MATH 721: HOMOTOPY TYPE THEORY

EMILY RIEHL

Contents

Part 1. Martin-Löf's Dependent Type Theory

August 30: Dependent Type Theory

Martin-Löf's dependent type theory is a formal language for writing mathematics: both constructions of mathematical objects and proofs of mathematical propositions. As we shall discover, these two things are treated in parallel (in contrast

Date: Fall 2021.

to classical Set theory plus first-order logic, where the latter supplies the proof calculus and the former gives the language which you use to state things to prove).

Judgments and contexts. I find it helpful to imagine I'm teaching a computer to do mathematics. It's also helpful to forget that you know other ways of doing mathematics.[1]

defn. There are four kinds of **judgments** in dependent type theory, which you can think of as the "grammatically correct" expressions:

(i) $\Gamma \vdash A\ \texttt{type}$, meaning that A is a well-formed type in **context** Γ (more about this soon).
(ii) $\Gamma \vdash a : A$, meaning that a is a well-formed term of type A in context Γ.
(iii) $\Gamma \vdash A \doteq B\ \texttt{type}$, meaning that A and B are **judgmentally** or **definitionally** equal types in context Γ.
(iv) $\Gamma \vdash a \doteq b : A$, meaning that a and b are judgmentally equal terms of type A in context Γ.

These might be collectively abbreviated by $\Gamma \vdash \mathcal{J}$.

The statement of a mathematical theorem, often begins with an expression like "Let n and m be positive integers, with $n < m$, and let $\vec{v}_1, \dots, \vec{v}_m$ be vectors in $\mathbb{R}^n$. Then ..." This statement of the hypotheses defines a **context**, a finite list of types and hypothetical terms (called **variables**[2]) satisfying an inductive condition that that each type can be derived in the context of the previous types and terms using the inference rules of type theory.

defn. A **context** is a finite list of variable declarations:

$$x : A_1, x_2 : A_2(x_1), \dots, x_n : A_n(x_1, \dots, x_{n-1})$$

satisfying the condition that for each $1 \leqslant k \leqslant n$ we can derive the judgment

$$x_1 : A_1, \dots, x_{k-1} : A_{k-1}(x_1, \dots, x_{k-2}) \vdash A_k(x_1, \dots, x_{k-1})\ \texttt{type}$$

using the inference rules of type theory.

We'll introduce the inference rules shortly but the idea is that it needs to be possible to form the type $A_k(x_1, \dots, x_{k-1})$ given terms $x_1, \dots, x_{k-1}$ of the previously-formed types.

ex. For example, there is a unique context of length zero: the empty context.

ex. $n : \mathbb{N}, m : \mathbb{N}, p : n < m, \vec{v} : (\mathbb{R}^n)^m$ is a context. Here $n : \mathbb{N}, m : \mathbb{N} \vdash n < m$ is a dependent type that corresponds to the relation $\{n < m \mid n, m \in \mathbb{N}\} \subset \mathbb{N} \times \mathbb{N}$ and the variable p is a witness that $n < m$ is true (more about this later).

Type families. Absolutely everything in dependent type theory is context dependent so we always assume we're working in a background context Γ. Let's focus on the primary two judgment forms.

defn. Given a type A in context Γ a **family** of types over A in context Γ is a type $B(x)$ in context $\Gamma, x : A$, as represented by the judgment:

$$\Gamma, x : A \vdash B(x)\ \texttt{type}$$

We also say that $B(x)$ is a type **indexed** by $x : A$, in context Γ.

ex. $\mathbb{R}^n$ is a type indexed by $n \in \mathbb{N}$.

defn. Consider a type family B over A in context Γ. A **section** of the family B over A in context Γ is a term of type $B(x)$ in context $\Gamma, x : A$, as represented by the judgment:

$$\Gamma, x : A \vdash b(x) : B(x)$$

We say that b is a **section** of the family B over A in context Γ or that $b(x)$ is a term of type $B(x)$ indexed by $x : A$ in context Γ.

ex. $\vec{0}_n : \mathbb{R}^n$ is a term dependent on $n \in \mathbb{N}$.

Exercise. If you've heard the word "section" before you should think about what it is being used here.

[1] Indeed, there are very deep theorems that describe how to interpret dependent type theory into classical set-based mathematics. You're welcome to investigate these for your final project but they are beyond the scope of this course.

[2] We're not going to say anything about proper syntax for variables and instead rely on instinct to recognize proper and improper usage.

Inference rules. There are five types of inference rules that collectively describe the structural rules of dependent type theory. They are

(i) Rules postulating that judgmental equality is an equivalence relation:

$$\frac{\Gamma \vdash A\ \mathtt{type}}{\Gamma \vdash A \doteq A\ \mathtt{type}} \qquad \frac{\Gamma \vdash A \doteq B\ \mathtt{type}}{\Gamma \vdash B \doteq A\ \mathtt{type}} \qquad \frac{\Gamma \vdash A \doteq B\ \mathtt{type} \qquad \Gamma \vdash B \doteq C\ \mathtt{type}}{\Gamma \vdash A \doteq C\ \mathtt{type}}$$

and similarly for judgmental equality between terms.

(ii) Variable conversion rules for judgmental equality between types:

$$\frac{\Gamma \vdash A \doteq A'\ \mathtt{type} \qquad \Gamma, x : A, \Delta \vdash \mathcal{J}}{\Gamma, x : A', \Delta \vdash \mathcal{J}}$$

(iii) Substitution rules:

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

If Δ is the context $y_1 : B_1(x), \dots, y_n : B_n(x, y_1, \dots, y_{n-1})$ then $\Delta[a/x]$ is the context $y_1 : B(a), \dots, y_n : B_n(a, y_1, \dots, y_{n-1})$. A similar substitution is performed in the judgment $\mathcal{J}[a/x]$. Further rules indicate that substitution by judgmentally equal terms gives judgmentally equal results.

(iv) Weakening rules:

$$\frac{\Gamma \vdash A\ \mathtt{type} \qquad \Gamma, \Delta \vdash \mathcal{J}}{\Gamma, x : A, \Delta \vdash \mathcal{J}}$$

Eg if A and B are types in context Γ, then B is also a type in context $\Gamma, x : A$.

(v) The generic term:

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

This will be used to define the identity function of any type.

Derivations. A derivation in type theory is a finite rooted tree where each node is a valid rule of inference. The root is the conclusion.

ex. The interchange rule is derived as follows

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash B\ \mathtt{type}}{\Gamma, y : B \vdash y : B}}{\Gamma, y : B, x : A \vdash y : B} \qquad \dfrac{\Gamma \vdash B\ \mathtt{type} \qquad \dfrac{\Gamma, x : A, y : B, \Delta \vdash \mathcal{J}}{\Gamma, x : A, z : B, \Delta[z/y] \vdash \mathcal{J}[z/y]}}{\Gamma, y : B, x : A, z : B, \Delta[z/y] \vdash \mathcal{J}[z/y]}}{\Gamma, y : B, x : A, \Delta \vdash \mathcal{J}}$$

September 1: Dependent function types & the natural numbers

The rules for dependent function types. Consider a section b of a family B over A in context Γ, as encoded by a judgment:

$$\Gamma, x : A \vdash b(x) : B(x).$$

We think of the section b as a function that takes as input $x : A$ and produces a term $b(x) : B(x)$. Since the type of the output is allowed to depend on the term being input, this isn't quite an ordinary function but a **dependent function**. The type of all dependent functions is the **dependent function type**

$$\Pi_{x:A} B(x)$$

What is a thing in mathematics? Structuralism says the ontology of a thing is determined by its behavior. In dependent type theory, we define dependent function types by stating their rules, which have the following forms:

(i) **formation rules** tell us how a type may be formed
(ii) **introduction rules** tell us how to introduce new terms of the type
(iii) **elimination rules** tell us how the terms of a type may be used
(iv) **computation rules** tell us how the introduction and elimination rules interact

There are also **congruence rules** that tell us that all constructions respect judgmental equality. See [R] for more details.

defn (dependent function types)**.** The Π-**formation rule** has the form:

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

The Π-**introduction rule** has the form:

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

The λ-**abstraction** $\lambda x.b(x)$ can be thought of as notation for $x \mapsto b(x)$.

The Π-**elimination rule** has the form of the evaluation function:

$$\frac{\Gamma \vdash f : \Pi_{x:A} B(x)}{\Gamma, x : A \vdash f(x) : B(x)}$$

Finally, there are two computation rules: the β-**rule**

$$\frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma, x : A \vdash (\lambda y.b(y))(x) \doteq b(x) : B(x)}$$

and the η-**rule**, which says that all elements of a Π-type are dependent functions:

$$\frac{\Gamma \vdash f : \Pi_{x:A} B(x)}{\Gamma \vdash \lambda x.f(x) \doteq f : \Pi_{x:A} B(x)}$$

Ordinary function types.

defn (function types)**.** The formation rule is derived from the formation rule for Π-types together with weakening:

$$\frac{\dfrac{\Gamma \vdash A\ \mathtt{type} \qquad \Gamma \vdash B\ \mathtt{type}}{\Gamma, x : A \vdash B\ \mathtt{type}}}{\Gamma \vdash \Pi_{x:A} B\ \mathtt{type}}$$

We adopt the notation

$$A \to B \coloneqq \Pi_{x:A} B$$

for the dependent function type in the case where the type family B is constant over $x : A$.

The introduction, evaluation, and computation rules are instances of term conversion: eg

$$\frac{\Gamma \vdash B\ \mathtt{type} \qquad \Gamma, x : A \vdash b(x) : B}{\Gamma \vdash \lambda x.b(x) : A \to B} \qquad \frac{\Gamma \vdash f : A \to B}{\Gamma, x : A \vdash f(x) : B}$$

plus the two computation rules:

$$\frac{\Gamma \vdash B\ \mathtt{type} \qquad \Gamma, x : A \vdash b(x) : B}{\Gamma, x : A \vdash (\lambda y.b(y))(x) \doteq b(x) : B} \qquad \frac{\Gamma \vdash f : A \to B}{\Gamma \vdash \lambda x.f(x) \doteq f : A \to B}$$

defn. Identity functions are defined as follows:

$$\frac{\dfrac{\Gamma \vdash A\ \mathtt{type}}{\Gamma, x : A \vdash x : A}}{\Gamma \vdash \lambda x.x : A \to A}$$

which is traditionally denoted by $\mathtt{id}_A \coloneqq \lambda x.x$.

The idea of composition is that given a function $f\colon A \to B$ and $g\colon B \to C$ you should get a function $g \circ f\colon A \to C$. Using infix notation you might denote this function by $\_ \circ \_$.

Q. $\_ \circ \_$ is itself a function, so it's a term of some type. What type?[3]

[3] Really the type should involve three universe variables but let's save this for next week.

defn. Composition has the form:

$$\frac{\Gamma \vdash A\ \mathtt{type} \qquad \Gamma \vdash B\ \mathtt{type} \qquad \Gamma \vdash C\ \mathtt{type}}{\Gamma \vdash \_ \circ \_ : (B \to C) \to ((A \to B) \to (A \to C))}$$

It is defined by

$$\_ \circ \_ \coloneqq \lambda g.\lambda f.\lambda x.g(f(x))$$

which can be understood as the term constructed by three applications of the Π-introduction rule followed by two applications of the Π-elimination rule.

Composition is associative essentially because both $(h \circ g) \circ f$ and $h \circ (g \circ f)$ are defined by $\lambda x.h(g(f(x)))$. We'll think about this more formally when we come back to identity types.

Similarly, you can compute that for all $f : A \to B$, $\mathtt{id}_B \circ f \doteq f : A \to B$ and $f \circ \mathtt{id}_A \doteq f : A \to B$.

The type of natural numbers. The type $\mathbb{N}$ of natural numbers is the archetypical example of an **inductive type** about more which soon. It is given by rules which say that it has a term $0_{\mathbb{N}} : \mathbb{N}$, it has a successor function $\mathtt{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ and it satisfies the induction principle.

The $\mathbb{N}$-formation rule is

$$\frac{}{\vdash \mathbb{N}\ \mathtt{type}}$$

In other words, $\mathbb{N}$ is a type in the empty context.

There are two $\mathbb{N}$-introduction rules:

$$\frac{}{\vdash 0_{\mathbb{N}} : \mathbb{N}} \qquad \frac{}{\vdash \mathtt{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}$$

Digression (traditional induction). In traditional first-order logic, the principle of $\mathbb{N}$-induction is stated in terms of a **predicate** P over $\mathbb{N}$. One way to think about P is as a function $P\colon \mathbb{N} \to \{\top, \bot\}$. That is, for each $n \in \mathbb{N}$, $P(n)$ is either true or false. We could also think of P as an indexed family of sets $(P(n))_{n \in \mathbb{N}}$ where for each n either $P(n) = \varnothing$ (corresponding to $P(n)$ being false) or $P(n) = *$ (corresponding to $P(n)$ being true).

The induction principle then says

$$\forall P : \{0,1\}^{\mathbb{N}}, (P(0) \wedge (\forall n, P(n) \to P(n+1)) \to \forall n, P(n)).$$

In dependent type theory it is most natural to let P be an arbitrary type family over $\mathbb{N}$. This is a stronger assumption, as we'll see.

