Homotopy Type Theory

Marc Bezem[1]

[1]Department of Informatics
Bergen University

Spring 2016

Practical Matters

- Lecturer: Marc Bezem (Baker Hall 152)
- Occasionally: guest lecturers?
- Place and time:
  - Baker Hall 150
  - Monday 13h30 – 16h30, exercises/lectures
  - Wednesday 13h30 – 16h30, exercises/lectures
- Textbook (link): Homotopy Type Theory
- Lecture Notes: these slides HoTT.pdf + DTUA.pdf

Untyped Lambda Calculus

- A formalism for binding variables and substitution
- Binder Zoo: quantification, integrals, generalized products, functions, ...
- Terms: $M, N ::= x \mid MN \mid \lambda x.\, M$
- Examples of terms: $y, \lambda x.\, x, \lambda x.\, (\lambda y.\, x), \lambda x.\, (\lambda y.\, y(yx))$
- Binding x in M (*lambda abstraction*): $\lambda x.\, M$
- Intention to unbind (*application*): MN
- Actual unbinding (β-*contraction*): $(\lambda x.\, M)N \to M[x := N]$
- Substitution:
  - $x[x := N] \equiv N$
  - $y[x := N] \equiv y \ (y \not\equiv x)$
  - $(MM')[x := N] \equiv (M[x := N])(M'[x := N])$
  - $(\lambda y.\, M)[x := N] \equiv \lambda y.\, (M[x := N]) \ (y \not\equiv x$, avoiding *caption*)

Terminology and Notation

- Avoid caption by renaming bound variables
- Technically better, but hard to read (De Bruijn): f.e. $\lambda\lambda 1$
- Application left-associative :
  $M_1 M_2 \ldots M_n \equiv (\ldots(M_1 M_2)\ldots M_n)$
- Abstraction right-associative :
  $\lambda x_1 x_2 \ldots x_n . M \equiv \lambda x_1 . (\lambda x_2 . \ldots (\lambda x_n . M))$
- Convenient combination: $(\lambda x_1 x_2 \ldots x_n . M) M_1 M_2 \ldots M_n$
- A *free* variable in a term is a variable that is not *bound* by a λ
- Reducible expression (*redex*): $(\lambda x . M) N$

Reduction

- Examples of contraction: $(\lambda xy.\, x)z \to \lambda y.\, z$, $(\lambda xy.\, x(xy))f \to \lambda y.\, f(fy)$, $(\lambda x.\, xx)(\lambda x.\, xx) \to \ldots$
- Reduction is contraction of a subterm (ind. def.): if $M \to M'$, then $MN \to M'N$, $NM \to NM'$, $\lambda x.\, M \to \lambda x.\, M'$
- Reductions may be iterated: $\stackrel{*}{\to}$ is the reflexive and transitive closure of $\to$ (zero steps, one-step or many-step reduction)
- Convertibility: $=_\beta$ is the transitive, symmetric and reflexive closure of $\to$
- Convertibility is a congruence wrt. application and abstraction
- THEOREM (*confluence*): if $M =_\beta N$, then M and N have a *common reduct* R, that is, $M \stackrel{*}{\to} R \stackrel{*}{\leftarrow} N$
- COR: lambda calculus is consistent, $\lambda xy.\, x \neq_\beta \lambda xy.\, y$

Useful encodings

- Booleans: true $\equiv \lambda xy.\, x$, false $\equiv \lambda xy.\, y$
  - Negation: $\neg \equiv \lambda b.\, b(\text{false})(\text{true})$
  - Conjunction: $\wedge \equiv \lambda b.\, b(\lambda x.\, x)(\lambda x.\, \text{false})$
  - Remarkable: $\wedge\, \text{false}\, x =_\beta \text{false}$, but NOT $\wedge\, x\, fal =_\beta \text{false}$
- Natural numbers (Church): $\underline{0} \equiv \lambda fx.\, x$, $\underline{1} \equiv \lambda fx.\, fx$, $\underline{2} \equiv \lambda fx.\, f(fx)$ ... , in general $\underline{n} \equiv \lambda fx.\, f^n x$
  - Successor: $S \equiv \lambda nfx.\, nf(fx)$ (indeed $S\underline{0} =_\beta \underline{1}$, $S\underline{1} =_\beta \underline{2}, \ldots$)
  - Addition: $+ \equiv \lambda nm.\, nSm$ ($+\underline{0}x =_\beta x$, NOT $+x\underline{0} =_\beta x$)
  - Multiplication: $* \equiv \lambda nm.\, n(+m)\underline{0}$
  - Exponentiation: $e \equiv \lambda nm.\, m(*n)\underline{1}$
- Fixpoint operator: $Y \equiv \lambda f.\, ((\lambda x.\, f(xx))(\lambda x.\, f(xx)))$
- COR: lambda calculus is Turing complete
- COR: lambda calculus is ‘inconsistent’, $Y(\neg) =_\beta \neg(Y(\neg))$

Chapter 1 — Type Theory

- Judgment: $t : T$ (logical stuff inside t, T)
- Assumption: judgment of the form $x : T$ (x a variable)
- Context: list of assumptions Γ (with different variables)
- Typing: a judgment in a context, notation $\Gamma \vdash t : T$
- Example: $f : A \to A,\ x : A \vdash f(fx) : A$
- Type theory: system of rules to derive typings
- Two notions of equality:
  - definitional (or judgmental) equality: $a \equiv b$ $(\beta, \eta, \iota, \delta, \ldots)$
  - propositional equality (logical operations): a *type* $a =_A b$

Function types

- If A and B are types, then so is their function type $A \to B$
- Introduction rule:

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

- Elimination rule:

$$\frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash fa : B}$$

- No product types needed (but they will come nevertheless):

$$\frac{\dfrac{\Gamma, x : A, y : B \vdash t : C}{\Gamma, x : A \vdash \lambda y{:}B.\, t : B \to C}}{\Gamma \vdash \lambda x{:}A.\, \lambda y{:}B.\, t : A \to (B \to C)}$$

Universes and families of types

- Universe of types: $\mathcal{U}$, 'A is a type' becomes judgment $A : \mathcal{U}$
- Rather not $\mathcal{U} : \mathcal{U}$, but $\mathcal{U}_0 : \mathcal{U}_1, \ldots$
- Formation rule for $\to$:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A \to B : \mathcal{U}}$$

- Introduction rule for functions:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash B : \mathcal{U} \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x{:}A.\, t : A \to B}$$

- This includes:

$$\frac{\Gamma \vdash U : \mathcal{U}_1 \quad \Gamma \vdash U' : \mathcal{U}_1}{\Gamma \vdash (U \to U') : \mathcal{U}_1} \qquad \frac{\vdash \mathcal{U}_0 : \mathcal{U}_1 \quad A : \mathcal{U}_0 \vdash (A \to A) : \mathcal{U}_0}{\vdash (\lambda A{:}\mathcal{U}_0.\, A \to A) : \mathcal{U}_0 \to \mathcal{U}_0}$$

- Type family: $B : A \to \mathcal{U}$ with $A : \mathcal{U}$, example $B \equiv \lambda n{:}Nat.\, \mathbb{R}^n$

Dependent product types, aka Π-types

- Given $A : \mathcal{U}$, $B : A \to \mathcal{U}$ and $a : A$, we have $Ba : \mathcal{U}$
- Dependent product type: $\Pi x{:}A.\, Bx$ (or $\Pi\, A\, B$)
- Formation rule for Π-type:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash B : A \to \mathcal{U}}{\Gamma \vdash \Pi x{:}A.\, Bx : \mathcal{U}}$$

- Introduction rule for Π-type:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash B : A \to \mathcal{U} \quad \Gamma, x : A \vdash t : Bx}{\Gamma \vdash \lambda x{:}A.\, t : \Pi x{:}A.\, Bx}$$

- Π-type is the type of dependent functions (co-domain varies), examples: element of infinite product, $\lambda n{:}Nat.\, \vec{0}(n) : \Pi\, Nat\, B$
- Elimination rule for Π-types:

$$\frac{\Gamma \vdash f : \Pi x{:}A.\, Bx : \mathcal{U} \quad \Gamma \vdash a : A}{\Gamma \vdash fa : Ba}$$

so, e.g., $\vec{0}(3) : \mathbb{R}^3$

Type constructors

- Type Zoo is ever extending (social process!)
- Type constructors, so far: $\rightarrow$, Π
- Actually, $A \rightarrow B$ is a special case: $\Pi\, A\, (\lambda x{:}A.\, B)$
- How to systematically manage the Type Zoo
  - Name a new type constructor
  - Formation: how to construct types with the new constructor
  - Introduction: how to construct elements of the new type
  - Elimination: how to destruct (work with) these elements
  - Computation: how to simplify desconstruction (β, ι)
  - Optional: uniqueness principle for condestruction (η)
- Example: $\rightarrow$, abstraction, application, β-, η-reduction
  $(\lambda x.\, t)a \rightarrow_\beta t[x := a]$, $\lambda x.\, fx \rightarrow_\eta f$

Products (1)

- Type constructor: $\times$, idea: cartesian product
- Formation rule for (non-dependent) product:

$$\frac{A : \mathcal{U} \quad B : \mathcal{U}}{A \times B : \mathcal{U}}$$

- Introduction rule for product:

$$\frac{a : A : \mathcal{U} \quad b : B : \mathcal{U}}{(a, b) : A \times B}$$

- Elimination rules for product:

$$\frac{p : A \times B}{pr_1 p : A} \qquad \frac{p : A \times B}{pr_2 p : B}$$

- Computation rules for pairs and projections:
  - $pr_1(a, b) \to_\iota a, \; pr_2(a, b) \to_\iota b$
  - Optional: $(pr_1 p, pr_2 p) \to_\eta p$

Products (2)

- We can infer the following jugment:

$$\lambda f{:}A{\rightarrow}B{\rightarrow}C.\, \lambda p{:}A{\times}B.\, f\,(pr_1 p)(pr_2 p) : (A{\rightarrow}B{\rightarrow}C) \rightarrow (A{\times}B{\rightarrow}C)$$

- As an alternative to the pr_i's, we can postulate:

$$rec_{A\times B} : \Pi C{:}\mathcal{U}.\,(A{\rightarrow}B{\rightarrow}C) \rightarrow (A{\times}B{\rightarrow}C)$$

- ... and recover the projections:
  - $pr_1 \equiv rec_{A\times B}\, A\,(\lambda a{:}A.\,\lambda b{:}B.\,a) : A{\times}B{\rightarrow}A$
  - $pr_2 \equiv rec_{A\times B}\, B\,(\lambda a{:}A.\,\lambda b{:}B.\,b) : A{\times}B{\rightarrow}B$
- Computation rule for the recursor:

$$rec_{A\times B}\, C\, g\,(a, b) \rightarrow_\iota g\, a\, b$$

- This works well in general, we like recursors

Products (3)

- Syntactic sugar can impair understanding: $\mathit{pair} := g$

$$\mathit{rec}_{A\times B}\; C\; g\; (\mathit{pair}\; a\; b) \to_\iota g\; a\; b$$

- Keep in mind: recursor replaces constructor by other term
- Still possible:
  $A = \{a\}$, $B = \{b\}$, $A\times B = \{(a, b), p\}$, $\mathit{pr}_1 p = a$, $\mathit{pr}_2 p = b$
- Will be solved (propositionally) by an induction principle (dependent version of $\mathit{rec}_{A\times B}$)
- This also helps: $(\mathit{pr}_1 p, \mathit{pr}_2 p) \to_\eta p$
- Q: how does this relate to cartesian products in category theory?

Products (0)

- Formation: $\mathbf{1} : \mathcal{U}$, idea: empty product
- Introduction: $\star : \mathbf{1}$
- Elimination: $\mathit{rec}_{\mathbf{1}} : \Pi C{:}\mathcal{U}.\, C \to \mathbf{1} \to C$
- Computation: $(\mathit{rec}_{\mathbf{1}}\ C\ c\ \star) \to_{\iota} c$
- Q: $\star \to_{\eta}$?

