Introduction to Homotopy Type Theory

Nicolai Kraus

Midlands Graduate School 2021, 12–16 April

(combined slides for all lectures)

Ways to introduce (homotopy) type theory

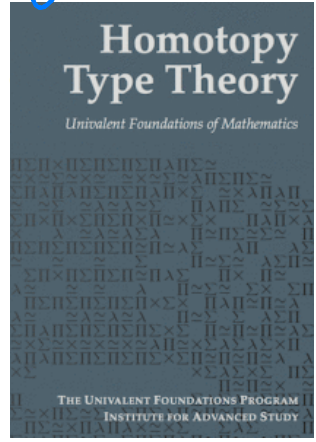
Agda

```
x·-mono : ∀ {x y z : Brw} → y ≤ z → x · y ≤ x · z
x·-mono ≤-zero = ≤-zero
x·-mono (≤-trans y≤w w≤z) = ≤-trans (x·-mono y≤w) (x·-mono w≤z)
x·-mono {x} (≤-succ-mono {y} {z} y≤z) = +x-mono x (x·-mono {x} y≤z)
x·-mono {x} {y} (≤-cocone f {k = k} y≤z) with decZero x
... | yes x≡zero = subst (λ z → z ≤ zero) (sym (zero·y≡zero y) • cong
... | no x≢zero = ≤-cocone (λ n → x · f n) {k = k} (x·-mono y≤z)
x·-mono {x} (≤-limiting f f≤z) with decZero x
... | yes x≡zero = ≤-zero
... | no x≢zero = ≤-limiting (λ n → x · f n) λ k → x·-mono (f≤z k)
x·-mono (≤-trunc p q i) = ≤-trunc (x·-mono p) (x·-mono q) i
```

inference rules

$$\frac{\Gamma, x{:}\mathbf{1} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c : C[\star/x] \qquad \Gamma \vdash a : \mathbf{1}}{\Gamma \vdash \mathsf{ind}_{\mathbf{1}}(x.C, c, a) : C[a/x]}\ \mathbf{1}\text{-ELIM}$$

natural math language



Prerequisites

- *Basic examples of types* Nat Bool Unit Empty
- *Universe(s)* Type $A, B : \mathrm{Type}$
- *Function types* $A \to B$
- *Dependent function types* if $C : A \to \mathrm{Type}$, then $(a : A) \to C\,a$
- *Binary products (pairs) and coproducts (sums)* $A \times B$ $A + B$
- *Dependent pairs* $\Sigma (a : A).\, C\,a$ $[a\,b\,a\ \Pi (a : A).\, C\,a]$
- *Inductive types* see above

Program = Proof

Exercise: Formulate a type which says that there are infinitely many primes (or simply: arbitrarily large primes)!

isPrime : Nat → Type

isPrime(n) :≡ $((m\ k : \mathbb{N}) \to (m \cdot k = n) \to (m = 1) + (m = n)) \times (n > 1)$

bigPrimes : $(n : \mathrm{Nat}) \to \Sigma\,(p : \mathrm{Nat}).\ \mathrm{isPrime}(p) \times p > n$

bigPrimes :≡ (exercise)

Caveat: judgments (meta-theoretic)

- A type "A is a valid type"
- $a : A$ "a is term of type A"
- $a \equiv b$ "a and b are the same" $(\lambda x.\, fx)\, y \equiv fy$
- $a :\equiv b$ same, by definition

We cannot ask or prove these statements *in* the language.

Exercise: Which of the following can we ask internally?