Q. What then corresponds to a proof that $\forall n, P(n)$?

The induction principle is encoded by the following rule:

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \mathtt{type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \Pi_{n:\mathbb{N}}(P(n) \to P(\mathtt{succ}_{\mathbb{N}}(n)))}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}(p_0, p_S) : \Pi_{n:\mathbb{N}} P(n)}$$

Remark. There are other forms this rule might take that are interderivable with this one.

The computation rules say that the function $\mathtt{ind}_{\mathbb{N}}(p_0, p_S) : \Pi_{n:\mathbb{N}} P(n)$ behaves like it should on $0_{\mathbb{N}}$ and successors:

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \mathtt{type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \Pi_{n:\mathbb{N}}(P(n) \to P(\mathtt{succ}_{\mathbb{N}}(n)))}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}(p_0, p_S)(0_{\mathbb{N}}) \doteq p_0 : P(0_{\mathbb{N}})}$$

and under the same premises

$$\Gamma, n : \mathbb{N} \vdash \mathtt{ind}_{\mathbb{N}}(p_0, p_S)(\mathtt{succ}_{\mathbb{N}}(n)) \doteq p_S(n, \mathtt{ind}_{\mathbb{N}}(p_0, p_S, n)) : P(\mathtt{succ}_{\mathbb{N}}(n)).$$

These computation rules don't matter so much if the type family $n : \mathbb{N} \vdash P(n)$ is really a predicate — $P(n)$ is either true or false and that's the end of the story — but they do matter if $P(n)$ is more like an indexed family of sets. In the latter case, $\mathtt{ind}_{\mathbb{N}}(p_0, p_S)$ is the recursive function defined from p_0 and p_S and these are the computation rules for that recursion.

Remark. Recall Peano's axioms for the natural numbers:

(i) $0_{\mathbb{N}} \in \mathbb{N}$
(ii) $\mathtt{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$

(iii) $\forall n, \mathsf{succ}_{\mathbb{N}}(n) \neq 0_{\mathbb{N}}$
(iv) $\forall n, m, \mathsf{succ}_{\mathbb{N}}(n) = \mathsf{succ}_{\mathbb{N}}(m) \to n = m$
(v) induction

We'll be able to *prove* the missing two axioms from the induction principle we've assumed once we have identity types and universes. We'll come back to this in a few weeks.

Addition on the natural numbers.

Remark. When addition is defined by recursion on the second variable, from the computation rules associated to function types and the natural numbers type you can derive judgmental equalities

$$m + 0 \doteq m \quad \text{and} \quad m + \mathsf{succ}_{\mathbb{N}}(n) \doteq \mathsf{succ}_{\mathbb{N}}(m + n).$$

But you can't derive the symmetric judgmental equalities.

We *will* be able to prove such equalities using the identity types, to be introduced shortly.

Pattern matching. To define a dependent function $f : \Pi_{n:\mathbb{N}} P(n)$ by induction on n it suffices, by the elimination rule for the natural numbers type, to provide two terms:

$$p_0 : P(0_{\mathbb{N}}) \qquad p_S : \Pi_{n:\mathbb{N}} P(n) \to P(\mathsf{succ}_{\mathbb{N}}(n)).$$

Thus the definition of f may be presented by writing

$$f(0_{\mathbb{N}}) \coloneqq p_0 \qquad f(\mathsf{succ}_{\mathbb{N}}(n)) \coloneqq p_S(n, f(n)).$$

This defines the function f by **pattern matching** on the variable n. When a function is defined in this form, the judgmental equalities accompanying the definition are immediately displayed.

September 8: The formal proof assistant `agda`

See https://github.com/emilyriehl/721/blob/master/introduction.agda

September 13: Inductive types

The rules for the natural numbers type $\mathbb{N}$ tell us:

(i) how to form terms in $\mathbb{N}$, and
(ii) how to define dependent functions in $\Pi_{n:\mathbb{N}} P(n)$ for any type family $n : \mathbb{N} \vdash P(n)$ `type` ,

while providing two computation rules for those dependent functions.

Many types can be specified by stating how to form their terms and how to define dependent functions out of them. Such types are called **inductive types**.

The idea of inductive types. Recall a type is specified by its formation rules, its introduction rules, its elimination rules, and its computation rules. For inductive types, the introduction rules specify the **constructors** of the inductive type, while the elimination rule provides the **induction principle**. The computation rules provide definitional equalities for the induction principle.

In more detail:

(i) The constructors tell us what structure the identity type is given with.
(ii) The induction principle defines sections of any type family over the inductive type by specifying the behavior at the constructors.
(iii) The computation rules assert that the inductively defined section agrees on the constructors with the data used to define it. So there is one computation rule for each constructor.

The unit type. The formal definition of the **unit type** is as follows:

$$\vdash \mathbb{1}\ \mathtt{type} \qquad \vdash \star : \mathbb{1} \qquad \frac{x : \mathbb{1} \vdash P(x)\ \mathtt{type} \qquad p : P(\star)}{x : \mathbb{1} \vdash \mathtt{ind}_{\mathbb{1}}(p, x) : P(x)} \qquad \frac{x : \mathbb{1} \vdash P(x)\ \mathtt{type} \qquad p : P(\star)}{x : \mathbb{1} \vdash \mathtt{ind}_{1}(p, \star) \doteq p : P(\star)}$$

As an inductive type, the definition is packaged as follows:

defn. The **unit type** is a type $\mathbb{1}$ equipped with a term $\star : \mathbb{1}$ satisfying the inductive principle that for any family $x : \mathbb{1} \vdash P(x)$ there is a function

$$\mathtt{ind}_{\mathbb{1}} : P(\star) \to \Pi_{x:1} P(x)$$

with the computation rule $\mathtt{ind}_1(p, \star) \doteq p$.

In agda, this definition has the form:

```
data unit : UU lzero where
  star : unit
```

Q. What does the induction rule look like for a constant type family A that does not depend on $\mathbb{1}$?

The empty type.

defn. The empty type is a type $\varnothing$ satisfying the induction principle that for any family of types $x : \varnothing \vdash P(x)$ there is a term

$$\mathtt{ind}_{\varnothing} : \Pi_{x:\varnothing} P(x).$$

That is the empty type is the inductive type with no constructors. Thus there are no computation rules. In agda, this definition has the form:

```
data empty : UU lzero where
```

Remark. As a special case of the elimination rule for the empty type we have

$$\frac{\vdash A\ \mathtt{type}}{\mathtt{ex\text{-}falso} \coloneqq \mathtt{ind}_{\varnothing} : \varnothing \to A}$$

By the elimination rule for function types it follows that if we had a term $x : \varnothing$ then we could get a term in any type. The name comes from latin *ex falso quodlibet*: "from falsehood, anything."

We've already seen a few glimpses of logic in type theory, something we'll discuss more formally soon. The basic idea is that we can interpret the formation of a type as akin to the process of formulating a mathematical statement that could be a sentence (if its a type in the empty context) or a predicate (if it's a dependent type). The act of constructing a term in that type is then analogous to proving the proposition so-encoded. These ideas motivate the logically-inflected terms in what follows.

For instance, we can use the empty type to define a negation operation on types:

defn. For any type A, we define its **negation** by $\neg A \coloneqq A \to \varnothing$ and say the type A **is empty** if there is a term in this type.

Remark. To construct a term of type $\neg A$, use the introduction rule for function types and assume given a term $a : A$. The task then is to derive a term of $\varnothing$. In other words, we prove $\neg A$ by assuming A and deriving a contradiction. This proof technique is called **proof of negation**.

This should be contrasted with **proof by contradiction**, which aims to prove a proposition P by assuming $\neg P$ and deriving a contradiction. This uses the logical step "$\neg\neg P$ implies P." In type theory, however, $\neg\neg A$ is the type of functions

$$\neg\neg A \coloneqq (A \to \varnothing) \to \varnothing)$$

and it is not possible in general to use a term in this type to construct a term of type A.

The law of contraposition does work, at least in one direction.

Proposition. *For any types P and Q there is a function*

$$(P \to Q) \to (\neg Q \to \neg P).$$

Proof. By λ-abstraction assume given $f : P \to Q$ and $\tilde{q} : Q \to \varnothing$. We seek a term in $P \to \varnothing$, which we obtain simply by composing: $\tilde{q} \circ f : P \to \varnothing$. Thus

$$\lambda f.\lambda \tilde{q}.\lambda p.\tilde{q}(f(p)) : (P \to Q) \to (\neg Q \to \neg P). \qquad \square$$

Coproducts. Inductive types can be defined outside the empty context. For instance, the formation and introduction rules for the coproduct type have the form:

$$\frac{\Gamma \vdash A\ \mathtt{type} \qquad \Gamma \vdash B\ \mathtt{type}}{\Gamma \vdash A + B\ \mathtt{type}}$$

$$\frac{\Gamma \vdash A\ \mathtt{type} \qquad \Gamma \vdash B\ \mathtt{type} \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathtt{inl} a : A + B} \qquad \frac{\Gamma \vdash A\ \mathtt{type} \qquad \Gamma \vdash B\ \mathtt{type} \qquad \Gamma \vdash b : B}{\Gamma \vdash \mathtt{inr} b : A + B}$$

defn. Given types A and B the **coproduct type** is the type equipped with

$$\mathtt{inl} : A \to A + B \qquad \mathtt{inr} : B \to A + B$$

satisfying the induction principle that says that for any family of types $x : A + B \vdash P(x)\ \mathtt{type}$ there is a term

$$\mathtt{ind}_+ : (\Pi_{x:A} P(\mathtt{inl}(x))) \to \left(\Pi_{y:B} P(\mathtt{inr}(y))\right) \to \Pi_{z:A+B} P(z)$$

satisfying the computation rules

$$\mathtt{ind}_+(f, g, \mathtt{inl}(x)) \doteq f(x) \qquad \mathtt{ind}_+(f, g, \mathtt{inr}(y)) \doteq g(y).$$

Not as a special case we have

$$\mathtt{ind}_+ : (A \to X) \to (B \to X) \to (A + B \to X)$$

which is similar to the elimination rule for disjunction in first order logic: if you've proven that A implies X and that B implies X then you can conclude that A or B implies X.

The type of integers. There are many ways to define the integers in Martin-Löf type theory, one of which is as follows:

defn. Define the **integers** to be the type $\mathbb{Z} \coloneqq \mathbb{N} + (\mathbb{1} + \mathbb{N})$ which comes equipped with inclusions:

$$\mathtt{in\text{-}pos} \coloneqq \mathtt{inr} \circ \mathtt{inr} : \mathbb{N} \to \mathbb{Z} \qquad \mathtt{in\text{-}neg} \coloneqq \mathtt{inl} : \mathbb{N} \to \mathbb{Z}$$

and constants

$$-1_{\mathbb{Z}} \coloneqq \mathtt{in\text{-}neg}(0_{\mathbb{N}}) \qquad 0_{\mathbb{Z}} \coloneqq \mathtt{inr}(\mathtt{inl}(\star)) \qquad 1_{\mathbb{Z}} \coloneqq \mathtt{in\text{-}pos}(0_{\mathbb{N}}).$$

Since $\mathbb{Z}$ is built from inductive types it is then an inductive type given with its own induction principle.

Dependent pair types. Of all the inductive types we've introduced, the final one is perhaps the most important.

Recall a **dependent function** $\lambda x.f(x) : \Pi_{x:A} B(x)$ is like an ordinary function except the output type is allowed to vary with the input term. Similarly, a **dependent pair** $(a, b) : \Sigma_{x:A} B(x)$ is like an ordinary (ordered) pair except the type of the second term $b : B(a)$ is allowed to vary with the first term $a : A$.

defn. Consider a type family $x : A \vdash B(x)\ \mathtt{type}$. The **dependent pair type** or **Σ-type** $\Sigma_{x:A} B(x)$ is the inductive type equipped with the function

$$\mathtt{pair} : \Pi_{x:A} \left(B(x) \to \Sigma_{y:A} B(y)\right).$$

The induction principle asserts that for any family of types $p : \Sigma_{x:A} B(x) \vdash P(p)\ \mathtt{type}$ there is a function

$$\mathtt{ind}_\Sigma : \left(\Pi_{x:A} \Pi_{y:B} P(\mathtt{pair}(x, y))\right) \to \left(\Pi_{z:\Sigma_{x:A} B(x)} P(z)\right)$$

satisfying the computation rule $\mathtt{ind}_\Sigma(g, \mathtt{pair}(x, y)) \doteq g(x, y)$.

It is common to write "(x, y)" as shorthand for "$\mathtt{pair}(x, y)$."

defn. Given a type family $x : A \vdash B(x)\ \mathtt{type}$ by the induction principle for Σ-types, we have a function

$$\mathtt{pr}_1 : \Sigma_{x:A} B(x) \to A$$

defined by $\mathtt{pr}_1(x, y) \coloneqq x$ and a dependent function

$$\mathtt{pr}_2 : \Pi_{p:\Sigma_{x:A} B(x)} B(\mathtt{pr}_1(p))$$

defined by $\mathtt{pr}_2(x, y) \coloneqq y$.