Induction

- We can infer the following jugment (short):

$$f : \Pi x{:}A.\,\Pi y{:}B.\,C(x,y) \vdash \lambda p.\,f(pr_1 p)(pr_2 p) : \Pi p{:}A\times B.\,C(pr_1 p, pr_2 p)$$

- ... but NOT the following jugment:

$$f : \Pi x{:}A.\,\Pi y{:}B.\,C(x,y) \vdash \lambda p.\,f(pr_1 p)(pr_2 p) : \Pi p{:}A\times B.\,C\ p$$

- ... unless we have $(pr_1 p, pr_2 p) \to_\eta p$, or postulate:

$$ind_{A\times B} : \Pi C{:}A\times B\to\mathcal{U}.\,((\Pi x{:}A.\,\Pi y{:}B.\,C(x,y)) \to \Pi p{:}A\times B.\,C\ p)$$

- Computation rule for the dependent eliminator (induction):

$$ind_{A\times B}\ C\ f\ (a,b) \to_\iota f\ a\ b$$

- We like induction (but it does not give us $(pr_1 p, pr_2 p) \to_\eta p$)

Induction on $\star$ and more

- Formation: $\mathbf{1} : \mathcal{U}$, idea as a set: $\{\star\}$
- Introduction: $\star : \mathbf{1}$
- Dependent elimination: $\mathit{ind}_{\mathbf{1}} : \Pi C{:}\mathbf{1} \to \mathcal{U}.\, C\,\star \to \Pi x{:}\mathbf{1}.\, C\,x$
- Computation: $(\mathit{ind}_{\mathbf{1}}\ C\ c\ \star) \to_{\iota} c$
- Provable (short):
  $\mathit{refl}_{\star} : (\star =_{\mathbf{1}} \star) \vdash \mathit{ind}_{\mathbf{1}}\,(\lambda x{:}\mathbf{1}.\,(x =_{\mathbf{1}} \star))\,\mathit{refl}_{\star} : \Pi x{:}\mathbf{1}.\,(x =_{\mathbf{1}} \star)$
- Computation: $\mathit{ind}_{\mathbf{1}}\,(\lambda x{:}\mathbf{1}.\,(x =_{\mathbf{1}} \star))\,\mathit{refl}_{\star}\,\star \to_{\iota} \mathit{refl}_{\star}$
- Define: $C \equiv \lambda p{:}A \times B.\,(pr_1 p, pr_2 p) =_{A \times B} p$
- On the blackboard: inhabitant of $\Pi p{:}A \times B.\, C\,p$
- More on equality types and *refl* later

Dependent pairs and Σ-types

- Dependent pair (a, b): type of b depends on a, $b : Ba$
- Σ-type, type of dependent pairs: $\Sigma x{:}A.\, Bx$ (or $\Sigma\, A\, B$)
- $\Sigma x{:}A.\, Bx$ where $B : A \to U$ can be seen as an indexed sum
- Formation rule for Σ-type:

$$\frac{A : \mathcal{U} \quad B : A \to \mathcal{U}}{\Sigma x{:}A.\, Bx : \mathcal{U}}$$

- Introduction rule for Σ-type:

$$\frac{A : \mathcal{U} \quad B : A \to \mathcal{U} \quad a : A \quad b : Ba}{(a, b) : \Sigma x{:}A.\, Bx}$$

- Elimination rules for Σ-types:

$$\frac{d : \Sigma x{:}A.\, Bx : \mathcal{U}}{pr_1 d : A} \qquad \frac{d : \Sigma x{:}A.\, Bx : \mathcal{U}}{pr_2 d : B(pr_1 d)}$$

Recursion and induction for Σ-types

- Define the recursor:

$$rec_{\Sigma A B} : \Pi C{:}\mathcal{U}.\, (\Pi x{:}A.\, (Bx \to C)) \to (\Sigma\, A\, B \to C)$$

- ... and recover the first projection:
  - $pr_1 \equiv rec_{\Sigma A B}\, A\, (\lambda ab.\, a) : \Sigma\, A\, B \to A$
- Define the dependent eliminator:

$$ind_{\Sigma AB} : \Pi C{:}(\Sigma AB \to \mathcal{U}).\, (\Pi x{:}A.\, \Pi y{:}Bx.\, C(x, y)) \to (\Pi p{:}\Sigma\, A\, B.\, Cp)$$

- ... and recover also the second (dependent) projection:

$$pr_2 \equiv ind_{\Sigma A B}\, (\lambda p{:}\Sigma\, A\, B.\, B(pr_1 p))\, (\lambda ab.\, b) : \Pi p{:}\Sigma\, A\, B.\, B(pr_1 p)$$

- Computation rules: $rec/ind_{\Sigma A B}\, C\, g\, (a, b) \to_\iota g\, a\, b$

The Axiom of Choice

- For $A : \mathcal{U},\ B : \mathcal{U},\ R : A \to B \to \mathcal{U}$, find ac with

$$\mathsf{ac} : (\Pi x{:}A.\, \Sigma y{:}B.\, R\,x\,y) \to \Sigma f{:}A{\to}B.\, \Pi x{:}A.\, R\,x\,(fx)$$

- we discuss this on the blackboard (see 1.6 of the book)

Use of Σ-types (and other types)

- A group is a set with *operations* satisfying *axioms*
- A Σ-type: $\Sigma A{:}\mathcal{U}.\,(A \to A \to A) \times ((A \to A) \times A)$
- This captures only the signature
- We let products and pairs associate to the right
- We assume sensible precedence rules
- Taking one group axiom into account:

$$\Sigma A{:}\mathcal{U}.\,\Sigma m{:}A \to A \to A.\,\Sigma i{:}A \to A.\,\Sigma u{:}A.\,(\Pi x{:}A.\,mux =_A x)$$

- More axioms:

$$\Sigma A{:}\mathcal{U}.\,\Sigma m{:}A \to A \to A.\,\Sigma i{:}A \to A.\,\Sigma u{:}A.\,(Ax1 \times Ax2 \times \ldots)$$

- This can be considered to be the type of groups

Coproducts

- Type constructor: $+$, idea: disjoint union
- Formation rule for coproduct:

$$\frac{A : \mathcal{U} \quad B : \mathcal{U}}{A + B : \mathcal{U}}$$

- Introduction rules for coproduct:

$$\frac{a : A : \mathcal{U}}{\mathit{inl}\ a : A + B} \qquad \frac{b : B : \mathcal{U}}{\mathit{inr}\ b : A + B}$$

- Elimination rule for coproduct:

$$\frac{s : A + B \quad f : A \to C \quad g : B \to C}{\mathit{case}\ s\ f\ g : C}$$

- Computation rules for coproducts and injections:
  - $\mathit{case}\ (\mathit{inl}\ a)\ f\ g \to_\iota fa,\ \mathit{case}\ (\mathit{inr}\ b)\ f\ g \to_\iota gb$
  - Optional: $\mathit{case}\ s\ \mathit{inl}\ \mathit{inr} \to_\eta s$

Recursion and induction for $+$

- We prefer a recursor:

$$rec_{A+B} : \Pi C{:}\mathcal{U}.\,(A{\to}C) \to (B{\to}C) \to A{+}B \to C$$

- ... and define: $case\ \_\ f\ \ g \equiv rec_{A+B}\ f\ \ g$
- We define a dependent eliminator ind_{A+B} of type:

$$\Pi C{:}A{+}B{\to}\mathcal{U}.\,(\Pi x{:}A.\,C(inl\,x)) \to (\Pi y{:}B.\,C(inr\,y)) \to \Pi s{:}A{+}B.\,C\,s$$

- Computation rules:
  - $rec/ind_{A+B}\ C\ f\ g\ (inl\ a) \to_\iota fa$
  - $rec/ind_{A+B}\ C\ f\ g\ (inr\ b) \to_\iota gb$

The empty coproduct

- Formation: $\mathbf{0} : \mathcal{U}$, set analogue: $\emptyset$
- Introduction: nope
- Elimination:
  - $rec_{\mathbf{0}} : \Pi C{:}\mathcal{U}.\, \mathbf{0} \to C$
  - $ind_{\mathbf{0}} : \Pi C{:}\mathbf{0}{\to}\mathcal{U}.\, \Pi x{:}\mathbf{0}.\, C\,x$
- Computation rules: none ($rec_{\mathbf{0}}\, C\, s \to\ ?$)
- Induction principle known as *ex falso [[sequitur] quodlibet] (C)*
- ($rec_{\mathbf{0}}\, \mathbf{0}$) and ($\lambda x{:}\mathbf{0}.\, x$) are only extensionally equal

Booleans

- $\mathbf{2} = \mathbf{1} + \mathbf{1}$ (p. 45), beating Principia Mathematica (p. 362!)
- Formation: $\mathbf{2} : \mathcal{U}$
- Introduction: $0_{\mathbf{2}} : \mathbf{2}$, $1_{\mathbf{2}} : \mathbf{2}$
- Elimination:
  - $\mathit{rec}_{\mathbf{2}} : \Pi C{:}\mathcal{U}.\, C \to C \to \mathbf{2} \to C$
  - $\mathit{ind}_{\mathbf{2}} : \Pi C{:}\mathbf{2}{\to}\mathcal{U}.\, C(0_{\mathbf{2}}) \to C(1_{\mathbf{2}}) \to \Pi x{:}\mathbf{2}.\, C\, x$
- Computation:
  - $\mathit{ind}/\mathit{rec}_{\mathbf{2}}\; C\; c_0\; c_1\; 0_{\mathbf{2}} \to c_0$, $\mathit{ind}/\mathit{rec}_{\mathbf{2}}\; C\; c_0\; c_1\; 1_{\mathbf{2}} \to c_1$
- Exercise:
  - $\mathit{refl}_0{:}(0_{\mathbf{2}}{=}_{\mathbf{2}}0_{\mathbf{2}}), \mathit{refl}_1{:}(1_{\mathbf{2}}{=}_{\mathbf{2}}1_{\mathbf{2}}) \vdash ? : \Pi x{:}\mathbf{2}.\, (x{=}_{\mathbf{2}}0_{\mathbf{2}}) + (x{=}_{\mathbf{2}}1_{\mathbf{2}})$
- Discussion:
  - $(\Pi\; \mathbf{2}\; (\mathit{Rec}_{\mathbf{2}}\, \mathcal{U}\; A\, B))$, $(\Sigma\; \mathbf{2}\; (\mathit{Rec}_{\mathbf{2}}\, \mathcal{U}\; A\, B))$
  - $A \to \mathbf{2}$: ‘decidable subsets’ of $A : \mathcal{U}$

Natural numbers

- Formation: $\mathbb{N} : \mathcal{U}$
- Introduction: $0 : \mathbb{N}$ and $Sx : \mathbb{N}$ if $x : \mathbb{N}$
- Elimination:
  - $it_{\mathbb{N}} : \Pi C{:}\mathcal{U}.\, C \to (C \to C) \to \mathbb{N} \to C$
  - $rec_{\mathbb{N}} : \Pi C{:}\mathcal{U}.\, C \to (\mathbb{N} \to C \to C) \to \mathbb{N} \to C$
  - $ind_{\mathbb{N}} : \Pi C{:}\mathbb{N}{\to}\mathcal{U}.\, C0 \to (\Pi x{:}\mathbb{N}.\, Cx \to C(Sx)) \to \Pi x{:}\mathbb{N}.\, Cx$
- Computation:
  - $it_{\mathbb{N}}\, Ccf0 \to_{\iota} c$, $it_{\mathbb{N}}\, Ccf(Sx) \to_{\iota} f(it_{\mathbb{N}}\, Ccfx)$
  - $rec_{\mathbb{N}}\, Ccf0 \to_{\iota} c$, $rec_{\mathbb{N}}\, Ccf(Sx) \to_{\iota} fx(rec_{\mathbb{N}}\, Ccfx)$
  - induction $ind_{\mathbb{N}}$ has the same rules as $rec_{\mathbb{N}}$
- Interdefinable: $it_{\mathbb{N}}$ (*iterator*) and $rec_{\mathbb{N}}$ (*primitive recursion*)

Useful encodings

- Example: $\mathit{double} \equiv \mathit{it}_{\mathbb{N}}\, \mathbb{N}\, 0\, (\lambda x{:}\mathbb{N}.\, S(Sx))$
  - $\mathit{double}\, 0 \to_\iota 0$
  - $\mathit{double}\, (Sn) \to_\iota (\lambda x{:}\mathbb{N}.\, S(Sx))(\mathit{double}\, n) \to_\beta S(S(\mathit{double}\, n))$
- Right-recursive addition: $\mathit{add} \equiv \lambda x{:}\mathbb{N}.\, \mathit{it}_{\mathbb{N}}\, \mathbb{N}\, x\, S$
- Left-recursive addition:
  $\mathit{adl} \equiv \mathit{it}_{\mathbb{N}}\, (\mathbb{N}{\to}\mathbb{N})\, (\lambda x{:}\mathbb{N}.\, x)\, (\lambda f{:}\mathbb{N}{\to}\mathbb{N}.\, S \circ f)$
  - $\mathit{adl}\, 0 \to_\iota \lambda x{:}\mathbb{N}.\, x$, so $\mathit{adl}\, 0\, m \to_\iota m$
  - $\mathit{adl}\, (Sn) \to_\iota S \circ (\mathit{adl}\, n)$, so
    $\mathit{adl}\, (Sn)\, m \to_\iota (S \circ (\mathit{adl}\, n))\, m \to_\beta S(\mathit{adl}\, n\, m)$
- Right-recursive multiplication:

$$\mathit{mult} \equiv \lambda x, y{:}\mathbb{N}.\, \mathit{it}_{\mathbb{N}}\, \mathbb{N}\, 0\, (\mathit{add}\, x)\, y$$

