered} \Delta \vdash a_0 : A_0 \\ \Delta \vdash a_1 : A_1[a_0/x_0] \\ \vdots \\ \Delta \vdash a_n : A_n[a_0/x_0, \ldots, a_{n-1}/x_{n-1}] \end{gathered}$$

Example: the syntactic category (II)

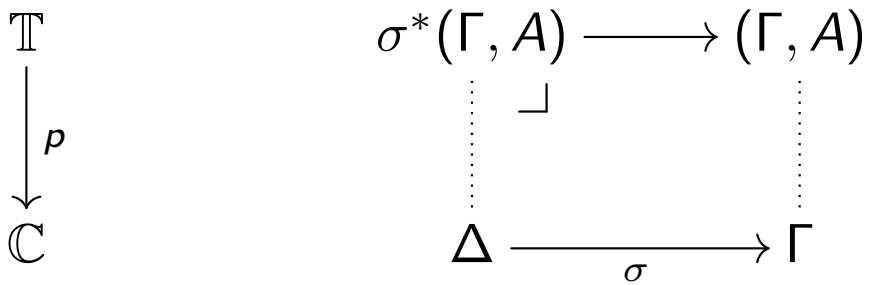
Let **Ty** the category of types-in-context, *i.e.*

- objects are pairs (Γ, A), where $\Gamma \vdash A$: type
- morphisms $(\Delta, B) \to (\Gamma, A)$ are context morphisms

$$(\Delta, y : B) \to (\Gamma, x : A)\,.$$

Let p: **Ty** $\to$ **Ctx** be the first projection, mapping (Γ, A) to Γ.

Fact. The functor p is a split Grothendieck fibration:



where $\sigma^*(\Gamma, A) = (\Delta, A[\sigma])$.

Example: the syntactic category (III)

The functor $\chi \colon \mathbf{Ty} \to \mathbf{Ctx}^{\to}$ sends (Γ, A) to the evident map

$$\chi_A \colon (\Gamma, x \colon A) \to (\Gamma)$$

Proposition. The functor χ sends Cartesian arrows to pullbacks

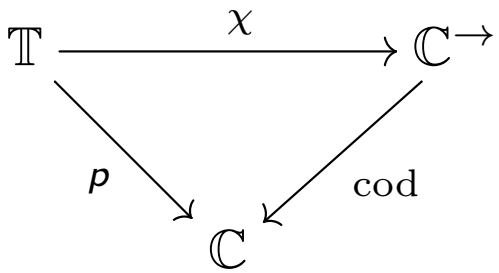
$$\begin{array}{ccc} (\Delta, x : A[\sigma]) & \longrightarrow & (\Gamma, x \colon A) \\ {\scriptstyle \chi_{A[\sigma]}}\downarrow & \lrcorner & \downarrow{\scriptstyle \chi_A} \\ \Delta & \xrightarrow[\sigma]{} & \Gamma \end{array}$$

so we have a split comprehension category

$$\begin{array}{ccc} \mathbf{Ty} & \xrightarrow{\chi} & \mathbf{Ctx}^{\to} \\ & {\scriptstyle p}\searrow \quad \swarrow{\scriptstyle \mathrm{cod}} & \\ & \mathbf{Ctx} & \end{array}$$

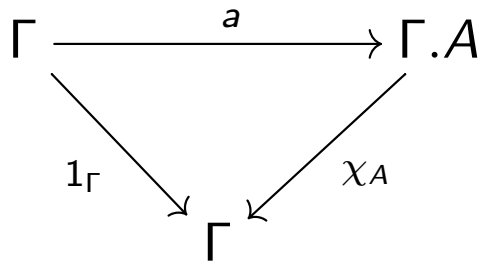
Notation and terminology

Given a general split comprehension category



For $\Gamma \in \mathbb{C}$ and $A \in \mathbb{T}_\Gamma$,

- we write $\chi_A \colon \Gamma.A \to \Gamma$ for $\chi(A)$,
- we think of a section



  as a term $\Gamma \vdash a \colon A$.

Modelling $\mathbf{ML_1}$

Split comprehension categories model only the core of $\mathbf{ML_1}$, *i.e.*

- structural rules,
- substitution.

Note. The Grothendieck fibration p being split corresponds to strict associativity and unitality of substitution.

We need additional structure and axioms to model each form of type.

We only discuss the case of Id-types.

Id-types in a comprehension category

For every $\Gamma \in \mathbb{C}$, $A \in \mathbb{T}_\Gamma$, we need

- an object $\mathrm{Id}_A \in \mathbb{T}_{\Gamma.A.A}$
- a factorisation of the diagonal

$$\Gamma.A \xrightarrow{r_A} \Gamma.A.A.\mathrm{Id}_A \longrightarrow \Gamma.A.A$$

- for each $E \in \mathbb{T}_{\Gamma.A.A.\mathrm{Id}_A}$ and e making

$$\begin{array}{ccc} \Gamma.A & \xrightarrow{e} & \Gamma.A.A.\mathrm{Id}_A.E \\ {\scriptstyle r_A}\downarrow & & \downarrow \\ \Gamma.A.A.\mathrm{Id}_A & \longrightarrow & \Gamma.A.A.\mathrm{Id}_A \end{array}$$

  commute, a diagonal filler $j_{A,E,e}$.

such that, for all $\sigma : \Delta \to \Gamma$,

- $\mathrm{Id}_A[\sigma] = \mathrm{Id}_{A[\sigma]}\,, \quad r_A[\sigma] = r_{A[\sigma]} \quad , j_{A,E,e}[\sigma] = j_{A[\sigma],E[\sigma],e[\sigma]}$

Part 3:
How to construct models

Coherence issues

Split comprehension categories modelling $\mathbf{ML}_1$ occur rarely in practice.

