Type theory and its homotopy-theoretical aspects

It is time to earn the right to be an LMFI course!

This section of the course has three lectures.

- Lecture A is about the syntax of type theory (Martin-Löf), with a particular emphasis on induction principles culminating with the identity type weak factorisation system
- Lecture B will be about the (pre-HoTT days) categorical interpretation of type theory, with a particular emphasis on coherence issues.
- Lecture C will be an introduction to the homotopical aspects of type theory: univalence, hLevels, higher inductive types, and time permitting, a glimpse at the weak higher groupoid structure of identity types.
- It is still a matter of research to have satisfactory material for a future lecture D about the categorical interpretation of **Homotopy** type theory (IMHO, and also by insufficient awareness of recent works!)

Lecture A Martin-Löf type theory

What is type theory?

The origin is the work on solving the paradoxes in the foundations of mathematics (beginning of the twentieth century):

$$\{a \mid a \notin a\}$$

TYPE information missing!

leads to contradictions! Indeed let b be this set, and suppose. $b \in b$. Then by definition of b, $b \notin b$: contradiction!

Russell invented type theory to put discipline in this jungle. One can only form sets of the form $\{a \in A \mid \text{some property holds}\}$. Here A is a type.

The modern forms of type theory start with Church (λ-calculus), until its present form known as Martin-Löf dependent type theory (1970's), for which we provide an introduction in this course.

part of the

Dependent type theory has received a new impetus in the late 2000's under the influence of Fields medallist Voevodsky, who found an interpretation of types as spaces. This has led to Homotopy type theory, which we shall also discuss in this course.

Some simple types known to the programmers

$\mathtt{nat}$ is a type, and $3 : \mathtt{nat}$ is an element of that type.
$\mathtt{nat} \times \mathtt{nat}$ is a type, and $(3, 4)$ is an element of this type.
$\mathtt{nat} \to \mathtt{nat}$ is a type and the function mapping x to $2x$ is an element of this type. We use the notations

$$\begin{array}{ll} x \mapsto 2x & \text{(standard in mathematical texts)} \\ \lambda x.(2 \times x) & (\lambda\text{-calculus}) \end{array}$$

A function can be applied to an argument: $(\lambda x.(2 \times x))\, 7 \equiv 14$
More slowly. we have

$$(\lambda x.(2 \times x))\, 7 \equiv 2 \times 7$$

and the rest is primary school mathematics!

λ-calculus

The λ-calculus forms the core of functional programming languages like OCaml and Haskell:

$$M ::\equiv x \mid \lambda x.M \mid MM$$

Definitional equality (oriented as reduction):

$$(\lambda x.M)N \to M[x \leftarrow N]$$

Warning: renaming of bound variables may be needed! Consider $M = \lambda x.\lambda y.x + y$. Then the intended conversion/value of $M\,y\,x$ is $y + x$. But blind application of the conversion gives

$$My \to \lambda y.y + y \qquad \text{and hence } (My)x \to (\lambda y.y + y)x \to x + x$$

The capture of the red occurrence of y should be avoided. The correct reduction is (α-conversion):

$$(My)x \to (\lambda z.y + z)x \to y + x$$

The wildness of untyped λ-calculus

In λ-calculus too, strange things happen! Consider

$$\Delta = \lambda x.xx$$

(self-application is permitted!). Then we have $\Delta\Delta \to \Delta\Delta$! Slowly:

$$\Delta\Delta \equiv (\lambda x.xx)\Delta \to \Delta\Delta$$

Hence computations may not terminate, or may not even produce some interesting infinite value....
To solve this problem, Church has introduced typed λ-calculi. For example, to type xx, we need to five a type $A \to B$ to x and a type A to x
In typed λ-calculi, all reductions **terminate**!

Δ can be typed in the polymorphic λ-calculus
(e.g. $\Delta : (\forall A.\, A \to A) \to Nat \to Nat$)

$\Delta\Delta$ cannot be "decently" typed!

PARADOXES, NON TERMINATION

même COMBAT!

Examples of dependent types

The type $\mathtt{Fin}(n)$ (defined p. 22) is the type of finite sets of cardinal n. For example,

$$\mathtt{Can}(n) :\equiv \{0_n, \ldots (n-1)_n\}$$

and we can define the map $\mathtt{max} := n \mapsto n_{n+1}$. We have

$$\mathtt{max} : \Pi_{n:\mathbb{N}} \mathtt{Can}(n+1)$$

We also have the type $\Sigma_{n:\mathbb{N}} \mathtt{Can}(n+1)$, whose elements are pairs of a number n and an element of $\mathtt{Fin}(n+1)$. For example:

$$(4, 2_5) : \Sigma_{n:\mathbb{N}} \mathtt{Can}(n+1)$$

Think of

$F : C \to \mathrm{Set}$ el F (x, a)

C Fx

"$\Sigma_{x:C} Fx$"

The identity type

The most important example of dependent type in Martin-Löf type theory is the identity type. If $x : A$ and $y : A$, then

$$\mathtt{Id}_A(x, y)\,, \quad \text{also written} \quad x =_A y\,, \text{ is a type}$$

When $x = y$, there is a term (constructor) of this type:

i.e. y is x

$$\mathtt{refl}_x : x =_A x$$

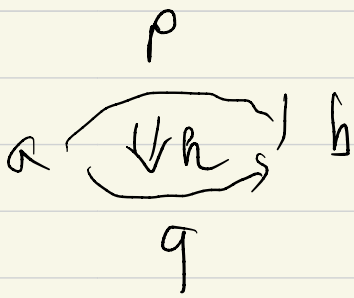
When $x =_A y$ is inhabited (i.e., there exists $p : x =_A y$), we say that x and y are propositionally equal.