Proofs by induction

- We prove in the context ...
  - $\mathit{refl}_{\mathbb{N}} : \Pi x{:}\mathbb{N}.\,(x =_{\mathbb{N}} x)$ (later: axiom)
  - $\mathit{funcS} : \Pi x,y{:}\mathbb{N}.\,(x =_{\mathbb{N}} y) \to (Sx =_{\mathbb{N}} Sy)$ (later: provable)
- ... on the blackboard:
  - $\vdash ? : \Pi x{:}\mathbb{N}.\,(\mathit{add}\,0\,x =_{\mathbb{N}} x))$
  - $\vdash ? : \Pi x{:}\mathbb{N}.\,(\mathit{double}\,(\mathit{add}\,x\,(S0)) =_{\mathbb{N}} S(S(\mathit{double}\,x)))$
  - $\vdash ? : \Pi x{:}\mathbb{N}.\,(\mathit{add}\,x\,0 =_{\mathbb{N}} x)$ (no induction needed!)
- Discussion

Pattern Matching

- Instead of $f \equiv rec_{A+B}\, C\, g_0\, g_1$:
$$\begin{cases} f\,(inl\ a) &= g_0\ a \\ f\,(inr\ b) &= g_1\ b \end{cases}$$
- Instead of
$f \equiv rec_{A\times B}\, C\, \lambda a{:}A.\, \lambda b{:}B.\,$(a term of type C in a and b):
$$f\,(a, b) \;=\; (\text{a term of type } C \text{ in } a \text{ and } b)$$
- Instead of $double \equiv it_{\mathbb{N}} \mathbb{N}\, 0\, (\lambda x{:}\mathbb{N}.\, S(Sx))$:
$$\begin{cases} double\, 0 &= 0 \\ double\,(S\,x) &= S(S(double\, x)) \end{cases}$$
- ... and of course not
$$\begin{cases} f\, 0 &= 0 \\ f\,(S\,x) &= f\,(S(S(x))) \end{cases}$$

Propositions as Types

- Correspondence:

true	*false*	*if _ then _*	*not _*	*and*	*or*	*for all ...*	*exists ...*
$\mathbf{1}$	$\mathbf{0}$	$\_ \to \_$	$\_ \to \mathbf{0}$	$\times$	$+$	$\Pi x{:}A.\, Px$	$\Sigma x{:}A.\, Px$

- We prove some constructive tautologies on the blackboard
- E.g., $(\Pi x{:}A.\, \Pi y{:}A.\, (Px \to Qxy)) \to \Pi x{:}A.\, (Px \to \Pi y{:}A.\, Qxy)$

Identity Types

- Formation: $Id_A ab : \mathcal{U}$ if $A : \mathcal{U}$ and $a, b : A$
- Notation: $Id_A ab$ or $a =_A b$ or even just $a = b$
- Introduction: $refl : \Pi x{:}A.\, Id_A\, x\, x$, notation $refl_a$ for $(refl\ a)$
- Elimination: ind_{Id_A} has type 'for every unary predicate C on the path space of A, and every function mapping points x to a proof of $C(x, x, refl_x)$, there exists a function mapping paths (x, y, p) with $p : x =_A y$ to a proof of $C(x, y, p)$' (book!)
- Computation: $ind_{Id_A}\, C\, c\, x\, x\, refl_x \to_\iota c\, x$
- Example: with $C \equiv \lambda x, y{:}A.\, \lambda p{:}(x{=_A}y).\, (y =_A x)$ we get

$$ind_{Id_A}\, C\, refl : \Pi x, y{:}A.\, (x =_A y \to y =_A x)$$

Path induction and based path induction

- Path induction (two lines): $\Pi C{:}(\Pi x, y{:}A.\,(x =_A y \to \mathcal{U}))$.

$$(\Pi x{:}A.\; C\,x\,x\,\mathit{refl}_x) \to (\Pi x, y{:}A.\,\Pi p{:}(x{=_A}y).\; C\,x\,y\,p)$$

- Based path induction: $\Pi a{:}A.\,\Pi C{:}(\Pi y{:}A.\,(a =_A y \to \mathcal{U}))$.

$$C\,a\,\mathit{refl}_a \to (\Pi y{:}A.\,\Pi p{:}(a{=_A}y).\; C\,y\,p)$$

- Equivalence on the blackboard (book!):
  - Path induction follows easily from based path induction
  - Based path induction follows from one 'universal' instance of path induction, 'pulling out' $\Pi y{:}A.\,\Pi p{:}(a{=_A}y).\,\_$

$$D\,a\,y\,p \equiv \Pi C{:}(\Pi y{:}A.\,(a =_A y \to \mathcal{U})).\; C\,a\,\mathit{refl}_a \to C\,y\,p$$

Homotopy theory

- Path in a topological space X: continuous map $[0,1] \to X$
- Problem for the foundations: $[0,1]$
- HoTT $=$ synthetic homotopy theory
- Striking: induction for identity types fits very well
- Pointwise equality of paths too fine (2-way trip $=$ stay home?)
- Homotopy between $p, q : [0,1] \to X$: a continuous

  $$H : [0,1]\times[0,1] \to X \text{ such that } H(t,0) = p(t),\ H(t,1) = q(t)$$

- Picture: image of square 'fills space between p and q in X'
- Example: $h(t) = 1 - |1 - 2t|,\ H(t,z) = z \cdot h(t)$.

More homotopy theory

- Path $p : [0,1] \to X$, *start* point $p(0)$, *end* point $p(1)$
- Loop: $p(0) = p(1)$, loop at x_0: $p(0) = x_0 = p(1)$
- Based homotopy: as above, with $H(0,y) = x_0 = H(1,y)$
- Q: homotopic loops at x_0 that are not *based* homotopic?
- Fundamental group: loops at x_0 modulo based homotopy
- Homotopy between $f, g : X \to Y$: easy generalization
- Homotopy between X, Y in TOP: $f : X \to Y$, $g : Y \to X$, $f \circ g$ and id_Y homotopic, $g \circ f$ and id_X homotopic
- Invariant: homotopic spaces have isomorphic fundamental groups (for every $x \in X$ we have $\pi_1(X, x) \cong \pi_1(Y, f(x))$)

Higher dimensional paths

- Homotopies: "paths between paths", 2-dimensional paths
- Homotopies form a topological space (Q: how?)
- Paths between homotopies: 3-dimensional paths
- ... and so on, an infinite tower called ∞-groupoid
- Weak groupoid (only up to homotopy), not group
- Q: how to compose $p, q : [0, 1] \to X$ if $p(1) \neq q(0)$?

Homotopy type theory

- Path in a type A: $p : x =_A y$
- 2-Path in a type A: path in $x =_A y$, for $x, y : A$
- More explicitly: $p2q : p =_{x=_A y} q$, for $p, q : x =_A y$
- What about the groupoid structure?
- $\_^{-1} \equiv ind_{Id_A}\, C\, refl\, x\, y : (x =_A y \to y =_A x)$, with $C \equiv \lambda x, y{:}A.\, \lambda p{:}(x{=}_A y).\, (y =_A x)$, satisfies $refl_a{}^{-1} =_\iota refl_a$
- Concatenation operator $\_ \cdot \_ : (x = y) \to (y = z) \to (x = z)$
- LEM: for all $A : \mathcal{U},\ x, y, z, w : A,\ p : x{=}y,\ q : y{=}z,\ r : z{=}w$
  1. $p = refl_x \cdot p = p \cdot refl_y$
  2. $p \cdot p^{-1} = refl_x,\ p^{-1} \cdot p = refl_y$
  3. $(p^{-1})^{-1} = p$
  4. $p \cdot (q \cdot r) = (p \cdot q) \cdot r$
- Proofs on blackboard

Loop spaces

- Loop space: $\Omega(A, a) \equiv (a =_A a)$ (with $\mathit{refl}_a : a =_A a$)
- NOT provable: $\Pi p{:}(a =_A a).\, p =_{(a=_A a)} \mathit{refl}_a$
- Group: $\Omega(A, a)$, refl_a, $\_ \cdot \_$, $\_^{-1}$ (modulo $=_{\Omega(A,a)}$)
- This group is not necessarily commutative
- The loop space of the loop space:

$$\Omega^2(A, a) \equiv (\mathit{refl}_a =_{(a=_A a)} \mathit{refl}_a)$$

- THM 2.1.6 (Eckmann-Hilton): $\Omega^2(A, a)$ is commutative
- Book: picture good, proof improved in current version (09/13)
- Fair attempt on the blackboard: by *based* path induction

$$\Pi a, b : A,\ p, q : (a = b),\ \alpha{:}(p = q).\, \Pi c{:}A.\, \Pi r{:}(b = c).\, p \cdot r = q \cdot r$$

- ... and a lot more (proof assistant dearly missed)

Q to the topologists

If we have full freedom of definition, then we can define the following predicate on the path space of some topological space X:

$$Cxyp \equiv (x = y \wedge p = \mathit{refl}_x)$$

By path induction: all continuous $p : [0,1] \to X$ are constant.
Restrict path induction to continuous C, that is, C boolean valued and continuous wrt the discrete topology on the booleans.
Q: what is the simplest (or: a simple) topological space X validating path induction, but not all paths constant? (A: $[0,1]$)

Pointed types and loop spaces

- $\mathcal{U}_\bullet \equiv \Sigma A{:}\mathcal{U}.\, A$
- Pointed type: $(A, a) \in \mathcal{U}_\bullet$ for $A \in \mathcal{U}$ and $a \in A$
- Pointed loop space: $\Omega(A, a) \equiv ((a =_A a), \mathit{refl}_a)$
- Iterated: $\Omega^0(A, a) \equiv (A, a)$,

$$\Omega^{n+1}(A, a) \equiv \Omega^n(\Omega(A, a))$$

- $\Omega^2(A, a) \equiv \Omega((a =_A a), \mathit{refl}_a) \equiv (\mathit{refl}_a =_{(a =_A a)} \mathit{refl}_a, \mathit{refl}_{\mathit{refl}_a})$

Functions as functors

- Type A as a category:
  - Objects $a : A$
  - Arrows $p : a =_A b$ for $a, b : A$
- Function $f : A \to B$ as a functor (in TOP: f continuous)
  - LEM: For all $x, y : A$ there is $ap_f : (x =_A y) \to (fx =_B fy)$
  - Proof: easy path induction $(ap_f\ refl_x =_\iota refl_{fx})$
- Shorthand: $f(p) \equiv (ap_f\ p)$ (application, action on paths)
- LEM: for all $f : A \to B,\ g : B \to C,\ p : x =_A y,\ q : y =_A z$
  1. $f(p \cdot q) =_{fx =_B fz} f(p) \cdot f(q)$
  2. $f(p^{-1}) =_{fy =_A fx} f(p)^{-1}$
  3. $g(f\ p) =_{g(fx) =_C g(fy)} (g \circ f)(p)$
  4. $id_A x =_\beta x,\ id_A(p) =_{x =_A y} p$

Transport

- Functor $f : A \to B$ maps paths in A to paths in B
- For $B : A \to \mathcal{U}$ and $f : \Pi x{:}A.\, Bx$ this is not so easy ...
- ... because Bx and By are different types
- Type family $B : A \to \mathcal{U}$ is a non-dependent function (of types)
- LEM: for all $x, y : A$ and $p : x =_A y$ there is $p_* : Bx \to By$
- Proof: easy path induction ($(\mathit{refl}_x)_* =_\iota \mathit{id}_{Bx}$)
- Longhand: $\mathit{transport}^B\, p \equiv p_*$, so $\mathit{transport}^B\, p : Bx \to By$
- We can now lift paths in A to the total space $\Sigma\, A\, B$ (picture)
- COR: for all $x, y : A, p : x =_A y,\ u : Bx$ there is

$$\mathit{lift}(u, p) : (x, u) = (y, p_* u)$$

- Type family B: fibration with base A
- Q: actually, the fibration is $\mathit{fst} : (\Sigma\, A\, B) \to A$

Heavy transport

- Picture of transport with dependent function $f : \Pi x{:}A.\, Bx$
- LEM: for all $x, y : A$ and $p : x =_A y$ there is

$$apd_f : (x =_A y) \to (p_*(fx) =_{By} fy) \;\text{ with } apd_f\, refl_x =_\iota refl_{fx}$$

- LEM: if $P : A \to \mathcal{U}$ with $Px = B$ fixed, then for all $x, y : A$, $p : x =_A y$ and $b : B$ there is $tpc^B_p\, b : transport^P\, p\, b =_B b$
- LEM: for $f : A \to B$ and $p : x =_A y$ we have

$$apd_f(p) = (tpc^B_p(fx)) \cdot (ap_f\, p)$$

- LEM: if $P : A \to \mathcal{U}$, $p : x =_A y$, $q : y =_A z$, and $u : Px$, then

$$(q_* \circ p_*)\, u = (p \cdot q)_*\, u$$

- LEM: if $f : A \to B$, $P : B \to \mathcal{U}$, $p : x =_A y$, and $u : P(fx)$,

$$transport^{P \circ f}\, p\, u = transport^P\, f(p)\, u$$

- LEM 2.3.11: book

Homotopies

- DEF: Let $f, g : \Pi x{:}A.\, Px$ for $P : A \to \mathcal{U}$. A *homotopy* from f to g is a dependent function of type $f \sim g$, where

$$(f \sim g) \equiv \Pi x{:}A.\, fx =_{Px} gx$$

- NB: $f \sim g$ is NOT the same as $f =_{\Pi x:A.\, Px} g$
- LEM: homotopy is an equivalence relation:
  - $?r : \Pi f{:}(\Pi x{:}A.\, Px).\, (f \sim f)$
  - $?s : \Pi f, g{:}(\Pi x{:}A.\, Px).\, (f \sim g \to g \sim f)$
  - $?t : \Pi f, g, h{:}(\Pi x{:}A.\, Px).\, (f \sim g \to (g \sim h \to f \sim h))$
- LEM: if $H : f \sim g$ for $f, g : A \to B$, and $p : x =_A y$, then $Hx \cdot g(p) = f(p) \cdot Hy$ (naturality, picture, proof by induction)
- COR: if $H : f \sim id_A$ for $f : A \to A$, and $x : A$, then $H(fx) = f(Hx)$ (picture, proof by cancelling Hx)