(1) Is 8 a natural number? no, b/c $8 : \mathsf{Nat}$ is judgment

(2) Is 8 a prime number? yes, b/c isPrime(8)

(3) Is "hello" a prime number? no, isPrime("hello") is not a type

(Identity *a.k.a.* equality *a.k.a.* identification *a.k.a.* path) types

- if $a, b : A$ then $a = b$ type "formation"
- given $a : A$ we have refl : $a = a$ "introduction"
- given $C : (a\, b : A) \to (a = b) \to \text{Type}$ "elimination"
  and $C\ a\ a$ refl (for all $a : A$)
  we get $C\ a\ b\ p$ (for all $a, b : A$, $p : a = b$)] aka path induction
- applying this rule and asking for the refl case gives back the assumption "computation"

sym : (a b : A) → a = b → b = a

sym a a refl :≡ refl

here: C a b p :≡ (b = a)

computation: sym a a refl ≡ refl

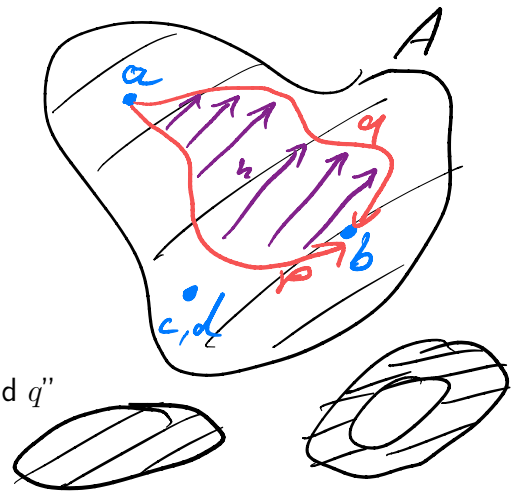
Model: types as spaces

The homotopical interpretation of dependent type theory works roughly like this:

judgment	interpretation
A type	"A is a space"
$a : A$	"a is a point in space A"
$c \equiv d$	"c and d are the same point"
$p : a = b$	"p is a path between a and b"
$h : p = q$	if p and q are equalities $a = b$: "h is a homotopy between p and q"

path in A

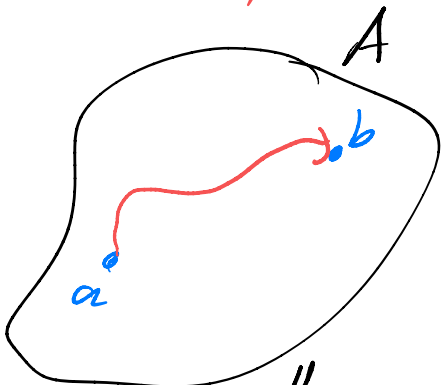
path in $a = b$



Types as spaces, examples: symmetry, transitivity

sym : $(a\,b : A) \to a = b \to b = a$

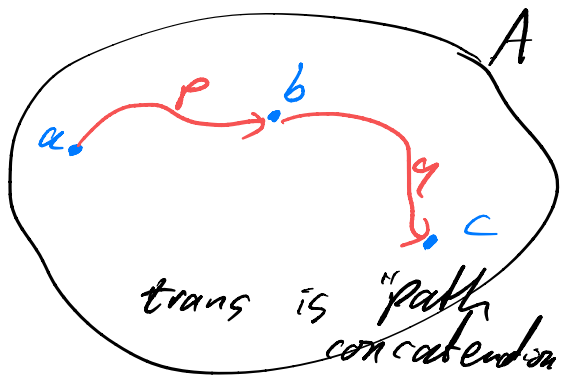
sym a b p refl :≡ refl



sym is "path reversal"

trans : $(a\,b\,c : A) \to$
$a = b \to b = c \to a = c$

trans a b c p q :≡ q



trans is "path concatenation"

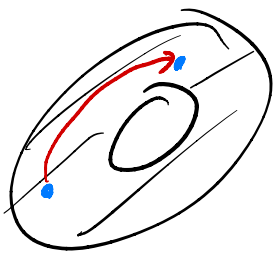
Types as spaces, example: uniqueness of identity proofs

Can we prove this for all types A?

UIP

$$\mathsf{uip} : (a\, b : A) \to (p\, q : a = b) \to p = q$$

uip a b p q :≡ ?