Two issues:

(1) Grothendieck fibrations need not be split
(2) Strict stability under reindexing is hard to satisfy

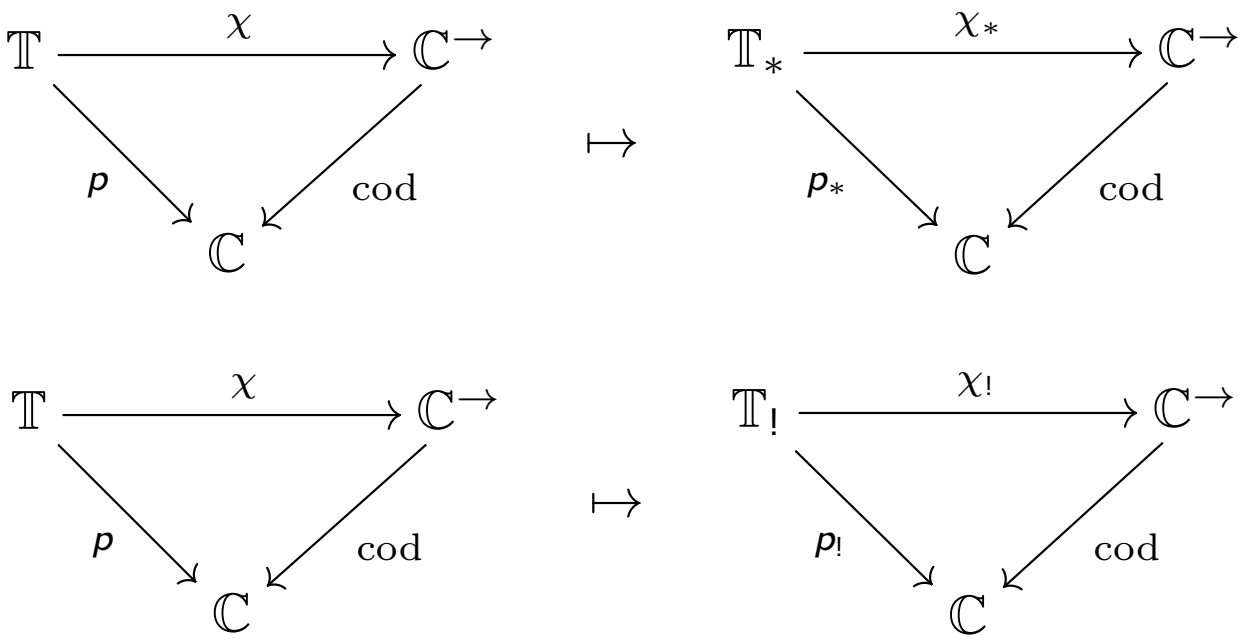
Issue (1) is reminiscent of the distinction between strict monoidal categories and monoidal categories.

Splitting comprehension categories

The inclusion

$$\text{Split Grothendieck fibrations}_{/\mathbb{C}} \hookrightarrow \text{Grothendieck fibrations}_{/\mathbb{C}}$$

has both a left and a right adjoint, so we get two splitting methods

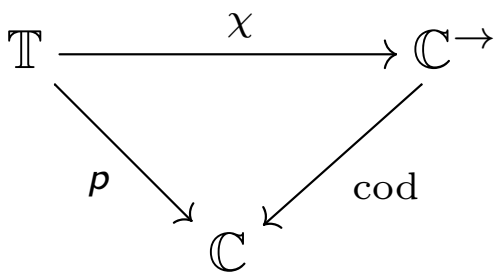


for comprehension categories.

Yet another splitting method is due to Voevodsky.

Strict stability

Fundamental question. What structure on a comprehension category



is sufficient in order for its left (or right, or Voevodsky) splitting to be a model of $\mathbf{ML_1}$?

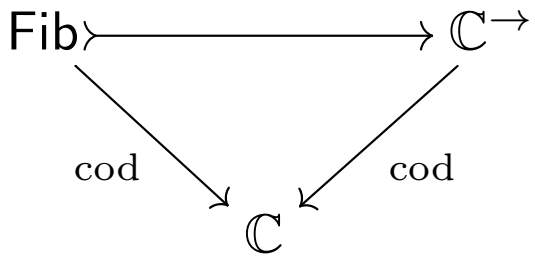
The case of Id-types

- For the left and Voevodsky splitting, a weak factorisation system of satisfying the Frobenius condition is sufficient.
- For the right splitting, one needs an algebraic weak factorisation system of satisfying the Frobenius condition.

Example: homotopical models

Consider a category $\mathbb{C}$ with a Quillen model structure $(\mathrm{Weq}, \mathrm{Fib}, \mathrm{Cof})$,

Then



is a comprehension category, but it is not split.

Under some assumptions, we can split it to obtain a model of $\mathbf{ML}_1$.

The simplicial model of Univalent Foundations arises in this way from the Kan-Quillen model structure on **SSet**.

References

- B. Jacobs
  Comprehension categories and the semantics of type dependency (1993)
- P.-L. Curien, R. Garner and M. Hofmann
  Revisiting the categorical interpretation of dependent type theory (2014)
- P. LeFanu Lumsdaine and M. A. Warren
  The local universe model: an overlooked coherence construction for dependent type theories (2015)
- V. Voevodsky
  A C-system defined by a universe category (2015)
- N. Gambino and M. F. Larrea
  Type-theoretic algebraic weak factorisation systems (2019)