When B is a constant type family over A, the type $\Sigma_{x:A} B$ is the type of ordinary pairs (x, y) where $x : A$ and $y : B$. Thus **product types** arise as special cases of Σ-types.

defn. Given types A and B their product type is the type $A \times B \coloneqq \Sigma_{x:A} B$. It comes with a pairing function

$$(-,-) : A \to B \to A \times B$$

and satisfies an induction principle:

$$\mathtt{ind}_\times : \Pi_{x:A}\Pi_{y:B} P(x,y) \to \Pi_{z:A\times B} P(z)$$

satisfying the computation rule $\mathtt{ind}_\times(g,(x,y)) \doteq g(x,y)$.

As a special case, we have

$$\mathtt{ind}_\times : (A \to B \to C) \to ((A \times B) \to C).$$

This is the inverse of the **currying function**. Thus $\mathtt{ind}_\times$ and $\mathtt{ind}_\Sigma$ sometimes go by the name **uncurrying**.

September 15: Identity types

We have started to develop an analogy in which types play the role of mathematical propositions and terms in a type play the role of proofs of that proposition. More exactly, we might think of a type as a "proof-relevant" proposition, the distinction being that the individual proofs of a given proposition—the terms of the type—are first class mathematical objects, which may be used as ingredients in future proofs, rather than mere witnesses to the truth of the particular proposition.

The various constructions on types that we have discussed are analogous to the logical operations "and," "or," "implies," "not," "there exists," and "for all." We also have the unit type $\mathbb{1}$ to represent the proposition $\top$ and the empty type $\varnothing$ to represent the proposition $\bot$. There is one further ingredient from first-order logic that is missing a counterpart in dependent type theory: the logical operation "=."

Given a type A and two terms $x, y : A$ it is sensible to ask whether $x = y$. From the point of view of types as proof-relevant propositions, "$x = y$" should be the name of a type, in fact a dependent type. The formation rule for **identity types** says

$$\frac{\Gamma \vdash A\ \mathtt{type}}{\Gamma, x : A, y : A \vdash x =_A y\ \mathtt{type}}$$

where "$x = y$" is commonly used as an abbreviation for "$x =_A y$" when the type of x and y is clear from context. A term $p : x = y$ of an identity type is called an **identification** of x and y or a **path** from x to y (more about this second term later). Identifications have a rich structure that follows from a very simple characterization of the identity type due to Per Martin-Löf: it is the inductive type family freely generated by the reflexivity terms.

The inductive definition of identity types. We can define identity types as inductive types in either a one-sided or two-sided fashion. The induction rule may be easier to understand from the one-sided point of view, so we present it first.

defn (one-sided identity types)**.** Given a type A and a term $a : A$, the **identity type** of A at a is the inductive family of types $x : A \vdash a =_A x\ \mathtt{type}$ with a single constructor $\mathtt{refl}_a : a =_A a$. The induction principle is postulates that for any type family $x : A, p : a =_A x \vdash P(x,p)\ \mathtt{type}$ there is a function

$$\mathtt{path\text{-}ind}_a : P(a, \mathtt{refl}_a) \to \Pi_{x:A}\Pi_{p:a=_A x} P(x,p)$$

satisfying $\mathtt{path\text{-}ind}_a(q, a, \mathtt{refl}_a) \doteq q$.

This is a very strong induction principle: it says that to prove a predicate $P(x,p)$ depending on any term $x : A$ and any identification $p : a =_A x$ it suffices to assume x is a and p is $\mathtt{refl}_a$ and prove $P(a, \mathtt{refl}_a)$.

More formally, identity types are defined by the following rules:

$$\frac{\Gamma \vdash a : A}{\Gamma, x : A \vdash a =_A x\ \mathtt{type}} \qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash \mathtt{refl}_a : a =_A a}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A, p : a =_A x \vdash P(x,p)\ \mathtt{type}}{\Gamma \vdash \mathtt{path\text{-}ind}_a : P(a, \mathtt{refl}_a) \to \Pi_{x:A}\Pi_{p:a=_A x} P(x,p)} \qquad \frac{\Gamma \vdash a : A \qquad \Gamma, x : A, p : a =_A x \vdash P(x,p)\ \mathtt{type}}{\Gamma \vdash \mathtt{path\text{-}ind}_a(q, a, \mathtt{refl}_a) \doteq q : P(a, \mathtt{refl}_a)}$$

Equally, the identity type can be considered in a two-sided fashion:

defn (two-sided identity types)**.** Given a type A, the **identity type** of A is the inductive family of types $x : A, y : A \vdash x =_A y \ \mathtt{type}$ with a single constructor $x : A \vdash \mathtt{refl}_x : x =_A x$. The induction principle is postulates that for any type family $x : A, y : A, p : x =_A y \vdash P(x, y, p) \ \mathtt{type}$ there is a function

$$\mathtt{path\text{-}ind} : \Pi_{a:A} P(a, a, \mathtt{refl}_a) \to \Pi_{x:A}\Pi_{y:A}\Pi_{p:x=_A y} P(x, y, p)$$

satisfying $\mathtt{path\text{-}ind}(q, a, a, \mathtt{refl}_a) \doteq q$.

In this form, the identity types are defined by the following rules:

$$\frac{\Gamma \vdash A}{\Gamma, x : A, y : A \vdash x =_A y \ \mathtt{type}} \qquad \frac{\Gamma \vdash A}{\Gamma, x : A \vdash \mathtt{refl}_x : x =_A x}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A, y : A, p : x =_A y \vdash P(x, y, p) \ \mathtt{type}}{\Gamma \vdash \mathtt{path\text{-}ind} : \Pi_{a:A} P(a, a, \mathtt{refl}_a) \to \Pi_{x:A}\Pi_{y:A}\Pi_{p:x=_A y} P(x, y, p)} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A, y : A, p : x =_A y \vdash P(x, y, p) \ \mathtt{type}}{\Gamma, a : A \vdash \mathtt{path\text{-}ind}(q, a, a, \mathtt{refl}_a) \doteq q : P(a, a, \mathtt{refl}_a)}$$

These presentations are interderivable.

The groupoid structure on types. Mathematical equality, as traditionally understood, is an equivalence relation: it's reflexive, symmetric, and transitive. But all we've asserted about identity types is that they are inductively generated by the reflexivity terms! As we'll now start to discover, considerable additional structure follows.

Proposition (symmetry)**.** *For any type A, three is an **inverse operation***

$$\mathtt{inv} : \Pi_{x,y:A} x = y \to y = x.$$

Proof. We define $\mathtt{inv}$ by path induction. By the introduction rule for function types it suffices to define $\mathtt{inv}p : y = x$ for $p : x = y$. Consider the type family $x : A, y : A, p : x = y \vdash P(x, y, p) \coloneqq y = x$. By path induction to inhabit $y = x$ it suffices to assume $x = y$ and p is $\mathtt{refl}_x$ in which case we may define $\mathtt{inv}\mathtt{refl}_x \coloneqq \mathtt{refl}_x : x = x$. Thus $\mathtt{inv}$ is

$$\mathtt{path\text{-}ind}(\lambda x, \mathtt{refl} x) : \Pi_{x:A}\Pi_{y:A}\Pi_{x=y} y = x.$$

□

Notation. Write p^{-1} for $\mathtt{inv}(p)$.

Proposition (transitivity)**.** *For any type A, there is a **concatenation** operation*

$$\mathtt{concat} : \Pi_{x,y,z:A} x = y \to y = z \to x = z.$$

Proof. We define $\mathtt{concat}$ by appealing to the path induction principle for identity types. By the introduction rule for dependent function types, to define $\mathtt{concat}$ you may assume given $p : x = y$. The task is then to define $\mathtt{concat}(p) : \Pi_z y = z \to x = z$. For this, consider the type family $x : A, y : A, p : x = y \vdash P(x, y, p)$ where $P(x, y, p) \coloneqq \Pi_{z:A}(y = z) \to (x = z)$. By applying the function $\mathtt{path\text{-}ind}$ to get a term of this type it suffices to assume y is x and p is $\mathtt{refl}_x$. So we need only define $\mathtt{concat}(\mathtt{refl}_x) : \Pi_{z:A} x = z \to x = z$ and we define this to be the identity function $\mathtt{id}_{x=z}$. Thus the function $\mathtt{concat}$ is

$$\mathtt{path\text{-}ind}(\lambda x, \lambda z, \mathtt{id}_{x=z}) : \Pi_{x:A}\Pi_{y:A}\Pi_{p:x=y}\Pi_{z:A} y = z \to x = z,$$

which can be regarded as a function in the type $\Pi_{x,y,z:A} x = y \to y = z \to x = z$ by swapping the order of the arguments p and z. □

Notation. Write $p \cdot q$ for $\mathtt{concat}(p, q)$.

While the elimination rule for identity types is quite strong the corresponding computation rule is relatively weak. It's not strong enough to show that $(p \cdot q) \cdot r$ and $p \cdot (q \cdot r)$ are judgmentally equal for any $p : x = y, q : y = z$, and $r : z = q$. In fact there are countermodels that show that this is false in general. However, since both $(p \cdot q) \cdot r$ and $p \cdot (q \cdot r)$ are terms of type $x = w$ we can ask whether there is an identification between them and it turns out this is always true.

Proposition (associativity)**.** *Given $x, y, z, w : A$ and identifications $p : x = y$, $q : y = z$, and $r : z = w$, there is an associator*

$$\mathtt{assoc}(p, q, r) : (p \cdot q) \cdot r = p \cdot (q \cdot r)$$

Proof. We define $\mathtt{assoc}(p,q,r)$ by path induction.

Consider the type family $x : A, y : A, p : x = y \vdash \Pi_{z:A}\Pi_{q:y=z}\Pi_{w:A}\Pi_{r:z=w}(p \cdot q) \cdot r = p \cdot (q \cdot r)$. To define a term $\mathtt{assoc}(p,q,r)$ in here it suffices to assume y is x and p is $\mathtt{refl}_x$ and define

$$\lambda z.\lambda q.\lambda w.\lambda r.\mathtt{assoc}(\mathtt{refl}_x, q, r) : \Pi_{z:A}\Pi_{q:x=z}\Pi_{w:A}\Pi_{r:z=w}(\mathtt{refl}_x \cdot q) \cdot r = \mathtt{refl}_x \cdot (q \cdot r).$$

By the definition of concatenation, $\mathtt{refl}_x \cdot q \doteq q$ and $\mathtt{refl}_x \cdot (q \cdot r) \doteq q \cdot r$. So we must define

$$\mathtt{assoc}(\mathtt{refl}_x, q, r) : q \cdot r = q \cdot r$$

and we can take this term to be $\mathtt{refl}_{q \cdot r}$. □

Proposition (units). *For any type A, there are left and right* ***unit laws***

$$\lambda x.\lambda y.\lambda p.\mathtt{left\text{-}unit}(p) : \lambda x, y : A\Pi_{p:x=y}\mathtt{refl}_x \cdot p = p \qquad \lambda x.\lambda y.\lambda p.\mathtt{right\text{-}unit}(p) : \Pi_{x,y:A}\Pi_{p:x=y} p \cdot \mathtt{refl}_y = p.$$

Proof. We are asked to define dependent functions that takes $x, y : A$ and $p : x = y$ and produce terms

$$\mathtt{left\text{-}unit}(p) : \mathtt{refl}_x \cdot p = p \qquad \mathtt{right\text{-}unit}(p) : p \cdot \mathtt{refl}_y = p.$$

By path induction, it suffices to assume y is x and p is $\mathtt{refl}_x$, in which case we require terms

$$\mathtt{left\text{-}unit}(\mathtt{refl}_x) : \mathtt{refl}_x \cdot \mathtt{refl}_x = \mathtt{refl}_x \qquad \mathtt{right\text{-}unit}(\mathtt{refl}_x) : \mathtt{refl}_x \cdot \mathtt{refl}_x = \mathtt{refl}_x.$$

By the definition of concatenation $\mathtt{refl}_x \cdot \mathtt{refl}_x \doteq \mathtt{refl}_x$ so we can take $\mathtt{refl}_{\mathtt{refl}_x}$ as both $\mathtt{left\text{-}unit}(\mathtt{refl}_x)$ and $\mathtt{right\text{-}unit}(\mathtt{refl}_x)$. □

Proposition (inverses). *For any type A, there are left and right* ***inverse laws***

$$\lambda x.\lambda y.\lambda p.\mathtt{left\text{-}inv}(p) : \Pi_{x,y:A}\Pi_{p:x=y} p^{-1} \cdot p = \mathtt{refl}_y \qquad \lambda x.\lambda y.\lambda p.\mathtt{right\text{-}inv}(p) : \Pi_{x,y:A}\Pi_{p:x=y} p \cdot p^{-1} = \mathtt{refl}_x.$$

Proof. We are asked to define dependent functions that takes $x, y : A$ and $p : x = y$ and produce terms

$$\mathtt{left\text{-}inv}(p) : p^{-1} \cdot p = \mathtt{refl}_y \qquad \mathtt{right\text{-}inv}(p) : p \cdot p^{-1} = \mathtt{refl}_x.$$