The identity type is best interpreted in the homotopy interpretation of type theory:

- Types are spaces
- $a : A$ is a point in space A
- $p : a =_A b$ is a path from a to b
- $h : p =_{a=_A b} q$ is a "path", or homotopy between the paths p and q

More on identity types

- at the end of this lecture
- in lecture C



The judgments of type theory

The main judgment is $a : A$.

A type A is itself an element of a very large type, called $U = U_0$ for the universe. But U is also an element of an even larger type U_1, etc., so there is a hierarchy of universes. (Setting $U : U$ would lead back to paradoxes!)

Actually, judgments are in context, like for example $x : \mathbb{N} \vdash (2 \times x) : \mathbb{N}$.

A context is a list of type declarations. In the setting of dependent types, *the order of these declarations matters*, as types may depend on the previously declared variables, for example, $x : \mathbb{N}, y : \mathtt{Fin}(x)\ \mathtt{ctx}$ (meaning that it is a valid context), while $y : \mathtt{Fin}(x), x : \mathbb{N}$ does not make sense.

Contexts are extended via the following rule:

$$\frac{\Gamma \vdash A : U}{\Gamma, x : A\ \mathtt{ctx}}$$

Finally, definitional (or judgmental) equality is also typed: $\Gamma \vdash a \equiv b : A$

In summary, three judgments: $\Gamma\ \mathtt{ctx}$ $\quad$ $\Gamma \vdash a : A$ $\quad$ $\Gamma \vdash a \equiv b : A$

We often omit Γ (or rather its "dummy" part that is not relevant for the discussion), and we write $a : A$ for $\vdash a : A$.

One can also add another judgment $\Gamma \vdash \rho : \Delta$

↳ substitution = list of terms

empty list ↘ ← empty context

$\Gamma \vdash [] : \emptyset$

$$\frac{\Gamma \vdash \rho : \Delta \qquad \Delta \vdash A \qquad \Gamma \vdash a : A[\rho]}{\Gamma \vdash (x \leftarrow a) \cdot \rho : \Delta, x : A}$$

$(x \leftarrow a) \cdot \rho$: (ρ, a)

Some typing rules and a remark

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash A \equiv B : U}{\Gamma \vdash a : B}$$

$$\frac{(i < j)}{U_i : U_j} \qquad \frac{A : U_i}{A : U_{i+1}}$$

A non-example: there is no type family $\lambda n.U_n$, because

- there is no universe large enough to be its codomain,
- we do not identify the indices n with the natural numbers of type theory.

In presence of dependent types, definitional equality contaminates types!

$Fin(4) \equiv Fin((\lambda x.\, x)\, 4)$

Families of types

$Can(n)$ is a family of types, over $\mathbb{N}$. More precisely, we have

$$n : \mathbb{N} \vdash Can(n) : U$$

We can also write

$$Can : \mathbb{N} \to U$$

Successive dependencies: if $A : U$ and $B : A \to U$, the type of a family C depending on A and B is

$$C : \Pi_{x:A}(B(x) \to U) \qquad (a\ (\Sigma_{x:A} Bx) \to U)$$

We have:

$$x : A,\ y : Bx \vdash Cxy : U \quad \text{and hence} \quad x : A,\ y : Bx,\ z : Cxy\ \mathtt{ctx}$$

also written $B(x)$ $C(x,y)$

→ is a special case of Π type (non-dependent, p.15)

$\Pi_{A:U} \cdots$ = polymorphic type

Application vs substitution : $Can(n)$ is in fact an informal notation: what we define p.22 is in fact $n : \mathbb{N} \vdash \underline{Can} : U$, and hence $Can = \lambda n.\underline{Can}$ and, say, $Can(3)$ is in fact $\underline{Can}[n \in 3]$

For this reason, we often write $\underline{Can}(n)$ for $\underline{Can}[n \in 3]$.

Curry-Howard

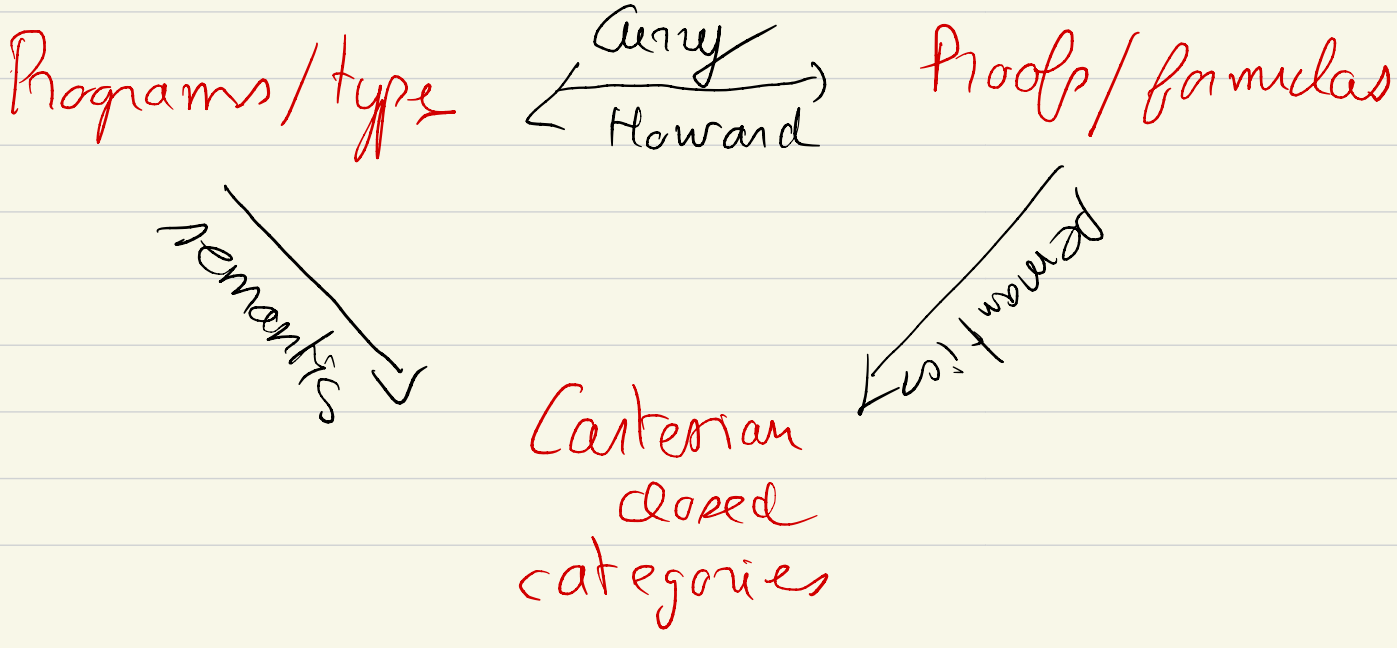
We can also interpret types as propositions, and terms as proofs (but the homotopy type theory perspective says that not all types are propositions).