Equivalences

- DEF: For $f : A \to B$, a *quasi-inverse* is a triple (g, α, β) with $g : B \to A$ and $\alpha : g \circ f \sim id_A$, $\beta : f \circ g \sim id_B$.
- DEF: the type $qinv(f)$ of quasi-inverses of f is

$$\Sigma g{:}B \to A.\, ((f \circ g \sim id_B) \times (g \circ f \sim id_A))$$

- Examples:
  - $? : qinv(id_A)$ for $id_A : A \to A$
  - $? : qinv(p \cdot \_)$ for $p \cdot \_ : y = z \to x = z$
  - $? : qinv(transport^P_p)$ for $transport^P_p : Px \to Py$
- *qinv* not well-behaved: nonequal inhabitants

Equivalences and Univalence

- DEF: For $f : A \to B$, the type $\mathit{isequiv}(f)$ is

$$(\Sigma g{:}B \to A.\,(f \circ g \sim id_B)) \times (\Sigma h{:}B \to A.\,(h \circ f \sim id_A))$$

- LEM: (i) $\mathit{qinv}(f) \to \mathit{isequiv}(f)$; (ii) $\mathit{isequiv}(f) \to \mathit{qinv}(f)$
- Proof: (i) take $g = h$; (ii) use $g \sim h \circ f \circ g \sim h$
- LEM: for all $e_1, e_2 : \mathit{isequiv}(f)$ we have $e_1 =_{\mathit{isequiv}(f)} e_2$
- Proof: postponed (interaction between $=$ and $\times, \Sigma$)
- DEF: $(A \simeq B) \equiv \Sigma f{:}A \to B.\,\mathit{isequiv}(f)$
- LEM: For all $A, B : \mathcal{U}$ there is $\mathit{idtoeqv} : (A =_{\mathcal{U}} B) \to (A \simeq B)$
- Proof: by induction, using $\mathit{isequiv}(id_A)$
- Univalence Axiom: for all $A, B : \mathcal{U}$, $\mathit{isequiv}(\mathit{idtoeqv})$; hence:

$$(A =_{\mathcal{U}} B) \simeq (A \simeq B)$$

Type equivalence

- An *equivalence* $e : A \simeq B$ is a *pair* (f, p) with $f : A \to B$ and $p : isequiv(f)$; sometimes p is left implicit
- LEM: Type equivalence is an equivalence relation on $\mathcal{U}$:
  - For any $A : \mathcal{U}$, $id_A : A \to A$ is an equivalence
  - For any $f : A \simeq B$ we have an equivalence $f^{-1} : B \simeq A$
  - For any $f : A \simeq B$ and $g : B \simeq C$ we have $g \circ f : A \simeq C$
- Proofs:
  - $id_A : A \to A$ is its own quasi-inverse; hence an equivalence
  - If $f : A \to B$ is an equivalence, it has a quasi-inverse $f^{-1} : B \to A$, which is also an equivalence
  - If $f : A \simeq B$ and $g : B \simeq C$, take their quasi-inverses ...

Structuralism

- Will turn out very different:
  - ‘Two pairs are equal if they are componentwise equal’
  - ‘Two functions are equal if they are pointwise equal’
- Type formers: $\times, +, \Sigma, \Pi, \mathcal{U}, \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbb{N}, Id$
- A lot of structural properties to investigate:
  - equality (example: lemma below)
  - transport
  - action on path
- LEM: $(x =_{A\times B} y) \simeq ((pr_1 x =_A pr_1 y) \times (pr_2 x =_B pr_2 y))$
- Proof on the blackboard

Equality in cartesian products

- LEM: $(x =_{A\times B} y) \simeq ((pr_1 x =_A pr_1 y) \times (pr_2 x =_B pr_2 y))$
- Proof: $\mathit{isequiv}(\lambda p{:}(x =_{A\times B} y).\,(pr_1(p), pr_2(p)))$, by:
  1. define the function in the 'other' direction (notation: $\mathit{pair}^{=}$)
  2. prove that $\mathit{pair}^{=}$ is a *quasi*-inverse
- $\mathit{pair}^{=}$: introduction rule for $x =_{A\times B} y$, elimination:
  1. $ap_{pr_1} : (x =_{A\times B} y) \to (pr_1 x =_A pr_1 y)$
  2. $ap_{pr_2} : (x =_{A\times B} y) \to (pr_2 x =_B pr_2 y)$
- yielding *propositional* computation rules:
  1. $(ap_{pr_1}(\mathit{pair}^{=}(p, q))) = p \quad$ for $p : (pr_1 x =_A pr_1 y)$
  2. $(ap_{pr_2}(\mathit{pair}^{=}(p, q))) = q \quad$ for $q : (pr_2 x =_B pr_2 y)$
- and a *propositional* uniqueness principle:
  $r = \mathit{pair}^{=}(ap_{pr_1} r, ap_{pr_2} r) \quad$ for $r : (x =_{A\times B} y)$
- plus a lot of other componentwise *propositional* equalities

Transport and action in cartesian products

- THM: If $A, B : Z \to \mathcal{U}$, $p : z =_Z w$ and $x : ((Az) \times (Bz))$, then $p_* x =_{(Aw)\times(Bw)} (p_*(pr_1 x), p_*(pr_2 x))$
- Proof: path induction plus propositional uniqueness
- Functoriality of *ap* under cartesian products: let $g : A \to A'$, $h : B \to B'$ and define $f : (A \times B) \to (A' \times B')$ by $f \equiv \lambda x{:}A \times A'.\,(g(pr_1 x), h(pr_2 x))$. Then:
- THM: if also $x, y : A \times B$, $p : (pr_1 x) =_A (pr_1 y)$ and $q : (pr_2 x) =_B (pr_2 y)$, we have (picture)

$$f(pair^{=}(p, q)) =_{fx=fy} pair^{=}(g(p), h(q))$$

- Proof: by induction on pairs and paths

Equality and transport in Σ-types

- THM: let $P : A \to \mathcal{U}$ and $w, w' : \Sigma x{:}A.\, Px$. Then:

$$(w =_{\Sigma x:A.\, Px} w') \quad \simeq \quad \sum_{p:pr_1 w = pr_1 w'} (p_*(pr_2 w) = pr_2 w')$$

- THM: let $P : A \to \mathcal{U}$ and $Q : (\Sigma x{:}A.\, Px) \to \mathcal{U}$. Then $\lambda x{:}A.\, (\Sigma u{:}Px.\, Q(x, u))$ is a type family such that for $p : x = y$ and $(u, z) : (\Sigma u{:}Px.\, Q(x, u))$ we have:

$$p_*(u, z) \quad =_{\Sigma u:Py.\, Q(y,u)} \quad (p_* u, \mathit{lift}(u, p)_* z)$$

- Generalizes: $p_* x \quad =_{(Aw)\times(Bw)} \quad (p_*(pr_1 x), p_*(pr_2 x))$
- Time for a picture!

The unit type

- THM: for all $x, y : \mathbf{1}$ we have $(x = y) \simeq \mathbf{1}$.
- Proof: exercise
- Pitfall: don't start proving $(\star = \star) \simeq \mathbf{1}$

Equality in Π-types

- Wanted, for $A : \mathcal{U},\ B : A \to \mathcal{U}$ and $f, g : \Pi x{:}A.\, Bx$:

$$(f = g) \simeq (\Pi x{:}A.\, fx =_{Bx} gx)$$

- By an easy path induction (to be viewed as elimination):

$$happly : (f = g) \to (\Pi x{:}A.\, fx =_{Bx} gx)$$

- Axiom (function extensionality): $isequiv(happly)$
- Quasi-inverse of *happly* (to be viewed as introduction):

$$funext : (\Pi x{:}A.\, fx =_{Bx} gx) \to (f = g)$$

- Propositional equalities (use functional extensionality):
  - $happly\,(funext\ h) = h$
  - $\alpha = funext\,(happly\,\alpha)$
  - $refl_f = funext\,(\lambda x{:}A.\, refl_{fx})$
  - $\alpha^{-1} = funext\,(\lambda x{:}A.\, (happly\,\alpha\,x)^{-1})$
  - $\alpha \cdot \beta = funext\,(\lambda x{:}A.\, (happly\,\alpha\,x) \cdot (happly\,\beta\,x))$

Transport in Π-types

- Let $A, B : X \to \mathcal{U}$ and define $A2B \equiv \lambda x{:}X.\,(Ax \to Bx)$. Given a path $p : x_1 =_X x_2$, there are two natural ways to transport $f : Ax_1 \to Bx_1$ to $Ax_2 \to Bx_2$ (picture):
    1. by applying $\mathit{transport}^{A2B}\, p : (Ax_1 \to Bx_1) \to (Ax_2 \to Bx_2)$
    2. by transporting any given $a : Ax_2$ first back to Ax_1, applying f, and then transporting the result in Bx_1 to Bx_2

  These two ways turn out to be propositionally equal.
- LEM: under conditions as above:

$$\mathit{transport}^{A2B}\, p\, f = \lambda a{:}Ax_2.\, \mathit{transport}^{B}\, p\, (f\, (\mathit{transport}^{A}\, p^{-1}\, a))$$

- Proof: by path induction
- This was only the non-dependent case ... (see the book)

Univalence

- $idtoeqv : (A =_{\mathcal{U}} B) \to (A \simeq B)$ defined by path induction
- Univalence Axiom: for all $A, B : \mathcal{U}$, $isequiv(idtoeqv)$; hence:

$$(A =_{\mathcal{U}} B) \simeq (A \simeq B)$$

- Abuse of notation: $(f, p) : A \simeq B$ identified with $f : A \to B$
- A different view on univalence:
  - Introduction (postulated): $ua : (A \simeq B) \to (A =_{\mathcal{U}} B)$
  - Elimination (transport): $[pr_1 \circ] idtoeqv : (A =_{\mathcal{U}} B) \to (A \to B)$
  - Propositional computation rule: $idtoequiv\,(ua\, f) = f$
  - Propositional uniqueness: $p = ua\,(istoeqv\, p)$, so $refl_A = ua\, id_A$
- LEM: $(ua\, f) \cdot (ua\, g) = ua\,(g \circ f)$; $(ua\, f)^{-1} = ua\,(f^{-1})$
- LEM: for $B : A \to \mathcal{U}$, $p : x =_A y$ we have (no UA!):

$$p_* \equiv transport^B\, p =_{Bx \to By} idtoequiv\,(ap_B\, p)$$

Identity types

- THM: if $f : A \to B$ is an equivalence, then for all $a, a' : A$ we have the equivalence $ap_f : (a =_A a') \to (fa =_B fa')$
- Transport in families of identity types, with $p : x_1 =_A x_2$.
  LEM: for $p : x_1 =_A x_2$ and $q : Px_1$ for superscript $P : A \to \mathcal{U}$
  1. $transport^{\lambda x:A.\,(a =_A x)}\, p\, q = q \cdot p$
  2. $transport^{\lambda x:A.\,(x =_A a)}\, p\, q = p^{-1} \cdot q$
  3. $transport^{\lambda x:A.\,(x =_A x)}\, p\, q = p^{-1} \cdot q \cdot p$
  4. $transport^{\lambda x:A.\,(fx =_B gx)}\, p\, q = (ap_f\, p)^{-1} \cdot q \cdot (ap_g\, p)$
     for $f, g : A \to B$
  5. $transport^{\lambda x:A.\,(fx =_B gx)}\, p\, q = (apd_f\, p)^{-1} \cdot p_*(q) \cdot (apd_g\, p)$
     for $f, g : \Pi x{:}A.\, Bx$

  Proofs by pictures
- THM: for $p : a =_A a'$, $q : a =_A a$, and $r : a' =_A a'$ we have:

$$((transport^{\lambda x:A.\,(x =_A x)}\, p\, q) = r) \simeq (q \cdot p = p \cdot r)$$

Coproducts

- Coproducts are interesting: try defining $f : A + B \to A$ ...
- Hopefully (proof not obvious, too special):
  1. $(\mathit{inl}\ a_1 = \mathit{inl}\ a_2) \simeq (a_1 = a_2)$
  2. $(\mathit{inr}\ b_1 = \mathit{inr}\ b_2) \simeq (b_1 = b_2)$
  3. $(\mathit{inl}\ a = \mathit{inr}\ b) \simeq \quad \mathbf{0}$