A quiz

given in addition:

Given A : Type with $x, y : A$

$B : (a\, b : A) \to (p : a = b) \to \mathsf{Type}$

$C : (p : x = y) \to \mathsf{Type}$

$D : (a : A) \to (p : a = y) \to \mathsf{Type}$

$e_1 : (a : A) \to B\ a\ a\ \mathrm{refl}_a$

$e_3 : D\ \mathrm{refl}_y$

Below are three statements which we may want to prove. Which is (probably) the most difficult and which the easiest?

$\mathsf{exercise}_1 : \quad (a\, b : A) \to (p : a = b) \to B\, a\, b\, p$

$\mathsf{exercise}_2 : \quad (p : x = y) \to C\, p$

$\mathsf{exercise}_3 : \quad (a : A) \to (p : a = y) \to D\, a\, p$

after path induction, we need to show:

$(a : A) \to B\ a\ a\ \mathrm{refl}_a$

(nothing we can do)

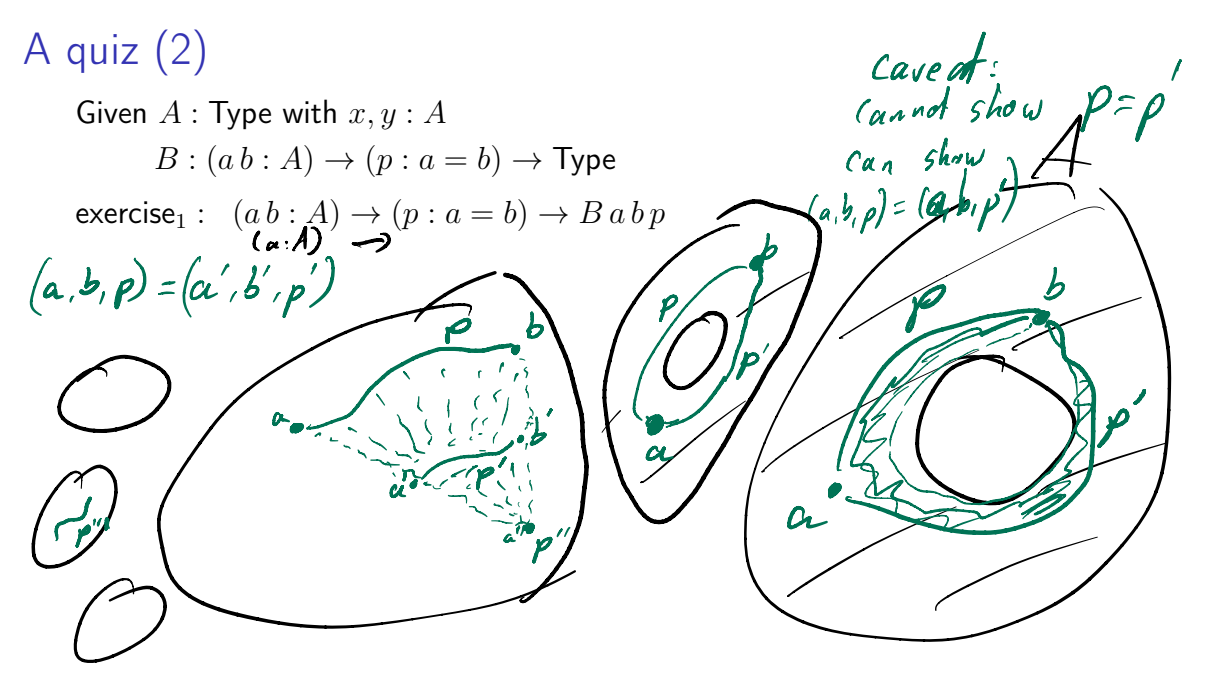
$D\ \mathrm{refl}_y$

A quiz (2)