- One can read $A \times B$ as $A \wedge B$, since in order to prove $A \wedge B$ we need a pair (p, q) where p is a proof of A and q is a proof of B.
- One can read $A \to B$ as $A \Rightarrow B$, since proving $A \Rightarrow B$ amounts to proving B under assumption A: compare with

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

- One can read $A + B$ as $A \vee B$, since in order to prove $A \vee B$ we need a proof of A or a proof of B.
- One can read the identity type as the equality predicate.
- One can read Π and Σ as universal and existential quantification, respectively.

eg. $\neg\neg A \to A$



The swap function

Let

- $A : U$, $B : U$, $C : A \to B \to U$,
- $h : \Pi_{x:A}\Pi_{y:B}.Cxy$

We want

$$\mathtt{swap}\, A\, B\, C\, h$$

to swap the arguments of h (which is possible, because neither B depends on A nor A depends on B).

We set

$$\mathtt{swap} :\equiv \lambda A \lambda B \lambda C \lambda h \lambda y \lambda x.h\, x\, y :$$
$$\Pi_{A:U}\Pi_{B:U}\Pi_{C:A\to B\to U}((\Pi_{x:A}\Pi_{y:B}\, C\, x\, y) \to (\Pi_{y:B}\Pi_{x:A}\, C\, x\, y))$$

This is an example of a parametric polymorphic function.

Exercise 1: Let $A : U$ and $B : U$. Construct a term (a de Morgan law!) of type

$$((A + B) \to 0) \to (A \to 0) \times (B \to 0)$$

Exercise 2: Let $A : U$, $P : A \to U$ and $Q : A \to U$. Construct a term of type

$$((\Pi_{x:A}(Px \times Qx)) \to ((\Pi_{x:A}Px) \times (\Pi_{x:A}Qx))$$

The general pattern for types

- Formation rules for the new type
- Introduction rules for the new type, featuring the new constructors:

CONSTRUCTION

$$\lambda x.a \quad (a, b) \quad \mathtt{inl}(a) \quad \mathtt{inr}(a) \quad 0_2 \quad 1_2 \quad \mathtt{refl}x$$

- Rules specifying how the type is used: application, induction operators (when the induction operator is applied to a constant type family, we call it recursion operator).
- Associated definitional equalities
- Sometimes a uniqueness principle, like $\lambda x.ax \equiv a$. Such principles are often valid propositionally (see exercise p. 29)

as imposed only

CONSUMPTION

Usually coming with an orientation

≠

Computation!

Two kinds of equality

We have seen

- The definitional equality, for which the notation is $\equiv$. In these lectures, we also use the symbol $:\equiv$, which stresses more the definitional side, especially when defining macros.
- The propositional equality $a =_A b$.

Let us stress the difference:

- The definitional equality is set up at the level of judgments:

$$\boxed{\Gamma \vdash a \equiv b : A}$$

  This is why it is also called judgmental.
- The propositional equality is set up at the level of types:

$$\boxed{x : A, y : A \vdash x =_A y : U}$$

Definitional equality is stronger, since if $a \equiv b$, then $(a =_A b) \equiv (a =_A a)$, and hence $\mathtt{refl}_A a : a =_A b$.

The Π-type

This is the dependent version of the function type.

- If $A : U$ and $B : A \to U$, then $\Pi_{x:A} B\,x : U$.
- If $x : A \vdash b : B\,x$, then $\lambda x.b : \Pi_{x:A} B\,x$.
- if $a : A$ and $b : \Pi_{x:A} B x$, then $b\,a : B\,a$
- We require the definitional equality $(\lambda x.b)a \equiv b[x \leftarrow a]$ (and optionally $\lambda x.b\,x \equiv b$).

β-rule

η-rule

uniqueness

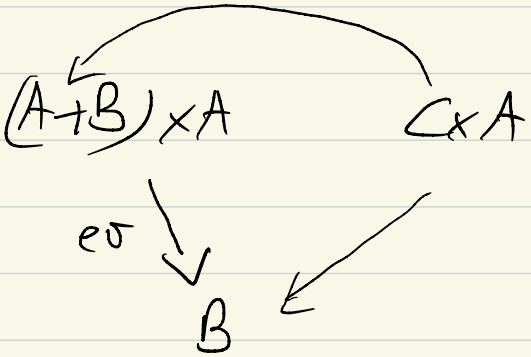
commutation

naturality

If B does not depend on A, then $\Pi_{x:A} B$ is $A \to B$, also written B^A (think of B^A as $\underline{B \times B \times \ldots \times B \text{ for cardinal of } A \text{ copies of } B}$).

cf. powers of

lecture 1 p. 5



An example from high school:

$$m : \mathbb{R} \vdash \lambda x.(x^2 + 3mx - (m-4)) : \mathbb{R} \to \mathbb{R}$$

parameter

variable

The Σ-type

This is the dependent version of the product type.