  Idea: combine 1,3 (2,3) and generalize! (Q: 1,2,3,4?)
- Fix $a_0 : A$; then $P \equiv \lambda x{:}A{+}B.\,(\mathit{inl}\ a_0 = x) : A{+}B \to \mathcal{U}$.
- Wanted: $P(\mathit{inl}\ a) \simeq (a_0 = a)$ and $P(\mathit{inr}\ b) \simeq \mathbf{0}$
- Define $\mathit{code} : A{+}B \to \mathcal{U}$ recursively by

$$\mathit{code}\,(\mathit{inl}\ a) \equiv (a_0 = a) \qquad \mathit{code}\,(\mathit{inr}\ b) \equiv \mathbf{0}$$

- Define $\mathit{encode} : \Pi x{:}A{+}B.\,\Pi p{:}(\mathit{inl}\ a_0 = x).\,(\mathit{code}\,x)$
  and $\mathit{decode} : \Pi x{:}A{+}B.\,\Pi c{:}(\mathit{code}\,x).\,(\mathit{inl}\ a_0 = x)$
- Prove that $\mathit{encode}(x, \_)$ and $\mathit{decode}(x, \_)$ are quasi-inverses

Coproducts (ctnd)

- THM: for all $x : A + B$ we have $((\mathit{inl}\, a_0) = x) \simeq (\mathit{code}\, x)$
- Details on the blackboard
- COR (of the proof):
  - $\mathit{encode}(\mathit{inl}\, a) : ((\mathit{inl}\, a_0) = (\mathit{inl}\, a)) \to (a_0 = a)$
  - $\mathit{encode}(\mathit{inr}\, b) : ((\mathit{inl}\, a_0) = (\mathit{inr}\, b)) \to \mathbf{0}$
- Transport: for $A, B : X \to \mathcal{U}$ and $p : x_1 =_X x_2$:
  - $\mathit{transport}^{\lambda x:X.\,(Ax+Bx)}\, p\, (\mathit{inl}\, a) = \mathit{inl}\, (\mathit{transport}^{A}\, p\, a)$
  - $\mathit{transport}^{\lambda x:X.\,(Ax+Bx)}\, p\, (\mathit{inr}\, b) = \mathit{inr}\, (\mathit{transport}^{B}\, p\, b)$

Natural numbers

- We define $code : \mathbb{N} \to \mathbb{N} \to \mathcal{U}$ such that:
  THM: for all $m, n : \mathbb{N}$ we have $(m = n) \simeq (code\, m\, n)$
- Details on the blackboard (or in book)
- COR: we have (inhabited)
  - $\Pi m{:}\mathbb{N}.\, ((Sm) = 0) \to \mathbf{0}$, as $code((Sm), 0) \equiv \mathbf{0}$
  - $\Pi n, m{:}\mathbb{N}.\, ((Sm) = (Sn)) \to (m = n)$, as $code(Sm, Sn) \equiv code(m, n)$
- COR: $\mathbb{N}$ is a *set* (type in which paths are unique)

Transporting structure

- $SGS(A) \equiv \Sigma m{:}A{\to}A{\to}A.\, \Pi x, y, z{:}A.\, (mx(myz) = m(mxy)z)$
- $SG \equiv \Sigma A{:}\mathcal{U}.\, SGS(A)$
- If $e : A =_{\mathcal{U}} B$, then $transport^{SGS}\, ua(e) : SGS(A) \to SGS(B)$
- For $(m, a) : SGS(A)$, $transport^{SGS}\, ua(e)(m, a) \equiv (m', a')$ with
  - $m' = ua(e)_* m \equiv transport^{\lambda X.\, X \to X \to X}\, ua(e) m$
  - $a' = transport^{\lambda(X,m).\, Assoc(X,m)}(pair^{=}(ua(e), refl_{m'}))\, a$

  where $Assoc(X, m) \equiv \Pi x, y, z{:}X.\, (mx(myz) = m(mxy)z)$
- NB: $pair^{=}(ua(e), refl_{m'}) : (A, m) = (B, m')$ (Thm. 2.7.2/4)
- Indeed $(m', a') : SGS(B)$ (no need to reprove)
  - $m' : B \to B \to B$
  - $a' : Assoc(B, m') \equiv \Pi x, y, z{:}B.\, (m'x(m'yz) = m'(m'xy)z)$

Some calculations

- If $p : A =_{\mathcal{U}} B$, then $transport^{\lambda X:\mathcal{U}.\,X}\,p : A \to B$
- Also: $[pr_1\circ]idtoeqv : (A =_{\mathcal{U}} B) \to (A \to B)$
- Verbatim the same: $transport^{\lambda X:\mathcal{U}.\,X} \equiv idtoeqv$
- Hence: $transport^{\lambda X:\mathcal{U}.\,X}$ and ua are each other's quasi-inverse
- So: $transport^{\lambda X:\mathcal{U}.\,X}\,ua(e)^{-1} = e^{-1}$
- Recall back-and-forth technique for $transport^{\lambda X.\,AX \to BX}$
- Then: $m'b_1b_2 \equiv (transport^{\lambda X.\,X\to X\to X}\,ua(e)m)b_1b_2 = tspt^{\lambda X.\,X}\,ua(e)(m\,(tspt^{\lambda X.\,X}\,ua(e)^{-1}b_1)(tspt^{\lambda X.\,X}\,ua(e)^{-1}b_2))$ $= e(m\,(e^{-1}b_1)\,(e^{-1}b_2))$ (recall $e : A \to B$ equivalence)
- Algebraic proof of $Assoc(B, m')$ not needed (equal to a')

Equality of semigroups

- By Thm. 2.7.2: the type $(A, m, a) =_{SG} (B, m', a')$ is equal to type of pairs

$$\begin{array}{lll} p_1 & : & A =_{\mathcal{U}} B \\ p_2 & : & transport^{SGS} p_1 \, (m, a) = (m' a') \end{array}$$

  where by univalence $p_1 = ua(e)$ for some equivalence e and $p_2 = (p_3, p_4)$ is a pair of proofs with p_3 of type

$$\Pi y_1, y_2{:}B. \, (e \, (m \, (e^{-1} y_1) \, (e^{-1} y_2)) = m' \, y_1 \, y_2$$

  which is equivalent to

$$\Pi x_1, x_2{:}A. \, (e \, (m \, x_1 \, x_2) = m' \, (e \, x_1) \, (e \, x_2)$$

- ... recovering the notion of *semigroup isomorphism*

Universal properties

- LEM: $\lambda f.\,(pr_1 \circ f, pr_2 \circ f)$ is an equivalence

$$(X \to (A \times B)) \to ((X \to A) \times (X \to B))$$

- ... and also for type families (see book)
- EXC: define an equivalence

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

- For $A : \mathcal{U},\ B : \mathcal{U}, R : A \to B \to \mathcal{U}$, ac is an equivalence

$$\mathsf{ac} : (\Pi x{:}A.\, \Sigma y{:}B.\, R\, x\, y) \to \Sigma f{:}A{\to}B.\, \Pi x{:}A.\, R\, x\, (fx)$$

- Cartesian closure: $((A \times B) \to C) \simeq (A \to (B \to C))$
- ... and also for type families (see book)

Sets

- A set is a type in which paths are unique:

$$isSet(A) \equiv \Pi x, y{:}A.\, \Pi p, q{:}(x =_A y).\, p =_{x=_A y} q$$

- Examples: $\mathbf{0}, \mathbf{1}, \mathbb{N}$
- Proofs: trivial, $\Pi x, y{:}\mathbf{1}.\,(x =_{\mathbf{1}} y) \simeq \mathbf{1}$ (picture), and $\Pi x, y{:}\mathbb{N}.\,(x =_{\mathbb{N}} y) \simeq code(x, y)$
- Most type forming operations preserve sets:
  - if A and B are sets, then so are $A \times B$ and $A + B$
  - if A is a set and $B : A \to \mathcal{U}$ such that Bx is a set for every $x : A$, then $\Sigma\, A\, B$ is a set (by 'structuralism')
  - if A is any type and $B : A \to \mathcal{U}$ such that Bx is a set for every $x : A$, then $\Pi\, A\, B$ is a set (using function extensionality twice!)
- Proof of last: if $f, g : \Pi\, A\, B$, $p, q : f = g$, then by fun.ext. $p = funext(happly\, p)$ and $q = funext(happly\, q)$. By assumption on Bx, $happly\, p\, x = happly\, q\, x$ for all $x : A$. Hence, again by fun.ext. $happly\, p = happly\, q$, so $p = q$.

Sets (ctnd)

- The universe is not a set: $isSet(\mathcal{U}) \to \mathbf{0}$
- Proof: we construct by univalence $p : \mathbf{2} = \mathbf{2}$ with $(p =_{\mathbf{2}=\mathbf{2}} refl_{\mathbf{2}}) \to \mathbf{0}$. Define the equivalence $e : \mathbf{2} \to \mathbf{2}$ by $e(0) = 1$, $e(1) = 0$ (e is its own quasi-inverse). If $ua(e) =_{\mathbf{2}=\mathbf{2}} refl_{\mathbf{2}}$, then $\mathbf{0}$ gets inhabited by $e = idtoeqv(ua(e)) = idtoeqv\ refl_{\mathbf{2}} = id_{\mathbf{2}}$,
- Definition of h-levels (later also levels $-2, -1$):
  - $0type(A) \equiv isSet(A) \equiv \Pi x, y{:}A.\, \Pi p, q{:}(x =_A y).\, p =_{x=_A y} q$
  - $1type(A) \equiv \Pi x, y{:}A.\, isSet(x =_A y) \equiv \dots$
- LEM: inhabited $isSet(A) \to 1type(A)$
- Proof on blackboard (uses Lemmas 2.3.4 and 2.11.2)

Types vs. propositions

- THM: UA conflicts with for all $A:\mathcal{U}$, $(\neg\neg A) \to A$
- More precisely:
  - without UA, $\Pi A{:}\mathcal{U}.\,((\neg\neg A) \to A)$ consistent
  - with UA, $\neg\Pi A{:}\mathcal{U}.\,((\neg\neg A) \to A)$ is inhabited
- Intuition: under UA, there cannot be a *natural* choice operator selecting an element from every non-empty type
- Proof: assume $f : \Pi A{:}\mathcal{U}.\,(((A \to \mathbf{0}) \to \mathbf{0}) \to A)$. We construct an inhabitant of $\mathbf{0}$. Take $e : \mathbf{2} \simeq \mathbf{2}$ as above. Use that f acts on $ua(e)$ by

  $$apd_f\, ua(e) : (transport^{\lambda A.\,(\neg\neg A)\to A}\, ua(e)\,(f\,\mathbf{2})) = f\,\mathbf{2}$$

  Rest on blackboard (use back-and-forth and $(\neg\neg\mathbf{2}) \simeq \mathbf{1}$)
- COR: UA conflicts with for all $A:\mathcal{U}$, $A + (\neg A)$
- Conclusion: we cannot use all types as propositions

Mere propositions

- Wanted: $\mathcal{V}$, UA consistent with $\Pi A{:}\mathcal{V}.\,((\neg\neg A) \to A)$
- Examples: $\mathbf{0}, \mathbf{1} : \mathcal{V}$, but not $\mathbf{2} : \mathcal{V}$ (UA: $\simeq$-naturality)
- Mere proposition: $\mathit{isProp}(P) \equiv \Pi x, y{:}P.\,(x =_P y)$
- Level 0: $\mathit{isSet}(A) \simeq \Pi x, y{:}A.\,\mathit{isProp}(x =_A y)$
- LEM: inhabited $\mathit{isProp}(P) \to P \to (P \simeq \mathbf{1})$
- LEM: *isProp* is closed under $\times$ (UA not needed)
- LEM: $\mathit{isProp}(P)$ and $P \simeq Q$, then $\mathit{isProp}(Q)$ (UA not needed)
- LEM: with funext, if $A : \mathcal{U}$ and $B : A \to \mathcal{U}$ such that $\mathit{isProp}(Bx)$ for every $x : A$, then $\mathit{isProp}(\Pi x{:}A.\,Bx)$.
- COR: $P \to Q$ is a mere proposition whenever Q is
- NB: *isProp* is not closed under $+$, nor Σ

More on Mere propositions

- LEM: if P, Q are mere propositions with $P \to Q,\ Q \to P$, then $P \simeq Q$.
- LEM: Every mere proposition is a set (cf. Lemma 3.1.8)
- LEM: for every type A, $isProp(A)$ and $isSet(A)$ are mere propositions
- Proof: use funext. If $f, g : isProp(A)$, then $fxy, gxy : x =_A y$, hence $fxy = gxy$ since A is a set. Analogously for $isSet(A)$ (use Lemma 3.1.8).
- The HoTT laws of excluded middle and double negation:
  - $LEM_{-1} \equiv \Pi A{:}\mathcal{U}.\, isProp(A) \to (A + \neg A)$
  - $DNL_{-1} \equiv \Pi A{:}\mathcal{U}.\, isProp(A) \to ((\neg\neg A) \to A)$