By path induction, it suffices to assume y is x and p is $\mathtt{refl}_x$, in which case we require terms

$$\mathtt{left\text{-}inv}(\mathtt{refl}_x) : \mathtt{refl}_x^{-1} \cdot \mathtt{refl}_x = \mathtt{refl}_x \qquad \mathtt{right\text{-}inv}(\mathtt{refl}_x) : \mathtt{refl}_x \cdot \mathtt{refl}_x^{-1} = \mathtt{refl}_x.$$

By the definitions of concatenation and inverses, again both left-hand and right-hand sides are judgementally equal so we take $\mathtt{left\text{-}inv}(\mathtt{refl}_x)$ and $\mathtt{right\text{-}inv}(\mathtt{refl}_x)$ to be $\mathtt{refl}_{\mathtt{refl}_x}$. □

September 20: More identity types

Types as ∞-groupoids. Martin-Löf's rules for the identity types date from a 1975 paper "An Intuitionistic Theory of Types." In the following two decades, there was a conjecture that went by the name "uniqueness of identity proofs" that for any $x, y : A, p, q : x =_A y$, the type $p =_{x=_A y} q$ is inhabited, meaning that it's possible to construct an identification between p and q. In 1994, Martin Hofmann and Thomas Streicher constructed a model of Martin-Löf's dependent type theory in the category of groupoids that refutes uniqueness of identity proofs.[4]

In the Hofmann-Streicher model, types A correspond to *groupoids* and terms $x, y : A$ correspond to *objects* in the groupoid. An identification $p : x = y$ corresponds to a(n iso)morphism $p : x \to y$ in the groupoid, while an identification between identifications exists if and only if p and q define the same morphism. Since there are groupoids with multiple distinct morphisms between a fixed pair of objects, we see that it is not always the case that $p =_{x=_A y} q$. Following Hofmann-Streicher, it made sense to start viewing types as more akin to groupoids than to sets. The proofs of symmetry and transitivity for identity types are more accurately described as inverses and concatenation operations in a groupoid. As we've seen, these satisfy various associativity, unit, and inverse laws—up to identification at least—as required by a groupoid.

But that last caveat is important. We've shown that for any type A, its identity types $x, y : A \vdash x =_A y\ \mathtt{type}$ give it something like the structure of a groupoid. But for each $x, y : A, x =_A y$ is also a type, so *its* identity types $p, q : x =_A y \vdash p =_{x=_A y} q\ \mathtt{type}$ give $x =_A y$ its own groupoid structure. And the higher identity types, $\alpha, \beta : p =_{x=_A y} q \vdash \alpha = \beta\ \mathtt{type}$ give $p =_{x=_A y} q$ its own groupoid structure and so on. So a modern point of view is that the types in Martin-Löf's dependent type theory should be thought of as *∞-groupoids*.

[4] The technical details of what exactly it means to "construct a model of type theory" are quite elaborate and would be interesting to explore as a final project.

If A is an ∞-groupoid, its terms $x : A$ might be called **points** and its identifications $p : x =_A y$ might be called **paths**. This explains the modern name "path induction" for the induction principle for identity types. These ideas are at the heart of the homotopical interpretation of type theory, about more which later.

The uniqueness of `refl`. The definition of the identity types says that the family of types $a = x$ indexed by $x : A$ is inductively generated by the term $\mathtt{refl}_a : a = a$. It does *not* say that the type $a = a$ is inductively generated by $a : A$. In particular, we cannot apply path induction to prove that $p = \mathtt{refl}_a$ for any $p : a = a$ because in this case neither endpoint of the identity type is free.

There is a sense however in which the reflexivity term is unique:

Proposition. *For any type A and $a : A$, $(a, \mathtt{refl}_a)$ is the unique term of the type $\Sigma_{x:A} a = x$. That is, for any $z : \Sigma_{x:A} a = x$, there is an identification $(a, \mathtt{refl}_a) = z$.*

Proof. We're trying to define a dependent function that takes $z : \Sigma_{x:A} a = x$ and gives a term in the identity type $(a, \mathtt{refl}_a) =_{\Sigma_{x:A} a=x} z$. By Σ-induction it suffices to assume z is a pair (x, p) where $x : A$ and $p : a = x$ and construct an identification $(a, \mathtt{refl}_a) =_{\Sigma_{x:A} a=x} (x, p)$. So now we're trying to define a dependent function that takes $x : A$ and $p : a = x$ and constructs an identification $(a, \mathtt{refl}_a) =_{\Sigma_{x:A} a=x} (x, p)$. By path induction, it suffices to assume x is a and p is $\mathtt{refl}_a$. But now we can use reflexivity to show that $(a, \mathtt{refl}_a) = (a, \mathtt{refl}_a)$. □

In terminology to be introduced later, this result says that the type $\Sigma_{x:A} a = x$ is **contractible** with the term $(a, \mathtt{refl}_a)$ serving as its **center of contraction**.

The action of paths on functions. The structural rules of type theory guarantee that any function (and indeed any construction in type theory) preserve definitional equality. We now show that in addition every function preserves identifications.

Proposition. *Let $f\colon A \to B$. There is an operation that defines the **action on paths** of f*

$$\mathtt{ap}_f : \Pi_{x,y:A}(x = y) \to (f(x) = f(y))$$

that satisfies the coherence conditions

$$\mathtt{ap\text{-}id}_A : \Pi_{x,y:A}\Pi_{p:x=y} p = \mathtt{ap}_{\mathtt{id}_A}(p)$$

$$\mathtt{ap\text{-}comp}(f, g) : \Pi_{x,y:A}\Pi_{p:x=y} \mathtt{ap}_g(\mathtt{ap}_f(p)) = \mathtt{ap}_{g\circ f}(p).$$

Proof. By path induction to define $\mathtt{ap}_f(p) : f(x) = f(y)$ it suffices to assume y is x and p is $\mathtt{refl}_x$. We may then define $\mathtt{ap}_f(\mathtt{refl}_x) :\equiv \mathtt{refl}_{f(x)} : f(x) = f(x)$.

Next to define $\mathtt{ap\text{-}id}_A$ it similarly suffices to suppose y is x and p is $\mathtt{refl}_x$. Since $\mathtt{ap}_{\mathtt{id}_A}(\mathtt{refl}_x) \doteq \mathtt{refl}_x$, we may define $\mathtt{ap\text{-}id}_A(\mathtt{refl}_x) :\equiv \mathtt{refl}_{\mathtt{refl}_x} : \mathtt{refl}_x = \mathtt{refl}_x$.

Finally, to define $\mathtt{ap\text{-}comp}(f, g)$, by path induction we may again assume y is x and p is $\mathtt{refl}_x$. Since both $\mathtt{ap}_g(\mathtt{ap}_f(\mathtt{refl}_x))$ and $\mathtt{ap}_{g\circ f}(\mathtt{refl}_x)$ are defined to be $\mathtt{refl}_{g(f(x))}$ we may define $\mathtt{ap\text{-}comp}(f, g)(\mathtt{refl}_x)$ to be $\mathtt{refl}_{\mathtt{refl}_{g(f(x))}}$. □

If the types A and B are thought of as ∞-groupoids, then $f\colon A \to B$ can be thought of as a functor of ∞-groupoids in a sense hinted at by the following lemma.

Lemma. *For $f\colon A \to B$ there are identifications*

$$\mathtt{ap\text{-}refl}(f, x) : \mathtt{ap}_f(\mathtt{refl}_x) = \mathtt{refl}_{f(x)}$$

$$\mathtt{ap\text{-}inv}(f, p) : \mathtt{ap}_f(p^{-1}) = \mathtt{ap}_f(p)^{-1}$$

$$\mathtt{ap\text{-}concat}(f, p, q) : \mathtt{ap}_f(p \cdot q) = \mathtt{ap}_f(p) \cdot \mathtt{ap}_f(q)$$

for every $p : x = y$ and $q : y = z$.

Proof. For the first coherence, there is a definitional equality $\mathtt{ap}_f(\mathtt{refl}_x) \doteq \mathtt{refl}_{f(x)}$ so we take $\mathtt{ap\text{-}refl}(f, x) :\equiv \mathtt{refl}_{\mathtt{refl}_{f(x)}}$.

We define $\mathtt{ap\text{-}inv}(f, p)$ by path induction on p by defining $\mathtt{ap\text{-}inv}(f, \mathtt{refl}_x) :\equiv \mathtt{refl}_{\mathtt{refl}_{f(x)}}$.

Similarly, we define $\mathtt{ap\text{-}concat}(f, p, q)$ by path induction on p (since concat was defined by path induction on p) by defining $\mathtt{ap\text{-}concat}(f, \mathtt{refl}_x, q)$ to be $\mathtt{refl}_{\mathtt{ap}_f(q)}$. □

Transport. The term $\mathtt{ap}_f$ defines the action of a non-dependent function $f\colon A \to B$ on paths in A. It's natural to ask whether a dependent function $f : \Pi_{z:A} B(z)$ also induces an action on paths. There's a challenge here, though. If $x, y : A$ are terms belonging to the base type, then we can form the type $x =_A y$ to ask whether they are identifiable. But the terms $f(x) : B(x)$ and $f(y) : B(y)$ belong to different types and are not identifiable. But nevertheless if there is path $p : x = y$ identifying y with x intuition suggests there should be some way to compare $f(y)$ to $f(x)$.

To achieve this, we must construct a different sort of action of paths function first. This is called the **transport** function for dependent types $x : A \vdash B(x)$ $\mathtt{type}$ that, given an identification $p : x = y$ in the base type, can be used to transport any term in $B(x)$ to a term in $B(y)$.

Proposition. *For any type family $x : A \vdash B(x)$ $\mathtt{type}$, there is a **transport operation***

$$\mathtt{tr}_B : \Pi_{x,y:A}(x = y) \to (B(x) \to B(y)).$$

Proof. By path induction it suffices to define $\mathtt{tr}_B(\mathtt{refl}_x)) \coloneqq \mathtt{id}_{B(x)}$. □

As an application of transport we can now defined the action on paths of a dependent function.

Proposition. *For any dependent function $f : \Pi_{z:A} B(z)$ and identification $p : x =_A y$ there is a path*

$$\mathtt{apd}_f(p) : \mathtt{tr}_B(p, f(x)) =_{B(y)} f(y).$$

Proof. The function

$$\lambda x.\lambda y.\lambda p.\mathtt{apd}_f(p) : \Pi_{x,y:A}\Pi_{p:x=y}\mathtt{tr}_B(p, f(x)) =_{B(y)} f(y)$$

may be defined by path induction on p. It suffices to construct a path

$$\lambda x.\mathtt{apd}_f(\mathtt{refl}_x) : \Pi_{x:A}\mathtt{tr}_B(\mathtt{refl}_x, f(x)) =_{B(x)} f(x).$$

Since $\mathtt{tr}_B(\mathtt{refl}_x), f(x)) \doteq f(x)$ we may defined $\mathtt{apd}_f(\mathtt{refl}_x) \coloneqq \mathtt{refl}_{f(x)}$. □

The laws of addition on $\mathbb{N}$. Recall that we defined the addition of natural numbers in such a way that

$$m + 0 \doteq m \qquad m + \mathtt{succ}_{\mathbb{N}}(n) \doteq \mathtt{succ}_{\mathbb{N}}(m + n)$$

by induction on the second variable. With this definition, these are the only definitional equalities. However, it is possible to produce identifications proving the other commutative monoid axioms.

Lemma. *For any $n : \mathbb{N}$ there are identifications*

$$\mathtt{left\text{-}unit\text{-}law\text{-}add}_{\mathbb{N}}(n) : 0 + n = n \qquad \mathtt{right\text{-}unit\text{-}law\text{-}add}_{\mathbb{N}}(n) : n + 0 = n.$$

Proof. The second of these can be taken to be $\mathtt{refl}_n$ but the first is more complicated. We define $\mathtt{left\text{-}unit\text{-}law\text{-}add}_{\mathbb{N}}(n)$ by induction on $n : \mathbb{N}$. When $n = 0$, $0 + 0 = 0$ holds by reflexivity.