Given A : Type with $x, y : A$

$$B : (a\, b : A) \to (p : a = b) \to \text{Type}$$

exercise_1 : $(a\, b : A) \to (p : a = b) \to B\, a\, b\, p$

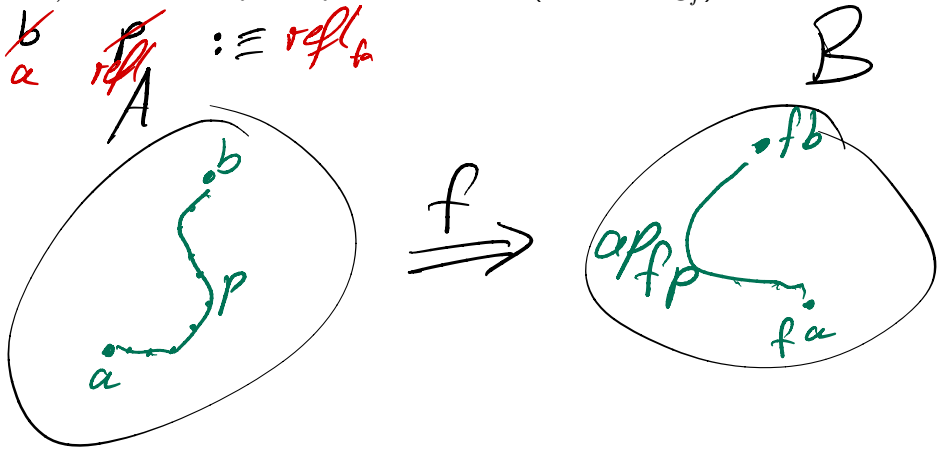


Types as spaces, examples: map on paths

Given $f : A \to B$, we have:

$\mathsf{ap}_f : (a\,b : A) \to a = b \to f\,a = f\,b$ (a.k.a. cong_f)

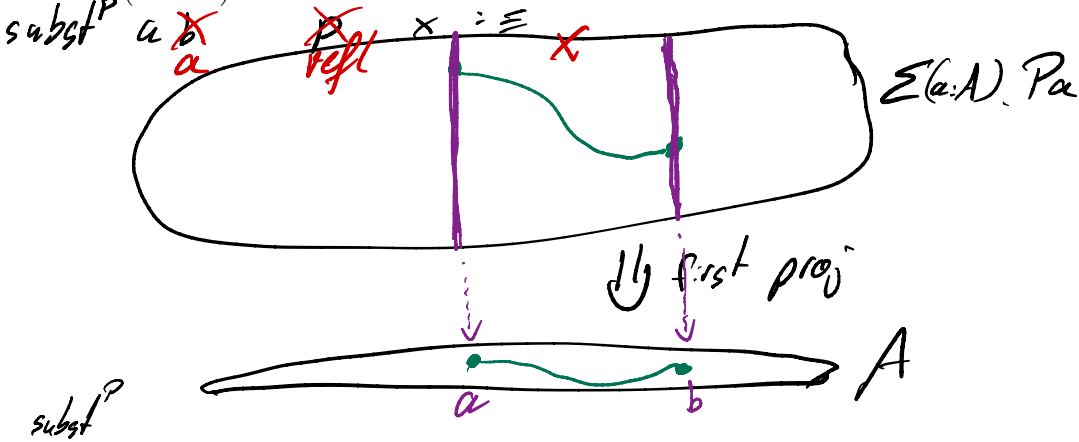
$\mathsf{ap}_f\ a\ b\ p :\equiv \mathsf{refl}_{fa}$



Types as spaces, example: substitution (a.k.a. transport)

Given a type A and $P : A \to \text{Type}$, we have:

$\text{subst}^P : (a\, b : A) \to a = b \to P\, a \to P\, b$



Isomorphisms and Equivalences

When is $f : A \to B$ an isomorphism?

$$\mathsf{isIso}(f) :\equiv \Sigma\, g : B \to A$$
$$\Sigma\, \alpha : (a:A) \to g(fa) = a$$
$$\beta : (b:B) \to f(gb) = b$$

question: given $a : A$, how do you show $f(g(fa)) = fa$?