  Both are equivalent, independent, consistent with UA

Decidability, subtypes and subsets

- Under LEM_{-1}, no need for $+$, nor Σ, for doing logic
- For $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, localized forms of LEM_{-1}:
  - A is *decidable* if $A + \neg A$
  - B is *decidable* if $\Pi x{:}A.\,(Bx + \neg Bx)$
  - A has *decidable equality* if $\Pi x, y{:}A.\,((x =_A y) + \neg(x =_A y))$
- Example: LEM_{-1} implies that sets have decidable equality
- For $A : \mathcal{U}$ and $P : A \to \mathcal{U}$ such that $isProp(Px)$ for all $x : A$, if $(x, p), (x, q) : \Sigma x{:}A.\,Px$, then $p = q$, and we write:

$$\{x{:}A \mid Px\} \equiv \Sigma x{:}A.\,Px$$

- EXC 3.3: $\Sigma x{:}A.\,Px$ is a set if A is a set and $P : A \to \mathcal{U}$ such that $isSet(Px)$ for all $x : A$

Propositional truncation

- Propositional truncation (or 'squash') hides all info about inhabitants beyond their mere existence.
- NEW: this is a *higher* inductive type (Chapter 6)!
- Formation: $\|A\| : \mathcal{U}$ if $A : \mathcal{U}$
- Introduction, both for objects and paths:
  - $|a| : \|A\|$ if $a : A$
  - $x =_{\|A\|} y$ if $x, y : \|A\|$
- Elimination: defining $f : \|A\| \to B$ means
  - specifing $f|a| : B$ for all $a : A$
  - making sure $f|a| =_{\|A\|} f|b|$ for all $a, b : A$
- Only 'constant' functions, or better: if $isProp(B)$, any $g : A \to B$ defines $f : \|A\| \to B$ with $f|a| = ga$
- EXC: $isProp(\|P\|)$, $isProp(P) \simeq (P \simeq \|P\|)$, for all $P : U$

Traditional logic, unique choice

- Under UA: like propositions as types, but with mere propositions

$P \iff Q$	$P \vee Q$	$\exists(x : A).Px$
$P =_{\mathcal{U}} Q$	$\|P + Q\|$	$\|\Sigma x{:}A.\, Px\|$

- LEM_{-1}, decidability: mathematical axioms
- LEM (unique choice): if $P : A \to \mathcal{U}$ such that
  1. Px is a mere proposition for all $x : A$
  2. for each $x : A$ we have $\|Px\|$ (so, $\Pi x{:}A.\, \|Px\|$ inhabited)

  Then $\Pi x{:}A.\, Px$ (proof: $isProp(Px) \to \|Px\| \to Px$)
- Choice can sometimes be refined to unique choice
- Homework: read 3.9 and 3.10

The Axiom of Choice (AC)

- Let $isSet(X)$, and $A : X \to \mathcal{U},\ P : \Pi x{:}X.\,(Ax \to \mathcal{U})$ such that
  1. Ax is a set for all $x : X$
  2. Pxa is a mere proposition for all $x : X,\ a : Ax$

  Then AC asserts

$$(\Pi x{:}X.\,\|\Sigma a{:}Ax.\,Pxa\|) \to \|\Sigma f{:}(\Pi x{:}X.\,Ax).\,\Pi x{:}X.\,Px(fx)\|$$

- LEM: AC is equivalent to, with $Y : X \to \mathcal{U}$ such that Yx sets,

$$(\Pi x{:}X.\,\|Yx\|) \to \|\Pi x{:}X.\,Yx\|$$

- Proof: use that ac is an equivalence (2.15.7) and that Yx is equally expressive as $\Sigma a{:}Ax.\,Pxa$ (subset!)
- Discussion

Contractible types

- Contractible type: inhabited mere proposition
- DEF: $\mathit{isContr}(A) \equiv \Sigma a{:}A.\, \Pi x{:}A.\, a = x$
- LEM: logical equivalences (Q: why not $\simeq$?)
  $\mathit{isContr}(A) \iff (A \times \mathit{isProp}(A)) \iff (A \simeq \mathbf{1})$
- LEM: $\mathit{isProp}(\mathit{isContr}(A))$, for all A
- Proof: first para in book + $\mathit{isProp}(\Pi x{:}A.\, a' = x)$
- $\mathit{isContr}(A) \to \mathit{isContr}(\mathit{isContr}(A))$ + other closure properties
- $\mathit{isContr}$ not closed under $+$

Retraction

- Retraction: $\pm$ half of an equivalence (Q: OK?)
- DEF: $r : A \to B$ *retraction* if there is $s : B \to A$ (the *section* of r) such that $r \circ s \sim id_B$. Then we call B a *retract* of A.
- LEM: if B a retract of A, then $isContr(A) \to isContr(B)$
- LEM: for all $A : \mathcal{U}$ and $a : A$, $isContr(\Sigma x{:}A.\, a = x)$
- LEM: let $P : A \to \mathcal{U}$ be a type family
  1. if each Px is contractible, then $A \simeq \Sigma x{:}A.\, Px$
  2. if A is contractible with center $a : A$, then $Pa \simeq \Sigma x{:}A.\, Px$
- LEM: $isProp(A) \simeq \Pi x, y{:}A.\, isContr(x =_A y)$

Equivalences

- Wanted: $XYZequiv(f) \leftrightarrow qinv(f)$ and $isProp(XYZequiv(f))$
- Q: desirable or *really needed* that $isProp(XYZequiv(f))$?
- LEM: $qinv(f) \to (qinv(f) \simeq \Pi x{:}A.\, x =_A x))$ for all $f : A \to B$
- Book: for some $A : \mathcal{U}$, $\Pi x{:}A.\, x =_A x$ not contractible
- Some information is missing from $qinv(f)$ ...
- Three alternative (equivalent) definitions:
  1. $ishae(f)$, adds coherence info to $qinv(f)$
  2. $biinv(f)$ ($\equiv isequiv(f)$, splits quasi-inverse in two
  3. $isContr(f)$, imposes contractibility of fibers

Half Adjoint Equivalences

- DEF: for $f : A \to B$, the type $ishae(f)$ is

$$\Sigma g{:}B \to A.\, \Sigma \alpha{:}(g \circ f \sim id_A).\, \Sigma \beta{:}(f \circ g \sim id_B).\, \Pi x{:}A.\, f(\alpha x) = \beta(fx)$$

- Diff with $qinv(f)$: **1** vs. last Π-type
- Logically equivalent: last Π-type $\Pi x{:}A.\, g(\beta x) = \alpha(gx)$
- LEM: for any $f : A \to B$, $qinv(f) \to ishae(f)$
- Proof: take 'the' g and α from $qinv(f)$. Define

$$\beta' b \equiv \beta(f(gb))^{-1} \cdot f(\alpha(gb)) \cdot (\beta b)$$

  Find $(\tau a) : (f(\alpha a) = \beta'(fa))$ (see book)
- DEF: The *fiber* of $f : A \to B$ over $b : B$ is

$$fib_f(b) \equiv \Sigma x{:}A.\, (fx = b)$$

- LEM: if $ishae(f)$, then any $fib_f(b)$ is contractible

Bi-invertible maps

- DEF: for $f : A \to B$, define:
  1. $linv(f) \equiv \Sigma g{:}B \to A.\,(g \circ f \sim id_A)$
  2. $rinv(f) \equiv \Sigma g{:}B \to A.\,(f \circ g \sim id_B)$
  3. $biinv(f) \equiv (linv(f) \times rinv(f))$ (that is, $isequiv(f)$)
- LEM: if $qinv(f)$, then $linv(f)$ and $rinv(f)$ are contractible
- Proof: $\Sigma g{:}B \to A.\,(g \circ f = id_A)$ is a fiber and $qinv(\_ \circ f)$
- DEF: for $f : A \to B$, $(g, \alpha) : linv(f)$, $(g, \beta) : rinv(f)$ define:
  1. $lcoh(f, g, \alpha) \equiv \Sigma\beta{:}(f \circ g \sim id_B).\,\Pi y{:}B.\,(g(\beta y) = \alpha(gy))$
  2. $rcoh(f, g, \beta) \equiv \Sigma\alpha{:}(g \circ f \sim id_A).\,\Pi x{:}A.\,(f(\alpha x) = \beta(fx))$
- Intuition: $lcoh(f, g, \alpha)$ expresses that 'g is also right inverse, plus coherence'

A Mere Proposition

- LEM: for all $f : A \to B$, $(g, \alpha) : linv(f)$, $(g, \beta) : rinv(f)$
  1. $lcoh(f, g, \alpha) \equiv \Pi y{:}B.\, (f(gy), \alpha(gy)) =_{fib_g(gy)} (y, refl_{gy}))$
  2. $rcoh(f, g, \beta) \equiv \Pi x{:}A.\, (g(fx), \beta(fx)) =_{fib_f(fx)} (x, refl_{fx}))$
- LEM: if $ishae(f)$, then $lcoh(f, g, \alpha)$ and $rcoh(f, g, \beta)$ are contractible for any $(g, \alpha) : linv(f)$, $(g, \beta) : rinv(f)$
- LEM: $ishae(f)$ is a mere proposition, for any $f : A \to B$
- Proof: $\Sigma(g, \beta){:}rinv(f).\, rcoh(f, g, \beta)$ .... is contractible
- LEM: $biinv(f)$ is a mere proposition for any $f : A \to B$, and $biinv(f) \simeq ishae(f)$

Contractible fibers

- DEF: for $f : A \to B$, we define:

$$isContr(f) \equiv \Pi y{:}B.\, isContr(fib_f(y))$$

- LEM: $isContr(f) \to ishae(f)$, for any $f : A \to B$
- Proof: blackboard, or *latest pdf* of book
- REM: converse has been shown already
- LEM: $isContr(f)$ is a mere proposition for any $f : A \to B$, and $isContr(f) \simeq ishae(f)$
- LEM: if $f : A \to B$ such that $B \to isequiv(f)$, then $isequiv(f)$
- THM (summing up): $biinv(f) \simeq ishae(f) \simeq isContr(f)$

Bijections, surjections and embeddings

- DEF: for *sets* $A, B : \mathcal{U}$, we call an equivalence a *bijection*
- DEF: for *types* $A, B : \mathcal{U}$, $f : A \to B$, we define:
  1. f is a *surjection* if for all $b : B$ we have $\|fib_f(b)\|$ (inhabited)
  2. f is a *split surjection* if $\Pi b{:}B.\, \Sigma a{:}A.\, (f(a) = b)$
  3. f is an *embedding* if for all $x, y : A$ we have $isequiv(ap_f)$
  4. f is an *injection* if f an embedding and A, B sets
- REM: last clause iff $\Pi x, y{:}A.\, (fx =_B fy) \to (x =_A y)$
- THM: $isequiv(f)$ iff $(isEmbedding(f)$ and $isSurjection(f))$
- COR: $isequiv(f) \simeq (isEmbedding(f) \times isSurjection(f))$

Fiberwise equivalences

- DEF: for $P, Q : A \to \mathcal{U}$, we call $f : \Pi x{:}A.\,(Px \to Qx)$ a *fiberwise equivalence* if $fx : (Px \simeq Qx)$ for all $x : A$
- DEF: for $P, Q : A \to \mathcal{U}$, $f : \Pi x{:}A.\,(Px \to Qx)$, we define:

$$\mathit{total}(f) \equiv \lambda w.\,(\mathit{pr}_1 w, f(\mathit{pr}_1 w, \mathit{pr}_2 w)) : (\Sigma x{:}A.\, Px) \to (\Sigma x{:}A.\, Qx)$$

- THM: for $f : \Pi x{:}A.\,(Px \to Qx)$, $x : A$ and $v : Qx$

$$\mathit{fib}_{\mathit{total}(f)}(x, v) \simeq \mathit{fib}_{fx}(v)$$

- THM: f is a fiberwise equivalence iff $\mathit{total}(f)$ is an equivalence

Univalence implies weak extensionality

- DEF: the *weak extensionality principle* is: for all $P : A \to \mathcal{U}$

$$(\Pi x{:}A.\, isContr(Px)) \to isContr(\Pi x{:}A.\, Px)$$

- Intuition: if co-domain singleton, there is only one function
- LEM: if $pr_1 : (\Sigma x{:}A.\, Px) \to A$ and $a : A$, then $fib_{pr_1}(a) \simeq Pa$
- LEM: if UA and $A, B, X : \mathcal{U}$, $e : A \simeq B$, then $([pr_1 \circ] e \circ \_)$ defines an equivalence $(X \to A) \to (X \to B)$
- THM: if UA and $P : A \to \mathcal{U}$ is a family of contractible types, then $\Pi x{:}A.\, Px$ is (a retract of $fib_\alpha(id_A)$ and so) contractible
- Proof: assume UA and a family of contractible types $P : A \to \mathcal{U}$. Then $pr_1 : (\Sigma x{:}A.\, Px) \to A$ is an equivalence, defining an equivalence $\alpha : (A \to \Sigma x{:}A.\, Px) \to (A \to A)$. $fib_\alpha(id_A) = \Sigma f{:}(A {\to} \Sigma x{:}A.\, Px).\, (pr_1 \circ f = id_A)$, retract ...