Our final goal is to show $0 + \mathtt{succ}_{\mathbb{N}}(n) = \mathtt{succ}_{\mathbb{N}}(n)$, for which it suffices to construct an identification

$$\mathtt{succ}_{\mathbb{N}}(0 + n) = \mathtt{succ}_{\mathbb{N}}(n)$$

by the definition of addition. We may assume we have an identification $p : 0 + n = n$. Thus, we can use the action on paths of $\mathtt{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ to obtain a term $\mathtt{ap}_{\mathtt{succ}_{\mathbb{N}}}(p) : \mathtt{succ}_{\mathbb{N}}(0 + n) = \mathtt{succ}_{\mathbb{N}}(n)$. □

Proposition. *For any $m, n : \mathbb{N}$ there are identifications*

$$\mathtt{left\text{-}successor\text{-}law\text{-}add}_{\mathbb{N}}(m, n) : \mathtt{succ}_{\mathbb{N}}(m) + n = \mathtt{succ}_{\mathbb{N}}(m + n)$$
$$\mathtt{right\text{-}sucessor\text{-}law\text{-}add}_{\mathbb{N}}(m, n) = m + \mathtt{succ}_{\mathbb{N}}(n) = \mathtt{succ}_{\mathbb{N}}(m + n)$$

Proof. Again the second identification holds judgmentally so we define

$$\mathtt{right\text{-}sucessor\text{-}law\text{-}add}_{\mathbb{N}}(m, n) \coloneqq \mathtt{refl}_{\mathtt{succ}_{\mathbb{N}}(m+n)}.$$

We construct the former using induction on $n \in \mathbb{N}$. The base case $\mathtt{succ}_{\mathbb{N}}(m) + 0 = \mathtt{succ}_{\mathbb{N}}(m + 0)$ holds by $\mathtt{refl}_{\mathtt{succ}_{\mathbb{N}}(m)}$. For the inductive step we assume we have an identification $p : \mathtt{succ}_{\mathbb{N}}(m) + n = \mathtt{succ}_{\mathbb{N}}(m + n)$. Our goal is to show that $\mathtt{succ}_{\mathbb{N}}(m) + \mathtt{succ}_{\mathbb{N}}(n) = \mathtt{succ}_{\mathbb{N}}(m + \mathtt{succ}_{\mathbb{N}}(n))$. By action of paths of $\mathtt{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ we obtain a term

$$\mathtt{ap}_{\mathtt{succ}_{\mathbb{N}}}(p) : \mathtt{succ}_{\mathbb{N}}(\mathtt{succ}_{\mathbb{N}}(m) + n) = \mathtt{succ}_{\mathbb{N}}(\mathtt{succ}_{\mathbb{N}}(m + n))$$

but here the left hand side is judgmentally equal to $\mathtt{succ}_{\mathbb{N}}(m) + \mathtt{succ}_{\mathbb{N}}(n)$ while the right hand side is judgmentally equal to $\mathtt{succ}_{\mathbb{N}}(m + \mathtt{succ}_{\mathbb{N}}(n))$. □

Proposition (associativity)**.** *For all* $k, m, n : \mathbb{N}$,

$$\texttt{associative-add}_{\mathbb{N}}(k, m, n) : (m + n) + k = m + (n + k).$$

Proof. We construct $\texttt{associative-add}_{\mathbb{N}}(k, m, n)$ by induction on n. In the base case we have

$$(k + m) + 0 \doteq k + m \doteq k + (m + 0),$$

so we define $\texttt{associative-add}_{\mathbb{N}}(k, m, 0) \coloneqq \texttt{refl}_{m+n}$.

For the inductive step let $p : (k + m) + n = k + (m + n)$. We then have

$$\texttt{ap}_{\texttt{succ}_{\mathbb{N}}}(p) : \texttt{succ}_{\mathbb{N}}((k + m) + n) = \texttt{succ}_{\mathbb{N}}(k + (m + n)).$$

We have $\texttt{succ}_{\mathbb{N}}((k+m)+n) \doteq (k+m)+\texttt{succ}_{\mathbb{N}}(n)$ and $\texttt{succ}_{\mathbb{N}}(k+(m+n)) \doteq k+\texttt{succ}_{\mathbb{N}}(m+n) \doteq k+(m+\texttt{succ}_{\mathbb{N}}(n))$ so this term is the term we wanted. □

Proposition (commutativity)**.** *For all* $m, n : \mathbb{N}$,

$$\texttt{commutative-add}_{\mathbb{N}}(m, n) : m + n = n + m.$$

Proof. By induction on m we have to show $0 + n = n + 0$, which holds by the unit laws for n. Then we may assume $p : m + n = n + m$ and must show $\texttt{succ}_{\mathbb{N}}(m) + n = n + \texttt{succ}_{\mathbb{N}}(m)$. We have

$$\texttt{ap}_{\texttt{succ}_{\mathbb{N}}}(p) : \texttt{succ}_{\mathbb{N}}(m + n) = \texttt{succ}_{\mathbb{N}}(n + m).$$

We then concatenate this path with the paths $\texttt{left-successor-law-add}_{\mathbb{N}}(m, n)$ and $\texttt{right-successor-law-add}_{\mathbb{N}}(n, m)$ to obtain the identification we want. □

September 22: Universes

Recall that in Martin-Löf's dependent type theory, $\mathbb{N}$ was defined as the inductive type freely generated by a term $0_{\mathbb{N}} : \mathbb{N}$ and a function $\texttt{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$. The corresponding induction principle gives a strengthened version of the Dedekind-Peano principle of mathematical induction, but two of the traditional axioms—namely that $0_{\mathbb{N}}$ is not a successor and $\texttt{succ}_{\mathbb{N}}$ is injective—are missing. Using our type forming operations, we can define the types that assert those axioms:

$$\Pi_{n:\mathbb{N}}(n = 0_{\mathbb{N}}) \to \varnothing \qquad \Pi_{n,m:\mathbb{N}}(\texttt{succ}_{\mathbb{N}}(n) = \texttt{succ}_{\mathbb{N}}(m)) \to (n = m)$$

but we don't yet have the tools needed to construct terms in those types. Type theoretic *universes* will enable us to construct terms in these types and prove many other things besides.

Informally, a universe $\mathcal{U}$ can be thought of as a "type whose terms are types." More precisely, a universe is a type $\mathcal{U}$ together with a type family $X : \mathcal{U} \vdash \mathcal{T}(X)$ called the *universal type family*. We think of the term X as an *encoding* of the type $\mathcal{T}(X)$ though its common to conflate these notions notationally, writing "X" for both the encoding and the type.

Universes are assumed to be closed under all the type constructors in a sense to be made precise below. To avoid a famous inconsistency, however, we do not assume that the universe is contained in itself. One way to think about this is that $\mathcal{U}$ is the type of "small" types, but $\mathcal{U}$ itself is not "small."

In the presence of a universe $\mathcal{U}$, a family of small types $x : A \vdash B(x)\ \texttt{type}$ over a type A can be encoded by a function $B\colon A \to \mathcal{U}$ defined by sending the term x to the encoding of the type $B(x)$.[5] In particular, if A is an inductive type, freely generated by some finite list of constructors, then *type families* over A—not just dependent functions over A—can be defined inductively by specifying types for each of the constructors. We will see examples of this soon.

Type theoretic universes.

defn. A **universe** is a type $\mathcal{U}$ in the empty context equipped with a type family $X : \mathcal{U} \vdash \mathcal{T}(X)\ \texttt{type}$ over $\mathcal{U}$ called the **universal family of types** that is closed under the type forming operations in the sense that it is equipped with the following structure:

(i) $\mathcal{U}$ contains terms $\check{\varnothing}$, $\check{\mathbb{1}}$, $\check{\mathbb{N}}$ that satisfy the judgmental equalities

$$\mathcal{T}(\check{\varnothing}) \doteq \varnothing, \quad \mathcal{T}(\check{\mathbb{1}}) \doteq \mathbb{1}, \quad \mathcal{T}(\check{\mathbb{N}}) \doteq \mathbb{N}.$$

[5] This is already how we have been defining type families in `agda`.

(ii) $\mathcal{U}$ is closed under coproducts in the sense that it comes equipped with a function

$$\check{+}\colon \mathcal{U} \to \mathcal{U} \to \mathcal{U}$$

that satisfies $\mathcal{T}(X \check{+} Y) \doteq \mathcal{T}(X) + \mathcal{T}(Y)$.

(iii) $\mathcal{U}$ is closed under Π-types in the sense that it comes equipped with a function

$$\check{\Pi}\colon \Pi_{X:\mathcal{U}}(\mathcal{T}(X) \to \mathcal{U}) \to \mathcal{U}$$

satisfying

$$\mathcal{T}(\check{\Pi}(X, P)) \doteq \Pi_{x:\mathcal{T}(X)}\mathcal{T}(P(x))$$

for all $X : \mathcal{U}$ and $P\colon \mathcal{T}(X) \to \mathcal{U}$.

(iv) $\mathcal{U}$ is closed under Σ-types in the sense that it comes equipped with a function

$$\check{\Sigma}\colon \Pi_{X:\mathcal{U}}(\mathcal{T}(X) \to \mathcal{U}) \to \mathcal{U}$$

satisfying

$$\mathcal{T}(\check{\Sigma}(X, P)) \doteq \Sigma_{x:\mathcal{T}(X)}\mathcal{T}(P(x))$$

for all $X : \mathcal{U}$ and $P\colon \mathcal{T}(X) \to \mathcal{U}$.

(v) $\mathcal{U}$ is closed under identity types in the sense that it comes equipped with a function

$$\check{\mathrm{Id}} : \Pi_{X:\mathcal{U}}\mathcal{T}(X) \to \mathcal{T}(X) \to \mathcal{U}$$

satisfying

$$\mathcal{T}(\check{\mathrm{Id}}(X, x, y)) \doteq (x = y)$$

for all $X : \mathcal{U}$ and $x, y : \mathcal{T}(X)$.

defn. Given a universe $\mathcal{U}$, we say a type A in context Γ is **small** if it occurs in the universe: i.e., if it comes equipped with a term $\check{A} : \mathcal{U}$ in context Γ for which the judgment

$$\Gamma \vdash \mathcal{T}(\check{A}) \doteq A \ \mathtt{type}$$

holds.

When A is a small type, it's common to write A for both $\check{A}$ and $\mathcal{T}(A)$. So by $A : \mathcal{U}$ we mean that A is a small type.

Assuming enough universes. Most of the time it's sufficient to assume just one universe $\mathcal{U}$. But on occasion, it is useful to assume that $\mathcal{U}$ itself is a type in some universe.

Postulate. We assume that there are **enough universes**, i.e., that for every finite list of types in context

$$\Gamma_1 \vdash A_1 \ \mathtt{type} \quad \ldots \quad \Gamma_n \vdash A_n \ \mathtt{type}$$

there is a universe $\mathcal{U}$ that contains each A_i in the sense that $\mathcal{U}$ has terms

$$\Gamma_i \vdash \check{A}_i : \mathcal{U}$$

for which $\Gamma_i \vdash \mathcal{T}(\check{A}_i) \doteq A_i \ \mathtt{type}$ holds.

With this assumption it's rarely necessary to work with more than one universe at the same time.

As a consequence of our postulate that there exist enough universes, we obtain specific universes:[6]

defn. The **base universe** $\mathcal{U}_0$ is obtained by applying the postulate to the empty list of types in context.

defn. The **successor universe** of any universe $\mathcal{U}$ is the universe $\mathcal{U}^+$ obtained from the finite list

$$\vdash \mathcal{U} \ \mathtt{type} \quad X : \mathcal{U} \vdash \mathcal{T}(X) \ \mathtt{type}$$

Thus the successor universe contains both $\mathcal{U}$ and any type in $\mathcal{U}$.

defn. The **join** of two universes $\mathcal{U}$ and $\mathcal{V}$ is the universe $\mathcal{U} \sqcup \mathcal{V}$ obtained by applying the postulate to the type families

$$X : \mathcal{U} \vdash \mathcal{T}_{\mathcal{U}}(X) \ \mathtt{type} \qquad Y : \mathcal{V} \vdash \mathcal{T}_{\mathcal{V}}(Y) \ \mathtt{type}$$

[6] In `agda`, this structure is formalized in the file Agda.Primitive.

Observational equality on $\mathbb{N}$. To illustrate what universes are for, we define a type family $m : \mathbb{N}, n : \mathbb{N} \vdash \mathrm{Eq}_{\mathbb{N}}(m,n)\ \mathtt{type}$ that we call **observational equality** on $\mathbb{N}$. Because type families can now be thought of as functions $\mathrm{Eq}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} \to \mathcal{U}$ we can use the induction principle of $\mathbb{N}$ to define this type family. We'll then prove that $\mathrm{Eq}_{\mathbb{N}}$ is **logically equivalent** to the identity type family; in fact, we'll later see that these types are **equivalent**, once we know what that means. The advantage of the type family $\mathrm{Eq}_{\mathbb{N}}$ is that it's characterized more explicitly, so this will help us prove theorems about the identity type family over the natural numbers.

defn. We define **observational equality** of $\mathbb{N}$ as the type family $\mathrm{Eq}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} \to \mathcal{U}$ satisfying

$$\mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) \doteq \mathbb{1} \quad \mathrm{Eq}_{\mathbb{N}}(\mathtt{succ}_{\mathbb{N}}(n), 0_{\mathbb{N}}) \doteq \varnothing \quad \mathrm{Eq}_{\mathbb{N}}(0, \mathtt{succ}_{\mathbb{N}}(n)) \doteq \varnothing \quad \mathrm{Eq}_{\mathbb{N}}(\mathtt{succ}_{\mathbb{N}}(m), \mathtt{succ}_{\mathbb{N}}(n)) \doteq \mathrm{Eq}_{\mathbb{N}}(m,n).$$

Lemma. *Observational equality on $\mathbb{N}$ is reflexive:*

$$\mathtt{refl} - \mathrm{Eq}_{\mathbb{N}} : \prod_{n:\mathbb{N}} \mathrm{Eq}_{\mathbb{N}}(n,n).$$

Proof. We define $\mathtt{refl} - \mathrm{Eq}_{\mathbb{N}}$ by induction by $\mathtt{refl} - \mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}) :\equiv \star$ and $\mathtt{refl} - \mathrm{Eq}_{\mathbb{N}}(\mathtt{succ}_{\mathbb{N}}(n)) :\equiv \mathtt{refl} - \mathrm{Eq}_{\mathbb{N}}(n)$. □

Proposition. *For any $m, n : \mathbb{N}$, the types $\mathrm{Eq}_{\mathbb{N}}(m,n)$ and $(m = n)$ are **logically equivalent**: that is there are functions*

$$(m = n) \to \mathrm{Eq}_{\mathbb{N}}(m,n) \quad \textit{and} \quad \mathrm{Eq}_{\mathbb{N}}(m,n) \to (m = n).$$

Proof. By path induction, there is a function $\mathtt{id\text{-}to\text{-}eq} : \prod_{m,n:\mathbb{N}}(m = n) \to \mathrm{Eq}_{\mathbb{N}}(m,n)$ defined by $\mathtt{id\text{-}to\text{-}eq}(n, \mathtt{refl}_n) :\equiv \mathtt{refl} - \mathrm{Eq}_{\mathbb{N}}(n)$.