solution: $\beta(fa)$ or $ap_f(\alpha a)$

$\Rightarrow$ ambiguity (which we usually want to avoid) / eg: (in category theory) $h \xrightarrow{} g \xrightarrow{} f$

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

Isomorphisms and Equivalences (2)

Recall from yesterday: For $f : A \to B$, we write $\mathsf{isIso}(f)$ [HoTT book: $\mathsf{qinv}(f)$] if we have:

- $g : B \to A$
- $\alpha : (a : A) \to g(f\,a) = a$
- $\beta : (b : B) \to f(g\,b) = b$.

how to prove $f(g(fa)) = fa$?
two solutions: $\mathsf{ap}_f(\alpha\, a)$
$\beta(fa)$

We say that f is an *equivalence* and write $\mathsf{isEqv}(f)$ [HoTT book: $\mathsf{isequiv}(f)$] if, *in addition*, we have

- $h : (a : A) \to \beta(f\,a) = \mathsf{ap}_f(\alpha\, a)$

Facts: $\mathsf{isEqv}(f) \to \mathsf{isIso}(f)$
$\mathsf{isIso}(f) \to \mathsf{isEqv}(f)$

$h' : (b : B) \to \alpha(g\,b) = \mathsf{ap}_g(\beta b)$

sometimes: $p\ q : \mathsf{isIso}(f)$ s.t. $p \neq q$

always: $(p\ q : \mathsf{isEqv}(f)) \to p = q$

fix g: α not unique, $\alpha \neq \alpha'$, $(g, \alpha) = (g, \alpha')$

Characterisation of path spaces: binary products

Lemma: Given $a_1, a_2 : A$ and $b_1, b_2 : B$,

$$(a_1, b_1) = (a_2, b_2) \quad \simeq \quad (a_1 = a_2) \times (b_1 = b_2)$$

equality / path in $A \times B$; path in A ; path in B

$$\mathrm{refl} \quad \mapsto \quad (\mathrm{refl}, \mathrm{refl})$$

$$(x\ y : A \times B) \to (x = y) \simeq (\mathrm{fst}\, x = \mathrm{fst}\, y) \times (\mathrm{snd}\, x = \mathrm{snd}\, y)$$

Characterisation of path spaces: Σ-types

Lemma: Given $a_1, a_2 : A$ and $b_1 : B(a_1)$ and $b_2 : B(a_2)$,

$$(a_1, b_1) = (a_2, b_2) \quad \simeq \quad \Sigma(p : a_1 = a_2).(\mathsf{subst}^B\, p\, b_1 = b_2)$$

(Looks more complicated but is essentially the same as for $\times$.
After path induction, the subst disappears.)

It can happen that $(a, b, p) = (a, b, p')$
but $p \neq p'$

Characterisation of path spaces: function- and Π-types

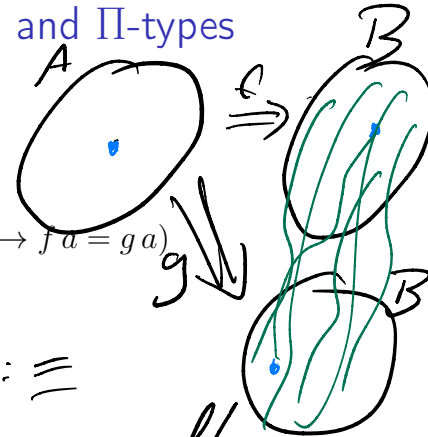
Given $f, g : (a : A) \to B\,a$, the principle of

function extensionality

says that the map

$$\mathsf{happly}_{f,g} : \quad (f = g) \quad \to \quad ((a : A) \to f\,a = g\,a)$$

is an equivalence.



happly f g p a :≡

happly f f refl a :≡ $\mathsf{refl}_{f\,a}$

funext : $(a : A) \to f\,a = g\,a) \to f = g$

On function extensionality

Why $\mathsf{happly}_{f,g} : (f = g) \to ((a : A) \to f\,a = g\,a)$? Compare:

- $\mathsf{strFunext} : \mathsf{isEqv}(\mathsf{happly}_{f,g})$ ← only one
- $\mathsf{funext} : ((a : A) \to f\,a = g\,a) \to f = g$ ← could be multiple

$$\left(\forall A B f g.\ ((a:A) \to fa = ga) \to f = g\right) \leftrightarrow \mathsf{isEqv}(\mathsf{happly}_{f,g})$$

(Voevodsky)

say you have: $h : (a:A) \to fa = ga$

funext gives $p : f = g$

happly gives $q : fa = ga$

Characterisation of path spaces: universe

Given A, B : Type, the principle of

univalence

says that the map

$$\mathsf{id2eqv}_{A,B} : \quad (A = B) \quad \to \quad (A \simeq B)$$

is an equivalence.

$(A \simeq B) :\equiv \Sigma(f : A \to B).\ \mathsf{isEqv}(f)$

id2eqv A B refl : $\equiv$ id, ...

A A

ua : $(A \simeq B) \to (A = B)$

Consequence of univalence

Univalence implies function extensionality (Voevodsky).

Some types have non-trivial higher paths. Example: Universe via Bool = Bool

example: swap : Bool → Bool

swap true :≡ false

swap false :≡ true

can show: e : isEqv(swap)

⇒ get: ua(swap, e) : Bool = Bool

Lemma: ua(swap, e) ≠ refl

assume =, then

id2eqv(ua(swap, e)) = id2eqv(refl)

⇒ (swap, e) = (id, —)

↯ swap true = id true ⇒ false = true ↯

Another example

$\mathbb{Z}$ integers (eg $\mathbb{N} + \mathbb{N}$)

$suc : \mathbb{Z} \to \mathbb{Z}$

$pred : \mathbb{Z} \to \mathbb{Z}$

$\Rightarrow e : isEqv(suc)$

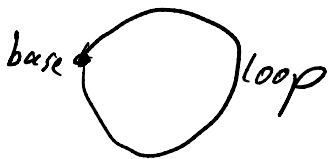
$\Rightarrow ua(suc\, e) : \mathbb{Z} = \mathbb{Z}$

Higher inductive types

Simple examples: circle, torus, interval

circle: data $\mathbb{S}^1$ where
point constr. → base : $\mathbb{S}^1$
path constr. → loop : base = base

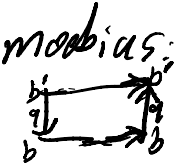
in "space model"



interval:
data I where
N : I
S : I
p : N = S

result: I ≃ Unit

möbius:



data M where
b : M
q : b = b
t : q = q^{-1}
subst along
sym q

Properties of the circle

data $\mathbb{S}^1$ where
  base : $\mathbb{S}^1$
  loop : base = base

non-dep elim. principle:

$elim : (A : Type) \to (a_0 : A) \to (p : a_0 = a_0) \to S^1 \to A$

computation:

$elim\ A\ a_0\ p\ base \equiv a_0$

$ap_{elim\ A\ a_0\ p}(loop) \equiv p$

Lemma: $loop \neq refl_{base}$

Proof:

$f : S^1 \to Type$

$base \mapsto Bool$

$loop \mapsto ua(swap, e)$

[i.e. $f :\equiv elim\ Type\ Bool\ (ua(swap, e))$]

If $loop = refl$

$\Rightarrow ap_f(loop) = ap_f(refl)$

$\Rightarrow ua(swap, e) = refl$

$\Rightarrow idtoeqv(ua(swap, e)) = idtoeqv(refl)$

$\Rightarrow swap = id$

Loop space / fundamental group of the circle

end of lecture

Lemma: $(base = base) \simeq \mathbb{Z}$

"fundamental group"

This is called "synthetic homotopy theory" (axiomatic)

Proof: Licata-Shulman (also HoTT book Sec 8.1)