- If $A : U$ and $B : A \to U$, then $\Sigma_{x:A} B\,x : U$.
- If $a : A$ and $b : B\,a$, then $(a, b) : \Sigma_{x:A} B\,x$.

We bootstrap the induction operator:

$$\begin{aligned}
\mathtt{rec}_{A\times B} &: \Pi_{C:U}(A \to B \to C) \to ((A \times B) \to C) \\
\mathtt{ind}_{A\times B} &: \Pi_{C:(A\times B)\to U}(\Pi_{x:A,y:B} C\,(x, y)) \to \Pi_{z:A\times B} C\,z \\
\mathtt{ind}_{\Sigma_{x:A}B\,x} &: \Pi_{C:(\Sigma_{x:A}B\,x)\to U}(\Pi_{x:A,y:B\,x} C\,(x, y)) \to \Pi_{z:\Sigma_{x:A}B\,x} C\,z
\end{aligned}$$

(intuition: every inhabitant of $\Sigma_{x:A} B\,x$ is a pair)

- We require the definitional equality $\mathtt{ind}_{\Sigma_{x:A}B}\ C\ f\ (a, b) :\equiv f\ a\ b$.

If B does not depend on A, then $\Sigma_{x:A} B$ is $A \times B$

(think of $A \times B$ as $B + B + \ldots + B$ for cardinal of A copies of B)

Cf. coproducts of Lecture 1 p. 5

Observe that this goes in the opposite direction with respect to Π-introduction

From induction operator to projections

Let $\mathtt{ind}_{\Sigma_{x:A}Bx}C : (\Pi_{x:A}\Pi_{y:Bx}C(x,y)) \to \Pi_{z:\Sigma_{x:A}Bx}Cz$. We define

$$\begin{aligned}\mathtt{pr}_1 &:= \mathtt{rec}_{\Sigma_{x:A}Bx}\,A\,(\lambda x\lambda y.x)\\ \mathtt{pr}_2 &:\equiv \mathtt{ind}_{\Sigma_{x:A}Bx}\,(\lambda a.B\,(\mathtt{pr}_1 a))\,(\lambda x\lambda y.y)\end{aligned}$$

eliminators

Slowly, setting $C :\equiv \lambda a.B\,(\mathtt{pr}_1 a)$, $\mathtt{ind}_{\Sigma_{x:A}Bx}\,(\lambda a.B\,(\mathtt{pr}_1 a))$ expects an argument of type $\Pi_{x:A}\Pi_{y:Bx}C(x,y) \equiv \Pi_{x:A}\Pi_{y:Bx}Bx$, so that the definition of $\mathtt{pr}_2$ type-checks. The two definitional equations for $\mathtt{pr}_1$ and $\mathtt{pr}_2$ are easily derived.

non-dependent case

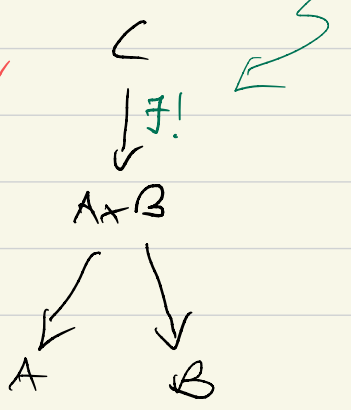
$pr_1 : A \times B \to A$

$pr_2 : A \times B \to B$

$$\begin{aligned}\mathtt{pr}_1 &: (\Sigma_{x:A}Bx) \to A\\ \mathtt{pr}_2 &: \Pi_{a:\Sigma_{x:A}Bx}B(\mathtt{pr}_1 a)\end{aligned}$$

$$\mathtt{pr}_1(x,y) \equiv x \quad \mathtt{pr}_2(x,y) \equiv y$$

Exercise show that the type $C :\equiv \lambda z.(\mathtt{pr}_1 z, \mathtt{pr}_2 z) =_{\Sigma_{x:A}Bx} z$ is inhabited (i.e. exhibit a term of this type) thus witnessing uniqueness propositionally



The coproduct type

- If $A : U$ and $B : U$, then $A + B : U$.
- If $a : A$ (resp. $b : B$), then $\mathtt{inl}\, a : A + B$ (resp. $\mathtt{inr}\, b : A + B$).

Induction operator:

$$\mathtt{ind}_{A+B} : \Pi_{C:(A+B)\to U}(\Pi_{a:A} C\,(\mathtt{inl}\, a)) \to (\Pi_{b:B} C\,(\mathtt{inr}\, b)) \to \Pi_{z:A+B} C\, z$$

(intuition: every inhabitant of $A + B$ comes from A or B)

- We require the definitional equalities

$$\mathtt{ind}_{A+B}\, C\, f\, g\,(\mathtt{inl}\, a) \equiv f\, a \qquad \mathtt{ind}_{A+B}\, C\, f\, g\,(\mathtt{inr}\, b) \equiv g\, b$$

The types 1 and 0

They are the 0-ary versions of the product and coproduct constructors, respectively.

- $1 : u$, $\star : 1$, and the induction operator is

$$\mathtt{ind}_1 :\equiv \Pi_{C:1\to U}(C\,\star) \to \Pi_{z:1}(C\,z)$$

- $0 : u$, there is no constructor, and the induction operator is

$$\mathtt{ind}_0 :\equiv \Pi_{D:0\to U}\Pi_{x:0}(D\,x)$$

The type of booleans

It is the special case $1 + 1$. It can be introduced directly:

- $2 : U$
- The constructors are $0_2 : 2$ and $1_2 : 2$
- Induction operator:

$$\mathtt{ind}_2 :\equiv \Pi_{C:2\to U}(C\,0_1) \to (C\,1_2) \to \Pi_{z:2} C\,z$$

(intuition: every boolean is either 0_2 or 1_2)

- We require the definitional equalities

$$\mathtt{ind}_2\,C\,c_0\,c_1\,0_2 \equiv c_0 \qquad \mathtt{ind}_2\,C\,c_0\,c_1\,1_2 \equiv c_1$$

Exercise Show that $A + B = \Sigma_{x:2}\mathtt{rec}_2\,U\,A\,B\,x$ provides an encoding of sums from the type of booleans. Define $\mathtt{rec}_{A+B}$ in terms of $\mathtt{rec}_2$ and $\mathtt{rec}_{\Sigma_{x:2}\mathtt{rec}_2}$.