For the converse, we define a function $\mathtt{eq\text{-}to\text{-}id} : \prod_{m,n:\mathbb{N}} \mathrm{Eq}_{\mathbb{N}}(m,n) \to (m = n)$ by induction on m and n. We define $\mathtt{eq\text{-}to\text{-}id}(0_{\mathbb{N}}, 0_{\mathbb{N}}) : \mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) \to (0_{\mathbb{N}} = 0_{\mathbb{N}})$, by induction on $\mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) \doteq \mathbb{1}$ to be the function that sends $\star : \mathbb{1}$ to $\mathtt{refl}_{0_{\mathbb{N}}} : 0_{\mathbb{N}} = 0_{\mathbb{N}}$. We define the functions $\mathtt{eq\text{-}to\text{-}id}(\mathtt{succ}_{\mathbb{N}}(n), 0_{\mathbb{N}})$ and $\mathtt{eq\text{-}to\text{-}id}(0_{\mathbb{N}}, \mathtt{succ}_{\mathbb{N}}(n))$ using `ex-falso`, since both of these are maps out of the empty type. Finally, to define $\mathtt{eq\text{-}to\text{-}id}(\mathtt{succ}_{\mathbb{N}}(m), \mathtt{succ}_{\mathbb{N}}(n))$ we may use a function $f : \mathrm{Eq}_{\mathbb{N}}(m,n) \to (m = n)$, in which case, $\mathtt{eq\text{-}to\text{-}id}(\mathtt{succ}_{\mathbb{N}}(m), \mathtt{succ}_{\mathbb{N}}(n))$ is defined to be the composite function

$$\mathrm{Eq}_{\mathbb{N}}(\mathtt{succ}_{\mathbb{N}}(m), \mathtt{succ}_{\mathbb{N}}(n)) \xrightarrow{\mathtt{id}} \mathrm{Eq}(m,n) \xrightarrow{f} (m = n) \xrightarrow{\mathtt{ap}_{\mathtt{succ}_{\mathbb{N}}}} (\mathtt{succ}_{\mathbb{N}}(m) = \mathtt{succ}_{\mathbb{N}}(n)).$$

□

Notation. For types A and B, we write $A \leftrightarrow B$ as an abbreviation for the type

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

Thus the logical equivalence defines a term in the type

$$\prod_{m,n:\mathbb{N}} \mathrm{Eq}_{\mathbb{N}}(m,n) \leftrightarrow (m = n).$$

Peano's axioms.

Theorem. *For any $m, n : \mathbb{N}$ we have*

$$(m = n) \leftrightarrow (\mathtt{succ}_{\mathbb{N}}(m) = \mathtt{succ}_{\mathbb{N}}(n))$$

Proof. The action of paths of the successor function proves the forwards implication

$$\mathtt{ap}_{\mathtt{succ}_{\mathbb{N}}} : (m = n) \to (\mathtt{succ}_{\mathbb{N}}(m) = \mathtt{succ}_{\mathbb{N}}(n))$$

The direction of interest is the converse which proves that successor is injective.

Using the logical equivalences $(m = n) \leftrightarrow \mathrm{Eq}_{\mathbb{N}}(m,n)$ we define the reverse implication to be the composite

$$(\mathtt{succ}_{\mathbb{N}}(m) = \mathtt{succ}_{\mathbb{N}}(n)) \xrightarrow{\mathtt{id\text{-}to\text{-}eq}(\mathtt{succ}_{\mathbb{N}}(m), \mathtt{succ}_{\mathbb{N}}(n))} \mathrm{Eq}_{\mathbb{N}}(\mathtt{succ}_{\mathbb{N}}(m), \mathtt{succ}_{\mathbb{N}}(n)) \xrightarrow{\mathtt{id}} \mathrm{Eq}_{\mathbb{N}}(m,n) \xrightarrow{\mathtt{eq\text{-}to\text{-}id}(m,n)} (m = n).$$

□

Theorem. *For any $n : \mathbb{N}$, $\neg(0_{\mathbb{N}} =_{\mathbb{N}} (n))$.*

Proof. We have a family of maps

$$\lambda n, \texttt{id-to-eq}(0_{\mathbb{N}}, n) : \Pi_{n:\mathbb{N}}(0_{\mathbb{N}} = n) \to \mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, n).$$

Since $\mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, \mathtt{succ}_{\mathbb{N}}(n)) \doteq \varnothing$ we have

$$\texttt{id-to-eq}(0_{\mathbb{N}}, \mathtt{succ}_{\mathbb{N}}(n)) : (0_{\mathbb{N}} = \mathtt{succ}_{\mathbb{N}}(n)) \to \varnothing$$

which is precisely the claim. □

September 27: Modular arithmetic

Having fully described Martin-Löf's dependent type theory, we may now start developing some mathematics in it. The fundamental idea used to develop mathematics is something we've already previewed: the Curry-Howard interpretation.

The Curry-Howard interpretation. The Curry-Howard interpretation is an interpretation of logic into type theory. In type theory, there is no separation between the logical framework and the general theory of collections of mathematical objects the way there is in the more traditional setup with Zermelo-Fraenkel set theory, which is postulated by axioms in first order logic. The idea is that propositions may be expressed as types with proofs of those propositions expressed as terms in those types. For example:

defn. We say that a natural number d divides a natural number n if there is a term in the type

$$d \mid n \coloneqq \Sigma_{k:\mathbb{N}} d \cdot k = n$$

defined using the multiplication $\cdot$ on $\mathbb{N}$, the identity type of $\mathbb{N}$, and the dependent sum of the type family $k : \mathbb{N} \vdash d \cdot k = n$ `type` .

Just as existential quantification ($\exists$) is expressed using Σ-types, universal quantification ($\forall$) is expressed using Π-types. For example, the type

$$\Pi_{n:\mathbb{N}} 1 \mid n$$

asserts that every natural number is divisible by 1. The term

$$\lambda n.(n, \texttt{left-unit}(n)) : \Pi_{n:\mathbb{N}} 1 \mid n$$

proves this result.

Proposition. *Let $d, m, n : \mathbb{N}$. If d divides any two of m, n, and $m + n$, then d divides the third.*

Proof. We prove only that if $d \mid m$ and $d \mid n$ then $d \mid m + n$. By hypothesis we have terms:

$$H : \Sigma_{k:\mathbb{N}} d \cdot k = m \quad \text{and} \quad K : \Sigma_{k:\mathbb{N}} d \cdot k = n.$$

By Σ-induction, we may assume that H is given by a pair $(h : \mathbb{N}, p : d \cdot h = m)$ and K is given by a pair $(k : \mathbb{N}, q : d \cdot k = n)$. To get a term in $\Sigma_{x:\mathbb{N}} d \cdot x = m + n$ we may use $x \coloneqq h + k$. Our goal is then to define an identification $d \cdot (h + k) = m + n$ which we obtain as a concatenation

$$d \cdot (h + k) \overset{\mathtt{dist}}{=\!=} d \cdot h + d \cdot k \overset{\mathtt{ap}_{+d \cdot k}(p)}{=\!=} m + d \cdot k \overset{\mathtt{ap}_{m+}(q)}{=\!=} m + n$$

□

We have observed many similarities between the rules of various type constructors and tautologies from logic. For instance, the elimination rule for the non-dependent function type supplies a function

$$\texttt{modus-ponens} : A \times (A \to B) \to B.$$

One important difference is that general types may contain multiple terms that cannot be identified: i.e., for which it is possible to prove that $x =_A y \to \varnothing$. Later we'll study the following predicate on types:

$$\texttt{is-prop}(A) \coloneqq \Pi_{x,y:A} x =_A y$$

which asserts that *if* A has multiple terms (which it may not) those terms can always be identified. This will be the $n = -1$ level of a hierarchy of n-types for $n \geqslant -2$.

The congruence relations on $\mathbb{N}$. The family of identity types can be understood as a type-valued binary relation on a type.

defn. For a type A, a **typal binary relation** on A is a family of types $x, y : A \vdash R(x,y)$ `type` . A binary relation R is

- **reflexive** if it comes with a term $\rho : \prod_{x:A} R(x,x)$,
- **symmetric** if it comes with a term $\sigma : \prod_{x,y:A} R(x,y) \to R(y,x)$,
- **transitive** if it comes with a term $\tau : \prod_{x,y,z:A} R(x,y) \to R(y,z) \to R(x,z)$

A **typal equivalence relation** on A is a reflexive, symmetric, and transitive, typal binary relation.

For instance, for each $k : \mathbb{N}$ we can define the relation of congruence modulo k by defining a type

$$x \equiv y \mod k$$

for each $x, y : \mathbb{N}$ comprised of proofs that x is equivalent to y modulo k. Following Gauss, we say that x is equivalent to y mod k if k divides the symmetric difference $\mathtt{dist}_{\mathbb{N}}(x,y)$ defined recursively by

$$\mathtt{dist}_{\mathbb{N}}(0,0) \coloneqq 0 \quad \mathtt{dist}_{\mathbb{N}}(0,y+1) \coloneqq y+1 \quad \mathtt{dist}_{\mathbb{N}}(x+1,0) \coloneqq x+1, \quad \mathtt{dist}_{\mathbb{N}}(x+1,y+1) \coloneqq \mathtt{dist}_{\mathbb{N}}(x,y).$$

defn. For $k, x, y : \mathbb{N}$ define

$$x \equiv y \mod k \coloneqq k \mid \mathtt{dist}_{\mathbb{N}}(x,y).$$

Note this defines the *type* $x \equiv y \mod k$. A term is then a pair comprised of an $\ell : \mathbb{N}$ together with an identification $k \cdot \ell = \mathtt{dist}_{\mathbb{N}}(x,y)$.

We leave the following to the course text:

Proposition. *For each k, the typal relation $\equiv \mod k$ is an equivalence relation.*

There are other important relations on $\mathbb{N}$ that are not-equivalence relations.

defn. The binary relation $\leqslant$ on $\mathbb{N}$ is defined by induction by

$$0 \leqslant 0 \coloneqq \mathbb{1} \quad 0 \leqslant n+1 \coloneqq \mathbb{1} \quad n+1 \leqslant 0 \coloneqq \varnothing \quad m+1 \leqslant n+1 \coloneqq m \leqslant n.$$

Similarly, the binary relation $<$ is defined by

$$0 < 0 \coloneqq \varnothing \quad 0 < n+1 \coloneqq \mathbb{1} \quad n+1 < 0 \coloneqq \varnothing \quad m+1 < n+1 \coloneqq m < n.$$

The standard finite types. The standard finite sets are classically defined as the sets $\{n \in \mathbb{N} \mid n < k\}$, so how do we interpret a subset $\{x \in A \mid P(x)\}$ characterized by a predicate in type theory?

In the Curry-Howard interpretation, the predicate $P(x)$ is interpreted as a type family and the type of terms x in A for which $P(x)$ is true is interpreted by the Σ-type $\Sigma_{x:A} P(x)$. Note for a general type family $P(x)$ it won't necessarily be the case that the map $\mathtt{pr}_1 : \Sigma_{x:A} P(x) \to A$ is a monomorphism[7] so this construction operates a bit differently than in set theory.

Through this mechanism it is possible to define the classical finite sets as

$$\text{Classical-Fin}_k \coloneqq \Sigma_{n:\mathbb{N}} n < k$$

though the standard definition is as follows:

defn. We define the type family Fin of **standard finite types** inductively (using the induction principle of $\mathbb{N}$ and the universe $\mathcal{U}$) as follows:

$$\text{Fin}_0 \coloneqq \varnothing, \quad \text{Fin}_{k+1} \coloneqq \text{Fin}_k + \mathbb{1}.$$

Write i for $\mathtt{inl} : \text{Fin}_k \to \text{Fin}_{k+1}$ and $\star$ for the point $\mathtt{inr}(\star) : \text{Fin}_{k+1}$.