Weak extensionality implies extensionality

- Recall: $\mathit{happly}\,f\,g : (f = g) \to (\Pi x{:}A.\, fx =_{Px} gx)$
- Recall: $\mathit{funext}\,f\,g : (\Pi x{:}A.\, fx =_{Px} gx) \to (f = g)$
- To prove (where $\Pi\,A\,P$ abbreviates $\Pi x{:}A.\, Px$):

$$\Pi A{:}U.\, \Pi P{:}(A \to \mathcal{U}).\, \Pi f, g{:}(\Pi\,A\,P).\, \mathit{isequiv}(\mathit{happly}\,f\,g))$$

- Proof: we show that $\mathit{total}(\mathit{happly}\,f)$ is an equivalence

$$(\Sigma g{:}(\Pi\,A\,P).\,(f = g)) \to (\Sigma g{:}(\Pi\,A\,P).\,\Pi x{:}A.\, fx =_{Px} gx))$$

Lhs contractible, it suffices that rhs is contractible too. Rhs is retract of $\Pi x{:}A.\,\Sigma u{:}Px.\,(fx = u)$, which is contractible by weak extensionality. (Retraction uses $=_\eta$, not extensionality.)

Inductive Types

- Inductive type: type of objects that are freely generated by *constructors* (roughly, functions with the inductive type as co-domain), plus an elimination principle (induction)
- Examples:
  1. $\mathbf{0}$ without constructors; $ind_{\mathbf{0}}\, C : \Pi x{:}\mathbf{0}.\ Cx$
  2. $\mathbf{1}$ with constructor $\star : \mathbf{1}$; $ind_{\mathbf{1}}\, C : C(\star) \to \Pi x{:}\mathbf{1}.\ Cx$
  3. $\mathbf{2}$ with constructors $0_{\mathbf{0}}, 1_{\mathbf{1}} : \mathbf{2}$; $ind_{\mathbf{2}}\, C : C(0){\to}C(1){\to}\Pi x{:}\mathbf{2}.\ Cx$
  4. $\mathbb{N}$ with constructors $0 : \mathbb{N}$ and $S : \mathbb{N} \to \mathbb{N}$; usual induction
- Recursion: non-dependent elimination ($C = \lambda x{:}iT.\, A$)

Inductive Types (ctnd)

- More examples:
  1. $A \times B$ with constructor $(\_, \_) : A \to B \to A \times B$; induction $ind_{A\times B}\, C : (\Pi a{:}A.\, \Pi b{:}B.\, C(a,b)) \to \Pi p{:}A{\times}B.\, Cp$
  2. $A + B$ with constructors $inl : A \to A + B, inr : B \to A + B$; $ind_{A+B}\, C : (\Pi a{:}A.\, C(inl\, a)){\to}(\Pi b{:}B.\, C(inr\, b)){\to}\Pi s{:}A{+}B.\, Cs$
  3. $List\, A$ with constructors $nil{:}List\, A$, $cons{:}A{\to}(List\, A){\to}(List\, A)$; $ind_{ListA}\, C{:}C\, nil \to \Pi a{:}A.\, \Pi \ell{:}List\, A.\, C(cons\, a\, \ell){\to}\Pi \ell{:}List\, A.\, C\, \ell$
- Uniqueness principle: under *funext*, induction and recursion yield unique functions in $\Pi x{:}iT.\, Cx$
- Example of uniqueness in case of N on blackboard (recall $ind_N Ce_0 e_S 0 =_\iota e_0,\ ind_N Ce_0 e_S (Sn) =_\iota e_S n(ind_N Ce_0 e_S n))$

Uniqueness of Inductive Types

- Y.a. inductive type: $\mathbb{N}'$ with constructors $0' : \mathbb{N}'$, $S' : \mathbb{N}' \to \mathbb{N}'$
- Looks familiar ..., but this is not $\mathbb{N}$
- Induction very similar, with computation rules
  $\mathit{ind}\, C\, e_0\, e_S\, 0' =_\iota e_0,\ \mathit{ind}\, C\, e_0\, e_S\, (S' n) =_\iota e_S\, n\, (\mathit{ind}\, C\, e_0\, e_S\, n)$
  where $n : \mathbb{N}',\ e_0 : C\, 0',\ e_S : \Pi n{:}\mathbb{N}'.\, (C n \to C(S' n))$
- Define $f \equiv \mathit{rec}_{\mathbb{N}}\, \mathbb{N}'\, 0'\, (\lambda n{:}\mathbb{N}.\, S') : \mathbb{N} \to \mathbb{N}'$,
  $g \equiv \mathit{rec}_{\mathbb{N}'}\, \mathbb{N}\, 0\, (\lambda n{:}\mathbb{N}'.\, S) : \mathbb{N}' \to \mathbb{N}$; prove $\mathbb{N} \simeq \mathbb{N}'$.
- Discuss options to define $d' \equiv \mathit{double}' : \mathbb{N}' \to \mathbb{N}'$ and prove

$$\Pi n{:}\mathbb{N}'.\, (\mathit{double}'(S' n) = S'(S'(\mathit{double}'\, n)))$$

- HoTT: transport along $\mathbb{N} = \mathbb{N}'$, $(\mathbb{N}, S, d) = (\mathbb{N}', S', d')$

W-Types

- Purpose: encoding inductive types *uniformly*
- Formation: if $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, then $W\,A\,B : \mathcal{U}$
- Intuition: the type of A-labelled, $B(a)$-branching wf trees
- One constructor: $sup : \Pi a{:}A.\,(Ba \to W\,A\,B) \to W\,A\,B$
- Examples:
  - $N^W \equiv W\,\mathbf{2}\,(rec_{\mathbf{2}}\,\mathcal{U}\,\mathbf{0}\,\mathbf{1})$ (why?)
  - $0^W \equiv sup\,0_{\mathbf{2}}\,(rec_{\mathbf{0}}\,N^W)$, $S^W \equiv \lambda n{:}N^W.\,sup\,1_{\mathbf{2}}\,(\lambda y{:}\mathbf{1}.\,n)$ (!)
  - $List\,A \equiv W\,(\mathbf{1} + A)\,(rec_{(\mathbf{1}+A)}\,\mathcal{U}\,\mathbf{0}\,\lambda a.\,\mathbf{1})$ (why?)
  - $nil \equiv sup\,(inl \star)\,(rec_{\mathbf{0}}\,(List\,A))$, *cons* on blackboard
- Exercise: find the W-type for labeled binary trees
- Exercise 5.7: $(C \to \mathbf{0}) \to C$ is not a valid constructor type

Induction in W-Types

- Recall: $sup : \Pi a{:}A.\,((Ba \to W\,A\,B) \to W\,A\,B)$
- Intuition for induction: to prove Px for all $x : W\,A\,B$ it suffices to show that P is closed under sup. That is, for all $a : A$ and $f : Ba \to W\,A\,B$, if (IH) for all $b : Ba$ we have $P(f\ b)$, then $P(sup\,a\,f)$.
- Intuition for recursion: to define $h : W\,A\,B \to C$ it suffices to define $h\,(sup\,a\,f)$, for all $a : A$ and $f : Ba \to W\,A\,B$, using function values $h\,(f\ b)$ for $b : Ba$ (and possibly a, and f as predecesor).
- Examples: $dbl^W \equiv rec_{N^W}\ N^W\ e$, with $e\,0 \equiv \lambda f, g{:}\mathbf{0}{\to}N^W.\,0^W$ and $e\,1 \equiv \lambda f, g{:}\mathbf{1}{\to}N^W.\,(S^W(S^W(g\star)))$
- Exercise: define a predecessor $N^W \to N^W$

Homotopy-initial algebras

- $\mathbb{N}$-algebra: a type C with objects $c_0 : C$ and $c_S : C \to C$
- Formal definition: a Σ-type $\mathbb{N}Alg$ (on blackboard)
- $\mathbb{N}$-homomorphism between $\mathbb{N}$-algebras $(C, c_0, c_S), (D, d_0, d_S)$
- Formal definition: an even bigger Σ-type $\mathbb{N}Hom(\_, \_)$
- $\mathbb{N}$-algebras thus form a category
- H-initial $\mathbb{N}$-algebra I:

$$isHinit_{\mathbb{N}}(I) \equiv \Pi C{:}\mathbb{N}Alg.\, isContr(\mathbb{N}Hom(I, C))$$

- THM: any two h-initial $\mathbb{N}$-algebras are equal
- THM: the $\mathbb{N}$-algebra $(\mathbb{N}, 0, S)$ is h-initial
- THM: any W-algebra $(W\,A\,B, sup)$ is h-initial
- We skip 5.5–8

Higher Inductive Types

- Inductive type: constructors freely generate the objects
- Higher inductive type: some constructors generate objects of this type, called *points*, but others may generate *paths*, or even *higher paths*.
- Key Q: what is the equivalent of 'freely'? Induction?
- Example: the circle $\mathbb{S}^1$ (cf. $\mathbb{N}, \mathbf{2}$):
  - a point constructor $\mathsf{base} : \mathbb{S}^1$
  - a path constructor $\mathsf{loop} : \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}$.
- Generation: takes the relevant operations into account
  - On the point level: none (the type has no apriori structure)
  - On the path level: groupoid structure ($\cdot$, *refl*, $^{-1}$)
  - Not: $\mathsf{loop} = \mathit{refl}_{\mathsf{base}}$, $\mathsf{loop} \cdot \mathsf{loop} = \mathit{refl}_{\mathsf{base}}$

Higher Inductive Types

- Example: the circle $\mathbb{S}^1$:
  - a point constructor $\mathsf{base} : \mathbb{S}^1$
  - a path constructor $\mathsf{loop} : \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}$.
- What is $\mathsf{base} =_{\mathbb{S}^1} \mathsf{base}$? (should be $\mathbb{Z}$)
- Later: a path constructor $merid : A \to (N =_{Susp(A)} S)$ generates higher paths in $Susp(A)$ from paths in A
- Example: the 2-dimensional sphere $\mathbb{S}^2$:
  - a point constructor $\mathsf{base} : \mathbb{S}^2$
  - a 2-path constructor $\mathsf{surf} : refl_{\mathsf{base}} =_{\mathsf{base}=\mathsf{base}} refl_{\mathsf{base}}$.
- We have $\mathsf{surf} \neq refl_{refl_{\mathsf{base}}}$. What is $\mathsf{base} =_{\mathbb{S}^2} \mathsf{base}$? Book: there is an unexpected 3-path, cf. the Hopf fibration

Induction in HITs

- Induction in $\mathbb{N}$: to prove $\Pi x{:}\mathbb{N}.\, Px$, it suffices to have *base* in the fiber above 0, and *step* 'acting on the fibers above S'.
- By analogy, in $\mathbb{S}^1$: to prove $\Pi x{:}\mathbb{S}^1.\, Px$, it suffices to have b in the fiber above base, and ℓ 'acting on the fiber(s) above loop'.
- Want means 'fiber(s) above loop : base = base'?
- **Not**: a path $b = b$ in the fiber above $P(\text{base})$ (cf. $\mathit{refl}_{\text{base}}$)
- **But**: transport of b along loop plus a path $\text{loop}_*(b) = b$
- Recall transport $Px \to Py$, $P(\text{base}) \to P(\text{base})$
- Example: torus as fibration $P \to \mathbb{S}^1$, Fig. 6.1,2
- Induction: $b : P(\text{base})$ and $\ell : b =^{P}_{\text{loop}} b$ define $f : \Pi x{:}\mathbb{S}^1.\, Px$ with $f(\mathit{base}) =_\iota b$ and $\mathit{apd}_f(\text{loop}) = \ell$ (propositionally!)
- The last equality: a pragmatic choice (!)

Recursion in HITs

- Recursion: if $B : \mathcal{U}$, then $b : B$ and $\ell : (b =^P_{\text{loop}} b)$ define $f : \mathbb{S}^1 \to B$ with $f(base) =_\iota b$ and $apd_f(\text{loop}) = \ell$
- Recall the following transport lemmas, with $P : A \to \mathcal{U},\ f : A \to B,\ x, y : A,\ p : x =_A y$:
  - $f(p) \equiv ap_f\, p : fx =_B fy$ (Lem. 2.2.1)
  - $p_* \equiv transport^P p : Px \to Py$ (Lem. 2.3.1)
  - if $g : \Pi x{:}A.\, Px$, then $apd_g\, p : p_*(gx) =_{Py} gy$ (Lem. 2.3.4)
  - if $P \equiv \lambda x{:}A.\, B,\ b : B$, then $tpc^B_p b : (p_* b = b)$ (Lem. 2.3.5)
  - if $P \equiv \lambda x{:}A.\, B$, then $tpc^B_p(fx) \cdot ap_f\, p = apd_f\, p$ (Lem. 2.3.8)
- LEM: $a : A$ and $p : a =_A a$ define a *unique* (!) $f : \mathbb{S}^1 \to A$ with $f(\text{base}) =_\iota a$ and $ap_f(\text{loop}) = p$
- COR: $(\mathbb{S}^1 \to A) \simeq \Sigma x{:}A.\, (x =_A x)$ (univ. prop. of the circle)

The Interval

- The *interval* I is the HIT generated by:
  - a point constructor $0_I : I$
  - a point constructor $1_I : I$
  - a path constructor $seg : 0_I =_I 1_I$.
- Recursion: the following data defines a unique $f : I \to B$
  - points $b_0 : B$, $b_1 : B$, a path $s : b_0 =_B b_1$
- Induction: the following data defines a unique $f : \Pi x{:}I.\, Px$
  - points $b_0 : P0_I$, $b_1 : P1_I$, a path $s : b_0 =^P_{seg} b_1$
- I is contractible; I gives function extensionality by magic (!)