The type of natural numbers

- $\mathbb{N} : U$
- The constructors are given by $0 : \mathbb{N}$ and $\mathtt{succ} : \mathbb{N} \to \mathbb{N}$
- Induction operator:

$$\mathtt{ind}_{\mathbb{N}} : \Pi_{C:\mathbb{N}\to U}(C\,0) \to (\Pi_{n:\mathbb{N}}(\quad (C\,n) \to (C\,(\mathtt{succ}\,n))) \to \Pi_{n:\mathbb{N}} C\,z$$

We require the definitional equalities

$$\mathtt{ind}_{\mathbb{N}}\,C\,c_0\,c_s\,0 \equiv c_0 \qquad \mathtt{ind}_{\mathbb{N}}\,C\,c_0\,c_s\,(\mathtt{succ}\,n) \equiv c_s\,n\,(\mathtt{ind}_{\mathbb{N}}\,C\,c_0\,c_s\,n)$$

This is a prototypical example of inductive type

Some constructors take as argument an element of the type being defined.

The dependent type of finite sets

We first define $\mathtt{Can} : \mathbb{N} \to U$ by induction (notice the typographical difference: on the right, 0 and 1 are types!) :

$$\mathtt{Can}\,0 :\equiv 0 \qquad \mathtt{Can}\,(\mathtt{succ}\,n) :\equiv (\mathtt{Can}\,n) + 1$$

We then define (without induction)

$$\mathtt{Fin}\,n :\equiv \Sigma_{A:U}(A =_U \mathtt{Can}\,n)$$

$$\mathtt{max}\,0 :\equiv (\mathtt{inr}\star) \qquad \mathtt{max}\,(\mathtt{succ}\,n) :\equiv (\mathtt{inl}\,(\mathtt{max}\,n))$$

where $\star$ is the (unique) constructor of type 1.

We introduce the notation $i_{(\mathtt{succ}\,n)} = \mathtt{pick}\,i\,n\,p$ where p is a proof of $i \leq n$, and where $\mathtt{pick} : \Pi_{n,i:\mathbb{N}}\,(i \leq n) \to \mathtt{Can}\,(\mathtt{succ} n)$ is defined by

$$\mathtt{pick}\,0\,n\,p = (\mathtt{inr}\star) \qquad \mathtt{pick}\,(\mathtt{succ}\,i)\,(\mathtt{succ}\,n)\,p = \mathtt{inl}(\mathtt{pick}\,i\,n\,p)$$

We define $\leq : \Pi_{x,y:\mathbb{N}} U$ by induction on x, and then on y:

$$(0 \leq y) :\equiv 1$$
$$(\mathtt{succ}\,x \leq 0) :\equiv 0$$
$$(\mathtt{succ}\,x \leq \mathtt{succ}\,y) :\equiv (x \leq y)$$

The identity type

- If $A : U$, then $x : A, y : A \vdash (x =_A y) : U$.
- Constructor: if $a : A$, then $(\mathtt{refl}\, a) : (a =_A a)$.
- Induction operator:

also written $\mathrm{Id}_A(a,a)$

(path induction)

$$\mathtt{ind}_{=_A} : \prod_{C:\prod_{x,y:A}(x=_A y)\to U} \prod_{x:A} C\, x\, x\, \mathtt{refl}_x \to \prod_{x,y:A}\prod_{p:x=_A y} C\, x\, y\, p$$

(intuition: the only *generic* element of the identity type is $\mathtt{refl}$)

We require the definitional equality $\mathtt{ind}_{=_A}\, C\, c\, x\, x\, (\mathtt{refl}_x) \equiv c\, x$

Exercise Show the interdefinability of path induction above and based path induction typed as follows:

$$\mathtt{ind}'_{=_A} : \prod_{x:A}\prod_{D:\prod_{y:A}(x=_A y)\to U} D\, x\, (\mathtt{refl}_A\, x) \to \prod_{y:A}\prod_{p:x=_A y} D\, y\, p$$

Hint. To define $\mathtt{ind}'_{=_A}$ from $\mathtt{ind}_{=_A}$, apply $\mathtt{ind}_{=_A}$ to

$$C :\equiv \lambda x \lambda y \lambda p.\prod_{E:\prod_{z:A}(x=_A z)\to U} E\, x\, (\mathtt{refl}_A\, x) \to E\, y\, p$$

(idea of Christine Paulin)

The category of contexts

- Objects are contexts Γ such that $\Gamma \ \mathrm{ctx}$ is provable (cf. p. 9)
  The general form of a context is (cf. p. 11)

$$\Delta = x_0 : B_0,\ x_1 : B_1(x_0),\ x_2 : B_2(x_0, x_1),\ \dots\ x_n : B_n(x_0, \dots, x_{n-1})$$

(B_1 stands for B_1)

CONTEXTS taken UP TO RENAMING

- A morphism from Γ to Δ is a sequence

$(b_0, \dots b_n)$ of terms s.t.

$$\Gamma \vdash b_0 : B_0 \quad \Gamma \vdash b_1 : B_1(b_0) \quad \Gamma \vdash b_2 : B_2(b_0, b_1) \quad \dots \quad \Gamma \vdash b_n : B_n(b_0, \dots, b_{n-1})$$

($B_1(b_0)$ stands for $B_1[x \leftarrow b_0]$ (cf. p. 4 for the notion of substitution))

The b_i's are taken UP TO DEFINITIONAL EQUALITY

If $\Delta = \Gamma_0, \Delta_1$ (where $\Gamma = \Gamma_0, \Gamma_1$), where, say, we have cut Δ above (between i and $i+1$, and Γ and Δ_1), and if $b_0 = x_0, \dots b_i = x_i$, then we abbreviate

$$(x_0, \dots x_i, b_{i+1}, \dots b_n) \text{ as } (id_{\Gamma_0}, b_{i+1}, \dots, b_n)$$

$: \Gamma \to \Gamma, \Delta_1$

note that this abbreviation allows to give several types to id !