By induction we can define functions $\iota_k : \text{Fin}_k \to \mathbb{N}$ for each k. When $k = 0$ there is nothing to show. To define $\iota_{k+1} : \text{Fin}_{k+1} \to \mathbb{N}$ we can use ι_k and define $\iota_{k+1}(i(x)) \coloneqq \iota_k(x)$ and $\iota_{k+1}(\star) \coloneqq k$.

[7] Though this will be the case if each type $P(x)$ is a proposition in the sense alluded to above.

The natural numbers modulo $k+1$. Given an equivalence relation $\sim$ on a set A the quotient $A_{/\sim}$ comes equipped with a quotient map $q\colon A \to A_{/\sim}$ that satisfies two important properties:

(i) q is **effective**: $q(x) = q(y)$ if and only if $x \sim y$
(ii) q is surjective: for all $[z] \in A_{/\sim}$ there is some $z \in A$ so that $q(z) = [z]$.

Both properties can be expressed in type theory, though there are some subtleties.

defn. In the context of types A and B there is a type family is-surj$\colon (A \to B) \to \mathcal{U}$ defined by

$$\text{is-surj}(f) \coloneqq \Pi_{b:B}\Sigma_{a:A} f(a) =_B b.$$

The subtlety is that this really defines a *split* notion of surjectivity. A term p in is-surj(f) defines a function that for each term $b : B$ produces a term of $\Sigma_{a:A} f(a) =_B b$. By compositing p with $\mathtt{pr}_1 : \Sigma_{a:A} f(a) =_B b \to A$, we obtain a function $s\colon B \to A$. By composing p with $\mathtt{pr}_2 : \Sigma_{a:A} f(s(b)) =_B b$ we also obtain a proof that s is a **section** of f. Thus surjective functions in homotopy type theory are really **split** surjective functions.

Our next challenge is to define the quotient maps $[-]_k\colon \mathbb{N} \to \text{Fin}_k$ that compute the remainder modulo k. Our strategy will be to define this function by induction on $n : \mathbb{N}$. The idea is that the term $0_{\mathbb{N}} : \mathbb{N}$ should get sent to some 0 while successors in $\mathbb{N}$ should be sent to successors in Fin_k, taken in the cyclic order. We define these auxiliary structures first.

defn. We define $\mathtt{zero}_k : \text{Fin}_{k+1}$ recursively by

$$\mathtt{zero}_0 \coloneqq \star \quad \mathtt{zero}_{k+1} \coloneqq i(\mathtt{zero}_k).$$

We then define $\mathtt{skip\text{-}zero}_k : \text{Fin}_k \to \text{Fin}_{k+1}$ recursively by

$$\mathtt{skip\text{-}zero}_{k+1}(i(x)) \coloneqq i(\mathtt{skip\text{-}zero}_k(x)) \quad \mathtt{skip\text{-}zero}_{k+1}(\star) \coloneqq \star.$$

Finally, we define $\mathtt{succ}_k : \text{Fin}_k \to \text{Fin}_k$ recursively by

$$\mathtt{succ}_{k+1}(i(x)) \coloneqq \mathtt{skip\text{-}zero}_k(x) \qquad \mathtt{succ}_{k+1}(\star) \coloneqq \mathtt{zero}_k.$$

defn. For any $k : \mathbb{N}$ define $[-]_{k+1}\colon \mathbb{N} \to \text{Fin}_{k+1}$ by

$$[0]_{k+1} \coloneqq 0 \quad \text{and} \quad [n+1]_{k+1} \coloneqq \mathtt{succ}_{k+1}[n]_{k+1}.$$

The text goes on to show that

- $n \equiv \iota[n]_k \mod k$ for all n and k,
- $[n]_k = [m]_k$ if and only of $n \equiv m \mod k$,
- and the map $[-]_k\colon \mathbb{N} \to \text{Fin}_k$ is split surjective.

Then it is possible to use this quotient map to define the cyclic group structure on Fin_k.

September 29: Decidability in elementary number theory

In constructive mathematics it is not possible to prove the law of excluded middle: namely that $P \vee \neg P$ for an arbitrary proposition P. Similarly in type theory, it is not possible to construct a term of type $A + \neg A$ for arbitrary A. But certain types do come with such terms.

Decidability.

defn. A type A is **decidable** if it comes equipped with an element of type

$$\text{is-decidable}(A) \coloneqq A + \neg A.$$

A type family $P\colon A \to \mathcal{U}$ is **decidable** if $P(a)$ is decidable for every $a : A$.

ex. The primary way to show that A is decidable is either to provide a term $a : A$ or provide a function $na : A \to \varnothing$. In particular $\mathbb{1}$ is decidable since we have $\mathtt{inl}(\star)$: is-decidable$(\mathbb{1})$. Similarly $\varnothing$ is decidable since we have $\mathtt{inr}(\mathtt{id})$: is-decidable$(\varnothing)$.

ex. Since the type families $n, m : \mathbb{N} \vdash n \leqslant m$ `type` and $n, m : \mathbb{N} \vdash n < m$ `type` were defined by induction from the types $\varnothing$ and $\mathbb{1}$, it follows that these type families are decidable.

Remark. If A and B are decidable then so are $A + B$, $A \times B$, and $A \to B$. Proofs use case analysis over the coproduct types $A + \neg A$ and $B + \neg B$.

defn. A type A **has decidable equality** if the identity type $x =_A y$ is decidable for every $x, y : A$. Thus

$$\text{has-decidable-eq}(A) :\equiv \Pi_{x,y:A}\text{is-decidable}(x =_A y).$$

Lemma. *Suppose A and B are types so that $A \leftrightarrow B$. Then A is decidable if and only if B is decidable.*

Proof. A proof of $A \leftrightarrow B$ supplies functions $f\colon A \to B$ and $g\colon B \to A$. Using the contrapositive function we obtain $\texttt{contrapositive}(f) : \neg B \to \neg A$ and $\texttt{contrapositive}(g)\colon \neg A \to \neg B$. We therefore have functions

$$f + \texttt{contrapositive}(g)\colon A + \neg A \to B + \neg B \qquad g + \texttt{contrapositive}(f) : B + \neg B \to A + \neg A,$$

proving the logical equivalence of is-decidable(A) and is-decidable(B). In particular, if either type is inhabited, both must be. □

Corollary. $\mathbb{N}$ *has decidable equality.*

Proof. We have shown that the identity types of $\mathbb{N}$ are logically equivalent to the observational equality types, which were defined to be $\mathbb{1}$ or $\emptyset$. As both types are decidable, the identity types of $\mathbb{N}$ must be as well. □

Remark. We will prove later that if a type has decidable equality then it must be a **set** in a technical sense to be introduced. Even so, not all sets have decidable equality unless one assumes that the law of excluded middle is true for all propositions.

Case analysis. Suppose you'd like to define a function by case analysis such as $\texttt{collatz} : \mathbb{N} \to \mathbb{N}$

$$\texttt{collatz}(n) :\equiv \begin{cases} n/2 & n \text{ is odd} \\ 2n+1 & n \text{ is even} \end{cases}$$

To justify this sort of case analysis we use a term

$$d : \Pi_{n:\mathbb{N}}\text{is-decidable}(2 \mid n)$$

whose construction we skipped. Note that $2 \mid n$ and $\neg(2 \mid n)$ cannot both hold because if so we could evaluate the function $\texttt{odd}(n) : \neg(2 \mid n)$ at the term $\texttt{even}(n) : 2 \mid n$ to get a contradiction. So this d can be thought of as a proof that for all $n : \mathbb{N}$, n is odd or n is even (but not both).

This puts us into the following abstract setup. Our goal is to define a function $c : \Pi_{x:A}C(x)$, namely the function $\texttt{collatz} : \mathbb{N} \to \mathbb{N}$. We already have a function $d : \Pi_{x:A}B(x)$, namely $d : \Pi_{n:\mathbb{N}}\text{is-decidable}(2 \mid n)$. So it suffices to define a function $h : \Pi_{x:A}B(x) \to C(x)$ because then we can define $c(x) :\equiv h(x, d(x))$. In this case this means we need a function

$$h : \Pi_{n:\mathbb{N}}\text{is-decidable}(2 \mid n) \to \mathbb{N}$$

which we can now define by cases from

$$h\text{-}\texttt{even}(n) :\equiv \lambda n.n/2\colon (2 \mid n) \to \mathbb{N} \quad \text{and} \quad h\text{-}\texttt{odd}(n) :\equiv \lambda n.3n+1 : \neg(2 \mid n) \to \mathbb{N}.$$

There is something called the "with-abstraction" that gives a concise syntax for functions defined in this manner.

The well-ordering principle of $\mathbb{N}$. The traditional well-ordering principle is about subsets of $\mathbb{N}$, or equivalently, about predicates on $\mathbb{N}$. In type theory, we replace these by decidable type families over $\mathbb{N}$.

defn. Let $P\colon \mathbb{N} \to \mathcal{U}$. A number $n : \mathbb{N}$ is a **lower bound** for P if it comes equipped with a term in the type

$$\text{is-lower-bound}_P(n) :\equiv \Pi_{k:\mathbb{N}}P(k) \to n \leqslant k$$

Similarly,

$$\text{is-upper-bound}_P(n) :\equiv \Pi_{k:\mathbb{N}}P(k) \to k \leqslant n$$

Theorem (well-ordering principle)**.** *Let P be a decidable family over $\mathbb{N}$ with d a witness that P is decidable. Then there is a function*

$$w(P, d) : (\Sigma_{n:\mathbb{N}}P(n)) \to (\Sigma_{m:\mathbb{N}}P(m) \times \text{is-lower-bound}_P(m)).$$

In other words, if $P(n)$ is inhabited for some n then there is a smallest $m : \mathbb{N}$ so that $P(m)$ is inhabited.

Proof. We will show that for any decidable type family $Q : \mathbb{N} \to \mathcal{U}$ that there is a function

$$Q(n) \to \Sigma_{m:\mathbb{N}} Q(m) \times \text{is-lower-bound}_Q(m)$$

by induction on n. When $n = 0$ we can use $m = 0$ since 0 is always a lower bound. For the inductive step we may assume we have the displayed function for every type family Q and consider a decidable type family Q with a term $q : Q(n+1)$. Our goal is to construct a term in the type

$$\Sigma_{m:\mathbb{N}} Q(m) \times \text{is-lower-bound}_Q(m)$$

. Since $Q(0)$ is decidable it suffices to construct a function

$$Q(0) + \neg Q(0) \to \Sigma_{m:\mathbb{N}} Q(m) \times \text{is-lower-bound}_Q m$$

so we can do a case analysis. If we have $Q(0)$ then it follows immediately that $m = 0$ is minimal. If $\neq Q(0)$, then we can consider the decidable family $Q' : \mathbb{N} \to \mathcal{U}$ defined by $Q(n) := Q'(\mathsf{succ}_{\mathbb{N}}(n))$. Since $q : Q'(n)$ we get a minimal element m for Q' by the inductive hypothesis. But since $Q(0)$ is assumed to be false then $m + 1$ is the minimal element for Q. □

The infinitude of primes. For natural numbers d and n we say d is a **proper divisor** of n if it comes with a term in the type

$$\text{is-proper-divisor}(n, d) := (d \neq n) \times (d \mid n)$$

With this notation we can say a natural number n is **prime** if it comes with a term in the type

$$\text{is-prime}(n) := \Pi_{x:\mathbb{N}} \text{is-proper-divisor}(n, x) \leftrightarrow (x = 1)$$

The proof of the infinitude of primes proceeds by constructing a prime number larger than n for any $n : \mathbb{N}$. So we can consider the type family for $n, m : \mathbb{N}$

$$R(n, m) := (n < m) \times \Pi_{x:\mathbb{N}}(x \leqslant n) \to (x \mid m) \to (x = 1)$$

of pairs so that m is greater than n and m is relatively prime to all numbers $x \leqslant n$. Since $n! + 1$ satisfies these properties for any n, the type family $m \mapsto R(n, m)$ in context $n : \mathbb{N}$ is inhabited. Thus, by the well-ordering principle, it has a least element p and this p must be prime.

Using the results we skipped it's possible to prove:

Lemma. *The type $R(n, m)$ is decidable for each $n, m : \mathbb{N}$.*

We leave it to the reader to verify that $R(n, n! + 1)$ is inhabited. Using these ingredients, we prove the infinitude of primes in the following form:

Theorem. *For each n, there is a prime number $p : \mathbb{N}$ so that $n < p$.*

Proof. It suffices to show this for each non-zero n since the case $n = 0$ follows. So let n be a non-zero natural number.