Properties of the Interval

- LEM: the interval I is contractible.
- Proof. Take 1_I as the center. By induction we prove $\Pi x{:}I.\, Px$ for $Px \equiv (x =_I 1_I)$. Take $seg : P\,0_I$ and $refl_{1_I} : P\,1_I$. We also need an inhabitant of $seg =^P_{seg} refl_{1_I}$. The latter type is $seg_*(seg) = refl_{1_I}$. By Lemma 2.11.2 we have $seg_*(seg) = seg^{-1} \cdot seg$ (picture) and by Lemma 2.1.4 $refl_{1_I} = seg^{-1} \cdot seg$.
- LEM: the interval I gives function extensionality (!)
- Proof. Let $f, g : A \to B$ and $p : \Pi x{:}A.\, fx =_B gx$. For every $x : A$, define $\tilde{p}_x : I \to B$ by $\tilde{p}_x\,0_I \equiv fx$, $\tilde{p}_x\,1_I \equiv gx$, $\tilde{p}_x(seg) = p$. Define $q : I \to (A \to B)$ by $q\,i = \lambda x{:}A.\,(\tilde{p}_x\,i)$. Then $q\,0_I =_\eta f$ and $q\,1_I =_\eta g$ and so $q(seg) : f =_{A \to B} g$.

More on the Circle

- LEM: the circle $\mathbb{S}^1$ is non-trivial: loop $\neq \mathit{refl}_{\mathsf{base}}$.
- Proof. If loop $= \mathit{refl}_{\mathsf{base}}$, then define for any $x : A$ and $p : x = x$ a function $f : \mathbb{S}^1 \to A$ by $f(\mathsf{base}) \equiv x$ and $f(\mathsf{loop}) = p$. By functoriality of ap_f we get $p = \mathit{refl}_x$. So $\Pi x{:}A.\, \Pi p{:}(x = x).\, (p = \mathit{refl}_x)$, which implies that A is a set (by path induction). Contradiction for $A = \mathcal{U}$, Example 3.1.9.
- LEM: the type $\Pi x{:}\mathbb{S}^1.\, (x = x)$ has an element that is not equal to $\lambda x{:}\mathbb{S}^1.\, \mathit{refl}_x$.
- Proof. Define $f : \Pi x{:}\mathbb{S}^1.\, (x = x)$ by induction taking $f(\mathsf{base}) \equiv \mathsf{loop}$ and $f(\mathsf{loop}) : \mathsf{loop}_*(\mathsf{loop}) = \mathsf{loop}$. By Lemma 2.11.2, with type family $\lambda x{:}\mathbb{S}^1.\, (x = x)$, $\mathsf{loop}_*(\mathsf{loop}) = \mathsf{loop}^{-1} \cdot \mathsf{loop} \cdot \mathsf{loop}$, so $f(\mathsf{loop}) = \mathit{refl}_{\mathsf{loop}}$ is OK. By *happly*, the previous lemma implies $f \neq \lambda x{:}\mathbb{S}^1.\, \mathit{refl}_x$.
- COR: if $\mathbb{S}^1 = \mathcal{U}_n$, then $\mathcal{U}_n$ is not a 1-type (?)

The 2-Sphere

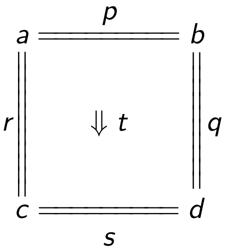
- Recall: the 2-dimensional sphere $\mathbb{S}^2$:
  - a point constructor base : $\mathbb{S}^2$
  - a 2-path constructor surf : $\mathit{refl}_{\mathrm{base}} =_{\mathrm{base}=\mathrm{base}} \mathit{refl}_{\mathrm{base}}$.
- Recursion: the following data defines a unique $f : \mathbb{S}^2 \to B$
  - a point $b : B$, a path $s : \mathit{refl}_b =_{b =_B b} \mathit{refl}_b$
  - we get $f(\mathrm{base}) \equiv b$, and $\mathit{apap}_f(\mathrm{surf}) = s$ (!)
- Induction: the following data defines a unique $f : \Pi x{:}\mathbb{S}^2.\ Px$
  - a point $b : P\,\mathrm{base}$, a path $s : \mathit{refl}_b =^P_{\mathrm{surf}} \mathit{refl}_b$ ...
  - ... and this gets complicated with *trtr* along a 2-path ...

Suspensions

- For any $A : \mathcal{U}$ we define a HIT $Susp(A)$ by :
  - two point constructors $N, S : Susp(A)$
  - a path constructor $merid : A \to (N =_{Susp(A)} S)$
- NB: the path constructor generates higher paths in $Susp(A)$
- Recursion: the following data defines a unique $f : Susp(A) \to B$
  - points $n, s : B$, a path function $m : A \to n =_B s$
- Induction: the following data defines a unique $f : \Pi x{:}Susp(A).\, Px$
  - points $n : P(N)$, $s : P(S)$, and $m : \Pi a{:}A.\, (n =^{P}_{merid(a)} s)$
- LEM: $Susp(\mathbf{0}) \simeq \mathbf{2}$, $Susp(\mathbf{2}) \simeq \mathbb{S}^1$, $\mathbb{S}^{n+1} \simeq Susp(\mathbb{S}^n)$
- Proofs: easy, medium (uses 2.11.3), difficult
- LEM 2.11.3 implies: $tr^{\lambda x.\,(hx=x)} pq = h(p^{-1}) \cdot q \cdot p$

The torus

- The *torus* is the HIT T^2 defined by :
  - a point $b : T^2$
  - two paths $p, q : b = b$
  - a 2-path $t : p \cdot q = q \cdot p$
- Intuition: put $r = q$ and $s = r$ in



- Very tricky induction principle (because of the 2-path)
- LEM: $T^2 \simeq \mathbb{S}^1 \times \mathbb{S}^1$

Truncation

- For every $A : \mathcal{U}$ we define the HIT $\|A\| : \mathcal{U}$ by:
  - a function $|\_| : A \to \|A\|$
  - a path function $path : \Pi x, y{:}\|A\|. (x =_{\|A\|} y)$
- Recursion: the following data defines a function $f : \|A\| \to B$ satisfying $f|a| \equiv ga$ and $ap_f(path(|a|, |a|')) = p(a, a')$:
  - a function $g : A {\to} B$ and a path function $p : \Pi x, y{:}B. (x =_B y)$
- LEM: $\|\mathbf{2}\|$ gives function extensionality
- Proof: let $f, g : A {\to} B$ and $p : f \sim g$. Define $p_x : \mathbf{2} \to \Sigma y{:}B. fx =_B y$. Note that $\Sigma y{:}B. fx =_B y$ is contractible. etc.

Homotopy *n*-levels

- Intuition: no interesting homotopy above dimension n
- Definition of homotopy n-levels:
  - $is(-2)type(A) \equiv isContr(A)$ (equivalent $A \simeq \mathbf{1}$)
  - $is(-1)type(A) \equiv \Pi x, y{:}A.\, is(-2)type(x =_A y)$ $(isProp(A))$
  - $is(0)type(A) \equiv \Pi x, y{:}A.\, is(-1)type(x =_A y)$ $(isSet(A))$
  - $is(n+1)type(A) \equiv \Pi x, y{:}A.\, is(n)type(x =_A y)$ $(n \geq -2)$
- Idea: understanding a space through its (higher) path spaces
- Later: n-truncation, trivializing homotopy above dimension n
- Later: n-connected, that is, n-truncation is contractible, means no interesting homotopy in or below dimension n
- $\mathbb{S}^1$ is not an 0-type, $\mathbb{S}^{n+1}$ is not an n-type
- CONJ: $\mathcal{U}$ is not an n-type for any n

Closure properties of homotopy n-levels

- LEM: if $p : X \to Y$ a retraction and X is an n-type, then Y is an n-type ($n \geq -2$)
- Proof: induction on $n \geq -2$. Base case easy. Assume OK for n and let $s : Y \to X$ with homotopy $\epsilon : p \circ s \sim 1$. Assume $\Pi x, x' {:} X.\, is(n)type(x =_X x')$, to prove $is(n)type(y =_Y y')$ for all $y, y' : Y$. Let $y, y' : Y$, then $sy =_X sy'$ is an n-type. By IH it suffices that $y =_Y y'$ is a retract of $sy =_X sy'$. Take ap_s and $t(q) \equiv \epsilon_y^{-1} \cdot p(q) \cdot \epsilon_{y'}$ and use naturality of ϵ (picture).
- COR: if $X \simeq Y$ and X is an n-type, then so is Y ($n \geq -2$)
- LEM: if X is an n-type, then X is also an $(n+1)$-type ($n \geq -2$). So, the levels are cumulative.
- Proof: by induction on $n \geq -2$

More closure properties of homotopy *n*-levels

- LEM: if $f : X \to Y$ an embedding and Y is an n-type, then so is X ($n \geq -1$)
- Proof: $x =_X x' \simeq fx =_Y fx'$ for embedding f. NB $f : \mathbf{0} \to \mathbf{1}$
- LEM: for $A : \mathcal{U}$, $B : A \to \mathcal{U}$, if Ba is an n-type for every $a : A$, then $\Pi\, A\, B$ is an n-type ($n \geq -2$)
- LEM: for $A : \mathcal{U}$, $B : A \to \mathcal{U}$, if A is an n-type and Ba is an n-type for every $a : A$, then $\Sigma\, A\, B$ is an n-type, for all $n \geq -2$
- LEM: for $A : \mathcal{U}$, $is(n)type(A)$ is a mere proposition ($n \geq -2$)
- DEF: $n\text{-}type_{\mathcal{U}} = \Sigma X{:}\mathcal{U}.\, is(n)type(X)$, for all $n \geq -2$
- LEM: $n\text{-}type_{\mathcal{U}}$ is an $(n+1)$-type, for all $n \geq -2$
- Proofs: by induction on $n \geq -2$

Uniqueness of Identity Proofs

- Axiom UIP(X): for all $x, y : X$ and $p, q : x =_X y$ we postulate $p = q$ (NB $UIP(X) \equiv isSet(X)$)
- Axiom K(X): for all $x : X$ and $p : x =_X x$ we postulate $p = refl_x$
- LEM: $K(X) \simeq UIP(X)$
- LEM: if $R : X \to X \to \mathcal{U}$ a reflexive mere relation implying $=_X$, then (1) $isSet(X)$ and (2) $\Pi x, y{:}X. (Rxy \simeq (x =_X y))$
- Proof: note that (1) and (2) are equivalent; prove, e.g., $K(X)$
- COR: if $\neg\neg(x =_X y) \to (x =_X y)$, then X is a set
- COR: if $(x =_X y) \vee \neg(x =_X y)$, then X is a set
- COR: $\mathbb{N}$ is a set (prove by induction that $=_{\mathbb{N}}$ is decidable)

n-Truncations

- Idea: n-truncation removes all interesting homotopy above dimension n
- DEF: for every $A : \mathcal{U}$, define:
  - (-2)-truncation: $\|A\|_{-2} \equiv \mathbf{1}$ ('contractible' truncation)
  - (-1)-truncation: $\|A\|_{-1} \equiv \|A\|$ (propositional truncation)
  - (0)-truncation: $\|A\|_0$ is defined as a HIT with two constructors: a function $|\_|_0 : A \to \|A\|_0$, and a path function $2path : \Pi x, y{:}\|A\|_0 .\, \Pi p, q{:}(x =_{\|A\|_0} y).\, (p =_{x =_{\|A\|_0} y} q)$
- The general definition in the book uses $\mathbb{S}^{n+1}$ (complicated)
- LEM: for all $n \geq -2$ we have that $\|A\|_n$ is an n-type
- Induction, recursion, properties ...

n-Connectedness

- Idea: *n*-connectness expresses that there is no interesting homotopy in and below dimension n
- DEF: for types $A : \mathcal{U}$, $conn_n(A) \equiv isContr(\|A\|_n)$
- DEF: function $f : A \to B$ is *n*-connected, if for any $b : B$, the fiber of f in b is connected, $conn_n(fib_f(b))$
- DEF: function $f : A \to B$ is *n*-truncated, if for any $b : B$, the fiber of f in b is *n*-truncated, $is(n)type(fib_f(b))$
- LEM: any function factors as an *n*-connected function followed by an *n*-truncated function (generalized epi-mono-decomposition)