- If $\Gamma_1 = \Delta_1 = \emptyset$ then we have $(x_0, \dots x_n) = id_\Gamma : \Gamma \to \Gamma$, which is the identity morphism

- Composition of morphisms is by substitution, e.g.

$\Gamma \vdash a : A$

$$\Gamma \xrightarrow{(id, a)} \Gamma, x : A \xrightarrow{b} y : B$$

$$b[x \leftarrow a]$$

(more details next page)

- Important morphisms are the display maps $id_\Delta : \Delta, x : A \to \Delta$ or more generally the projection maps $id_\Delta : \Delta, \Gamma \to \Delta$.

Exercise Show that projections are composites of display maps.

Composition in the category of contexts, slowly

- In reference to p. 24, we can write

$$\rho = (b_0, \dots b_n) = [\vec{x} \vdash \vec{b}]$$

- Let $\Gamma \xrightarrow[\rho]{[\vec{x} \vdash \vec{c}]} \Delta \xrightarrow[t]{[\vec{x}' \vdash \vec{c}']} \Delta'$. Then we define

$$t \circ \rho = [\vec{x}' \vdash \overrightarrow{c'[\vec{x} \vdash \vec{c}]}]$$

This type-checks: Let $c' = c'_k$, $\Delta' \equiv \Delta'', x_k : A, \dots$
Let $t' : \Delta \to \Delta''$ be the restriction of t to Δ''

Then $c' : A[t']$, and thus $c'[\rho] : A[t'][\rho]$

On the other hand, for $[\vec{x}' \vdash \overrightarrow{c'[\rho]}]$ to be well-formed, we need $c'[\rho] : A[t' \circ \rho]$

It is true, because in syntax we indeed have

$$\boxed{A[t'][\rho] \equiv A[t' \circ \rho]}$$

This is called substitution lemma

(More on this in Lecture 8)

$\mathrm{ind}_{=_A}$ as a lifting

after Gambino-Garner

Let us unravel

https://arxiv.org/abs/0803.4349

$$\mathrm{ind}_{=_A} :\equiv \Pi_{C:\Pi_{x,y:A}(x =_A y)\to U}\Pi_{x:A} C\,x\,x\,\mathrm{refl}_x \to \Pi_{x,y:A}\Pi_{p:x=_A y} C\,x\,y\,p$$

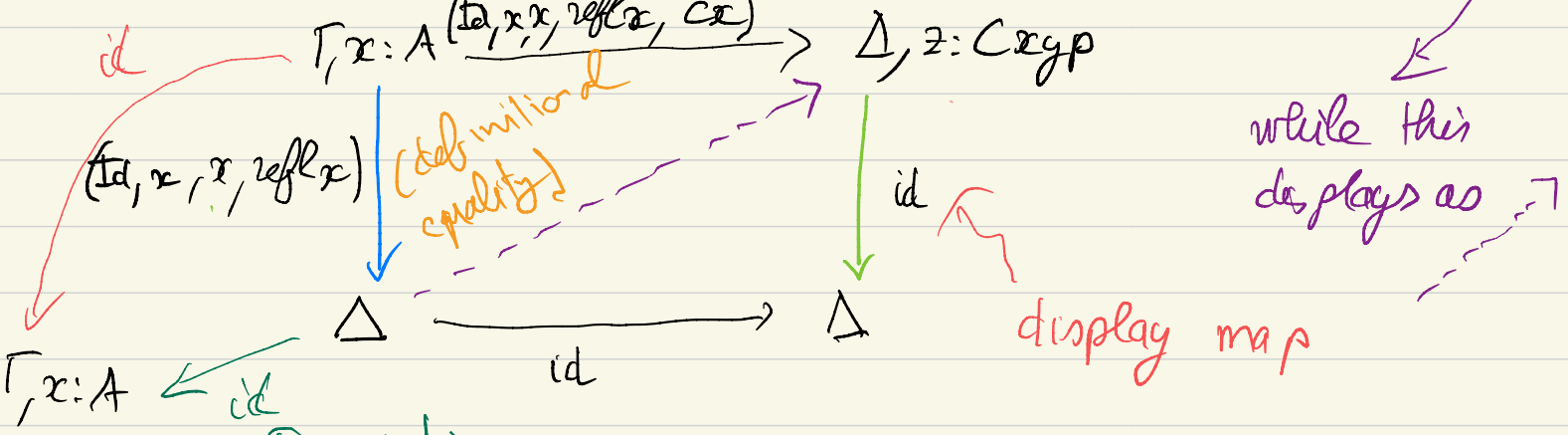
We have Δ

$$\begin{cases} \Gamma, x:A, y:A, p: x =_A y \vdash C\,x\,y\,p : U \\ \Gamma, x:A \vdash c\,x : C\,x\,x\,(\mathrm{refl}\,x) \end{cases}$$

and

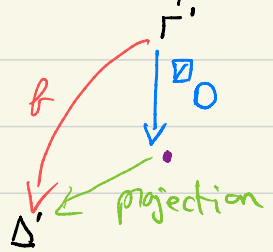
$$\Gamma, x:A, y:A, p: x =_A y \vdash \mathrm{ind}\,C\,c\,x\,y\,p : C\,x\,y\,p$$

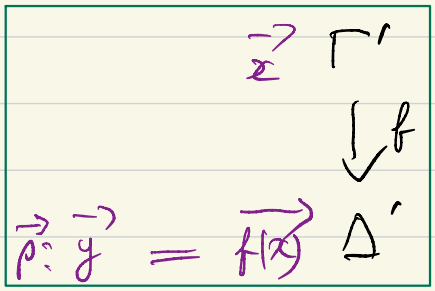
We can display this information in the category of contexts



We set $\underset{\Delta}{\mathbb{D}} = \{\text{display maps}\}$ in the category of contexts

It turns out that, replacing id by an arbitrary f, $\xi : \Gamma' \to \Delta'$ we can always factorise f as





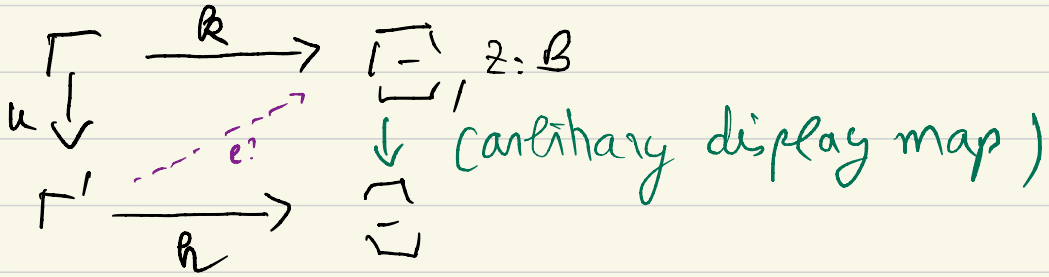
Exercise Prove this. Hint: (with hopefully self-explanatory notations) take $\mathrm{Id}(\xi)$ as $\bullet$, where $\mathrm{Id}(\xi) :\equiv \vec{x}:\Gamma', \vec{y}:\Delta', \vec{p}: f\vec{x} =_{\Delta'} \vec{y}$.

other hints in the exercises next page.

Some more details

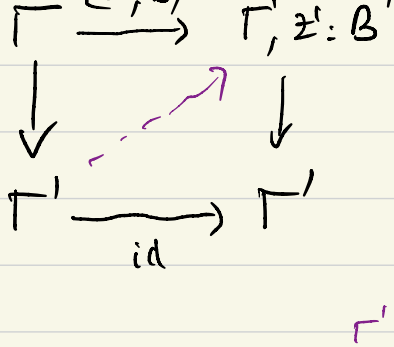
Exercise Suppose that we have $\Gamma \vdash A : U$ $\Gamma \vdash B : U$ $\Gamma \vdash f : A \to B$, $\Gamma, x:A, y:B, p : fx =_B y \vdash C\,x\,y\,p : U$ and $\Gamma, x:A \vdash c\,x : C\,x\,(fx)\,(\mathrm{refl}\,(fx))$.
Show that one can construct $\Gamma, x:A, y:B, p : fx =_B y \vdash$? $: C\,x\,y\,p$
(Hint: use based path induction!)

As further help towards showing that $\Gamma \downarrow \mathrm{Id}(\ell) \in {}^{\mathbb{D}}0$,
suppose that we have a lifting problem of the form



Then, writing $k = k', b$ and $e = e', b'$, the commutation of $\ulcorner\!\urcorner \downarrow$ and of $\nearrow \downarrow$ says that $k' = h \circ u$ and $e' = h$, and the commutation of $\downarrow \nearrow$ says that $b = b'[a]$.
So all we need is to find b' s.t. $b = b'[a]$. Moreover (cf p. 8 on 24)

- we have $b : B[k'] = B[h][u]$, (B')
- we seek $b' : B[h]$, (B')
- and our lifting problem is reformulated as



Indeed, this $\nearrow$ will be of the form (id, b') ($\nearrow \downarrow$) and we will have $b = b'[u]$ ($\downarrow \nearrow$).

These remarks bring us to a problem "of the shape" treated p. 26, which we solve for $\Gamma \downarrow \mathrm{Id}(\ell)$ along the lines of Exercise above.

The identity type weak factorisation system

Proposition The pair $({}^{\square}\mathcal{D}, ({}^{\square}\mathcal{D})^{\square})$ (display maps) forms a weak factorisation system.

Proof By the exercise p.24 and the lemma of lecture 7 p.2, we deduce a fortiori the $\to$ $\to$ factorisation from exercise p.26. Setting $I = {}^{\square}\mathcal{D}$ and $P = ({}^{\square}\mathcal{D})^{\square}$, we have that $L^{\square} = R$ is tautological, and that $L = R^{\square}$ follows by exercise lecture 7 p.2.

Exercise 1 Let (L, R) be a weak factorisation system on a category $\mathbb{C}$. Suppose that we have chosen $f \in \mathrm{Mor}\,\mathbb{C}$ and a factorisation $f = \overset{fe}{\underset{L}{\to}} \overset{fr}{\underset{R}{\to}}$.
Then show that $f \in L$ iff $f \downarrow \overset{fe}{=} \downarrow fr$ admits a lifting. Characterise $f \in R$ symmetrically.

Exercise 2 Show that (for $\mathcal{D}$ as above), we have that

- ${}^{\square}\mathcal{D}$ is the set of type-theoretic injective equivalences, i.e. the set of maphisms $f : \Gamma \to \Delta$ for which
  - there exists $\rho : \Delta \to \Gamma$ s.t. $\rho \circ f \equiv \mathrm{id}$ (in the category of contexts)
  - there exists $\vec{\varepsilon}$ such that $[f(\rho(\vec{y})), \vec{y}, \vec{\varepsilon})$ is a maphism from Δ to $\tilde{\Delta}$, where $\tilde{\Delta}$ is obtained by triplicating each $x : A$ in Δ into $x_1 : A, x_2 : A, x_3 : x_1 =_A x_2$.
    This expresses a homotopy from $f\rho$ to id.
    and s.t. $\vec{\varepsilon}[\vec{y} \leftarrow f(\vec{x})] \equiv \overrightarrow{\mathrm{refl}_{f(x)}}$

  = unraveling of $\Gamma \xrightarrow{\vec{x}, \vec{x}, \overrightarrow{\mathrm{refl}}} \widetilde{\mathrm{Id}(f)}$, $f \downarrow$, $(\rho, \mathrm{id}, \vec{\varepsilon} : \overrightarrow{f(\rho\vec{y}) = \vec{y}})$, $\Delta = \Delta$

- $({}^{\square}\mathcal{D})^{\square}$ is the set of type-theoretic normal isofibrations, i.e., the set of maphisms $f : \Gamma \to \Delta$ for which (taking the notation of exercise p.26 and exercise 1 above)

there exists $j : \mathrm{Id}(f) \to \Gamma$ such that

1. $j \circ fe \equiv \mathrm{id}$
2. $f \circ j \equiv fr$

$\Gamma = \Gamma$, $fe \downarrow$, j, $\downarrow f$, $\mathrm{Id}(f) \xrightarrow{fr} \Delta$

$f(\vec{x}, \vec{y}, \vec{p})$ $\vec{\Sigma}$ Γ, $\downarrow f$, $\vec{p} : \overrightarrow{\vec{y} = f(\vec{x})}$ Δ

Σ normal

1. $j(\vec{x}, f(\vec{x}), \overrightarrow{\mathrm{refl}}) \equiv \vec{x}$
2. $f(j(\vec{x}, \vec{y}, p)) \equiv \vec{y}$

isofibration

Summary of the steps taken

- We have defined $\mathbb{D} = \{ \Gamma, x:A \xrightarrow{id} \Gamma \}$ (the set of all display maps).
- (p.26) We have recognized that $(id_\Gamma, x, \mathrm{refl}\, x)$ is orthogonal to same display map.
- (p.27) We have suggested why, similarly, for all $f : \Gamma' \to \Delta'$, $(\vec{x}, f(\vec{x}), \mathrm{refl}_{f\vec{x}}) : \Gamma' \to \mathrm{Id}(f)$ is orthogonal to all display maps, i.e. $fe \in {}^{\boxtimes}\mathbb{D} = I$

For this, based path induction was needed.

- while $f_2 : \mathrm{Id}(f) \to \Delta$ is a composition of display maps, and hence $f_2 \in ({}^{\boxtimes}\mathbb{D})^{\boxtimes} = P$
- Then it follows obviously that (I, P) is a weak factorisation system
- then we used exercise[1] p.28 to characterise I and P.

Sanity check: We check that $(\mathrm{Id}, x, x, \mathrm{refl}\, x)$ is a type-theoretic injective equivalence. The "inverse" is the projection $\Delta \to \Gamma, x:A$ (in the notation of p.26). We have to find (omitting id_Γ)

$$\varepsilon : (x, y, p : x = y) \equiv (x, x, \mathrm{refl}\, x)$$

Here we use transport (cf. characterization of $=_{\Sigma x:A.B}$ Cours TTC, p.4)

We have $p^{-1} : y = x$ and $\mathrm{transport}^{\lambda y.\, x=y}\, p^{-1}\, p = \mathrm{refl}\, x$

(more generally $\mathrm{transport}^{\lambda y.\, x=y}\, (q : y = y')(p : x = y) = p \cdot q$)

On the magic of path induction

The picture $f(\vec{x},\vec{y},\vec{p})$ $\vec{x}$ Γ $\downarrow f$ $\vec{p}:\vec{y}=\overrightarrow{f(x)}$ Δ looks incomplete.

We would rather expect (anticipating the notion of Benabou-Grothendieck fibration, see lecture B)

$?:f(\vec{x},\vec{y},\vec{p})=\vec{x}$ Γ $\downarrow f$ $\vec{p}:\vec{y}=\overrightarrow{f(x)}$ Δ such that "$f(?)=\vec{p}$"

Indeed, $?$ can be derived by path induction!

Exercise Prove this! Hint: consider C defined as $C\,\vec{x}\,\vec{y}\,\vec{p}=\Sigma_{\vec{q}:\,f(\vec{x},\vec{y},\vec{p})=\vec{x}}\ ap\,f\,\vec{q}=\vec{p}$ and use $\bullet^1$ and $\bullet^2$ to make sure that the "obvious candidate" for $?$ type-checks correctly.

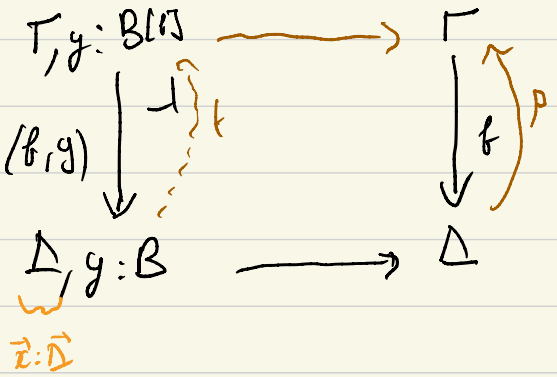
C, and

defined in lecture C!

A bonus property of the $(\mathbb{I}, P)$ weak factorisation system

- We have that $\mathbb{I}$ is stable by pullbacks along maps of P (a property we'll need in Cours TT B)

Proof (beginning) Suppose that f is a type-theoretic weak equivalence.



Our goal is to build g from $\Delta \to \Gamma$

We define $t = (\rho, ?)$, where

$?$ must have as type

$$B[f][\rho] = B[f \circ \rho] : B[f(\rho \vec{x})]$$

But we have $y : B$ and $\varepsilon : \overrightarrow{f(\rho \vec{x})} = \vec{x}$ over Δ

We thus take $? = \text{transport}^{B}\ \varepsilon^{-1}\ y$