Since the type $R(n, m)$ is decidable for each m and since $R(n, n! + 1)$ holds by the well-ordering principle there is a minimal $p : \mathbb{N}$ so that $R(n, p)$. We will show that p is prime by constructing a term in the type

$$\text{is-prime}(p) := (p \neq 1) \times \Pi_{x:\mathbb{N}} \text{is-proper-divisor}(p, x) \to (x = 1).$$

By construction, $n < p$ and n is non-zero so $p \neq 1$. So now let x be a proper divisor of p. Since $R(n, p)$ holds by construction we can show that $x = 1$ by proving that $x \leqslant n$. Since p is non-zero and $x \mid p$ we must have $x < p$. By minimality of p it follows that $\neg R(n, x)$ holds. However, any divisor of x must also divide p by transitivity of divisibility, so

$$\Pi_{y:\mathbb{N}}(y \leqslant n) \to (y \mid x) \to (y = 1).$$

Since

$$\neg R(n, x) \doteq \neg \Big((n < x) \times \Pi_{y:\mathbb{N}}(y \leqslant n) \to (y \mid x) \to (y = 1) \Big)$$

holds we conclude that $\neg(n < x)$. On account of the logical equivalence $\neg(n < x) \leftrightarrow (x \leqslant n)$, it follows that $x \leqslant n$. □

Part 2. The Univalent Foundations of Mathematics

The univalent foundations build on Martin-Löf's dependent type theory by adding a few new axioms inspired by the interpretation of types as ∞-groupoids or spaces rather than sets. One of these axioms, Voevodsky's *univalence axiom*, is inconsistent with the interpretation of types as sets. When we meet the hierarchy of truncation levels, we'll see that some types behave like types and some types behave like sets while others have non-trivial higher-dimensional structures.

Recall the notion of **logical equivalence** between types A and B

$$A \leftrightarrow B \coloneqq (A \to B) \times (B \to A).$$

For instance, you proved on your homework that bool and $\mathbb{1} + \mathbb{1}$ are logically equivalent by constructing functions

$$\begin{array}{ccc} \texttt{bool} \xrightarrow{\texttt{bool-to-1+1}} \mathbb{1} + \mathbb{1} & & \mathbb{1} + \mathbb{1} \xrightarrow{\texttt{1+1-to-bool}} \texttt{bool} \\ \texttt{true} \longmapsto \texttt{inl}(\star) & & \texttt{inl}(\star) \longmapsto \texttt{true} \\ \texttt{false} \longmapsto \texttt{inr}(\star) & & \texttt{inr}(\star) \longmapsto \texttt{bool} \end{array}$$

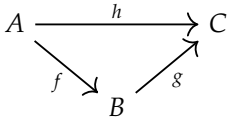
These inverse implications are obviously related. Our next goal is to develop language to explain how.

Homotopies. Surprisingly, we are not able, in dependent type theory, to construct identifications between functions $\texttt{1+1-to-bool} \circ \texttt{bool-to-1+1}$ and $\texttt{id}_{\text{bool}}$ or between $\texttt{succ}_{\mathbb{N}}$ and $\lambda n, n+1$. We can, however, construct *pointwise identifications* between these pairs of functions, which in homotopy type theory are commonly called **homotopies**:

defn. Let $f, g : \prod_{x:A} B(x)$. The type $f \sim g$ of **homotopies** from f to g is defined to be the type of pointwise identifications:

$$f \sim g \coloneqq \prod_{x:A} f(x) =_{B(x)} g(x).$$

Remark. Given three functions as below



we say the triangle **commutes** if $h \sim g \circ f$.

Note if $H, K : f \sim g \coloneqq \prod_{x:A} f(x) = g(x)$ are homotopies we can treat them as dependent functions in their own right and consider **homotopies between homotopies**, i.e., terms of the type $H \sim K \coloneqq \prod_{x:A} H(x) = K(x)$. This continues all the way up.

Indeed, the ∞-groupoid structure of identity types extends to homotopies as follows:

defn. For any type family $B \colon A \to \mathcal{U}$ we have operations

$$\texttt{refl-htpy} : \prod_{f:\prod_{x:A} B(x)} f \sim f \quad \texttt{inv-htpy} : \prod_{f,g:\prod_{x:A} B(x)} (f \sim g) \to (g \sim f)$$

$$\texttt{concat-htpy} : \prod_{f,g,h:\prod_{x:A} B(x)} f \sim g \to g \sim h \to f \sim h$$

defined pointwise by

$$\texttt{refl-htpy}(f) = \lambda x.\texttt{refl}_{f(x)} \quad \texttt{inv-htpy}(H) \coloneqq \lambda x.H(x)^{-1} \quad \texttt{concat-htpy}(H, K) \coloneqq \lambda x.H(x) \cdot K(x)$$

We abbreviate these latter two terms by writing H^{-1} and $H \cdot K$.

Note we don't abbreviate $\texttt{refl-htpy}_f : f \sim f \coloneqq \prod_{x:A} f(x) =_{B(x)} f(x)$ as $\texttt{refl}_f : f =_{\prod_{x:A} B(x)} f$ as these terms live in different types.

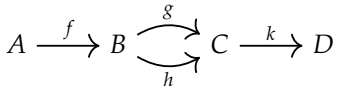
Proposition. *Homotopies satisfy the groupoid laws up to homotopy:*

(i) There is a homotopy $\texttt{assoc-htpy}(H, K, L) : (H \cdot K) \cdot L \sim H \cdot (K \cdot L)$ *for any homotopies* $H : f \sim g$, $K : g \sim h$, $L : h \sim i$.

(ii) There are homotopies $\texttt{left-unit-htpy}(H) : \texttt{refl-htpy}_f \cdot H \sim H$ *and* $\texttt{right-unit-htpy}(H) : H \cdot \texttt{refl-htpy}_g \sim H$.

(iii) There are homotopies $\texttt{left-inv-htpy}(H) : H^{-1} \cdot H \sim \texttt{refl-htpy}_g$ *and* $\texttt{right-inv-htpy}(H) : H \cdot H^{-1} \sim \texttt{refl-htpy}_f$.

In addition to the groupoid operations we make use of the following **whiskering operations** on homotopies which are relevant in the setting of functions



defn. Given the functions above and a homotopy $H\colon g \sim h$ define homopies

$$H \circ f \coloneqq \lambda a.H(f(a))\colon g \circ f \sim h \circ f$$

and

$$k \circ H \coloneqq \lambda b.\mathtt{ap}_k(H(b))\colon k \circ g \sim k \circ h.$$

Bi-invertible maps. We now explain what it means for a function $f\colon A \to B$ between types to define an **equivalence**. One explanation is that an equivalence is like a homotopy equivalence in topology, but we actually define the type family is-equiv in a manner that makes equivalences look more like "bi-invertible maps" for reasons that we will explain.

defn. Let $f\colon A \to B$.

(i) The type of **sections** of f is defined to be the type

$$\mathrm{sec}(f) \coloneqq \Sigma_{g:B\to A} f \circ g \sim \mathtt{id}_B.$$

(ii) The type of **retractions** of f is defined to be the type

$$\mathrm{retr}(f) \coloneqq \Sigma_{h:B\to A} h \circ f \sim \mathtt{id}_A.$$

(iii) We say f is an **equivalence** if it has a section and a retraction:

$$\text{is-equiv}(f)\colon \mathrm{sec}(f) \times \mathrm{retr}(f).$$

We write $A \simeq B$ for the type $\Sigma_{f:A\to B}\text{is-equiv}(f)$ of all equivalences from A to B.

Explicitly the data of a term in is-equiv(f) involves two maps $g, h : B \to A$ and two homotopies $G : f \circ g \sim \mathtt{id}_B$ and $H\colon h \circ f \sim \mathtt{id}_A$. We have seen a bunch of examples:

ex.

- Identity functions are equivalences.
- `bool-to-1+1`$\colon$ bool $\to \mathbb{1} + \mathbb{1}$ and `1+1-to-bool`$\colon \mathbb{1} + \mathbb{1} \to$ bool are inverse equivalences.
- `neg-bool`$\colon$ bool $\to$ bool is an equivalence.
- `pred`$\colon \mathbb{Z} \to \mathbb{Z}$ is an equivalence, inverse to `succ`$\colon \mathbb{Z} \to \mathbb{Z}$.

We might also define

$$\text{has-inverse}(f) \coloneqq \Sigma_{g:B\to A}(f \circ g \sim \mathtt{id}_B) \times (g \circ f \sim \mathtt{id}_A).$$

The reason we did not define equivalences to be functions that have inverses is that we wanted being an equivalence to be a **property** associated to a map f rather than a **structure** on the map, in a sense we'll discuss soon. Essentially because of the uniqueness of reflexivity paths, if f is an equivalence then it only has one section and homotopy and only one retraction and homotopy, up to homotopy. We'll see, however, that the data of inverses to f can be more complicated.

But even though the data encoded by terms in is-equiv(f) and has-inverse(f) may differ, these types are closely related:

Proposition. *The types* is-equiv(f) *and* has-inverse(f) *are logically equivalent. In particular, if f is an equivalence it has a two-sided inverse.*

Proof. The data of an inverse obviously defines the data of an equivalence, so we have has-inverse$(f) \to$ is-equiv(f).

For the converse, suppose we have $g, h\colon B \to A$ and homotopies $G : f \circ g \sim \mathtt{id}_B$ and $H\colon h \circ f \sim \mathtt{id}_A$. Then for any $b : B$ we have

$$g(b) \overset{H(g(b))^{-1}}{=\!=} h(f(g(b))) \overset{\mathtt{ap}_h(G(b))}{=\!=} h(b)$$

defining a homotopy $K\colon g \sim h$. Using this we can see that g is also a retraction of f by the identification for any $a : A$

$$g(f(a)) \overset{K(f(a))}{=\!=} h(f(a)) \xrightarrow{H(a)} a$$

□

Corollary. *A section or retraction of an equivalence is an equivalence.*

Proof. We've just seen that the section of an equivalence is also a retraction and thus is an invertible map with inverse f. A dual construction applies to a retraction. □

Up to equivalence, types satisfy many familiar laws, such as

$$A + B \simeq B + A \qquad A \times B \simeq B \times A$$
$$A \times \mathbb{1} \simeq A \qquad A + \varnothing \simeq A$$
$$A \times (B + C) \simeq (A \times B) + (A \times C)$$

where the required equivalences and homotopies can be constructed by induction. These equivalences extend to cases such as

$$\Sigma_{x:\varnothing} B(x) \simeq \varnothing \quad \Sigma_{x:\mathbb{1}} B(x) \simeq B(\star).$$

and

$$\Sigma_{z:\Sigma_{x:A} B(x)} C(z) \simeq \Sigma_{x:A} \Sigma_{y:B(x)} C(x,y) \quad \Sigma_{w:A+B} C(w) \simeq \Sigma_{x:A} C(\mathtt{inl}x) + \Sigma_{y:B} C(\mathtt{inr}y).$$

In the presence of an additional axiom, we'll be able to prove similar equivalences involving function types, but we can't deduce these just yet.

Characterizing the identity types of Σ-types. We return to the theme of characterizing identity types to characterize the identity type of a Σ-type, up to equivalence. We follow the same outline used to characterize the identity type family of the natural numbers above:

(i) We define a binary relation $R\colon A \to A \to \mathcal{U}$ that we suspect is equivalent to the identity type family.
(ii) Prove reflexivity by constructing a term in $\Pi_{x:A} R(x,x)$.
(iii) Use reflexivity and path induction to define a canonical map $\Pi_{x,y:A} x =_A y \to R(x,y)$.
(iv) Show that the map $x =_A y \to R(x,y)$ is an equivalence for each $x, y : A$.

The last step is usually the most difficult and we will refine our techniques for dealing with it soon.

In this section, we consider $B\colon A \to \mathcal{U}$ and the corresponding type $\Sigma_{x:A} B(x)$. Given dependent pairs (a,b) and (a',b') we might imagine that the data of an identification between them involves a pair of identifications comprised firstly of a path $p : a =_A a'$ and then of a path $q : \mathtt{tr}_B(p,b) = b'$. We first turn this idea into a binary relation.

defn. Define

$$\mathrm{Eq}_\Sigma\colon (\Sigma_{x:A} B(x)) \to (\Sigma_{x:A} B(x)) \to \mathcal{U}$$

by

$$\mathrm{Eq}_\Sigma(s,t) :\equiv \Sigma_{p:\mathtt{pr}_1(s)=\mathtt{pr}_1(t)} \mathtt{tr}_B(p, \mathtt{pr}_2(s)) = \mathtt{pr}_2(t).$$

Lemma. *The relation* Eq_Σ *is reflexive.*

Proof. By Σ-induction it suffices to let $s : \Sigma_{x:A} B(x)$ be a pair $(a : A, b : B(a))$ in which case we have

$$\lambda a, \lambda b.(\mathtt{refl}_a, \mathtt{refl}_b)\colon \Pi_{x:A} \Pi_{y:B(x)} \Sigma_{p:x=x} \mathtt{tr}_B(p,y) = y$$

since we have a definitional equality $\mathtt{tr}_B(\mathtt{refl}_a, b) \doteq b$. □

By path induction, we may define a map $\mathtt{pair\text{-}eq} : s = t \to \mathrm{Eq}_\Sigma(s,t)$ for any $s, t : \Sigma_{x:A} B(x)$ by sending $\mathtt{refl}_s$ to the corresponding reflexivity term.

Theorem. *For any type family* $B\colon A \to \mathcal{U}$*, the map*

$$\mathtt{pair\text{-}eq}\colon (s = t) \to \mathrm{Eq}_\Sigma(s,t)$$

is an equivalence for every $s, t : \Sigma_{x:A} B(x)$*.*

Proof. We define an inverse equivalence $\mathtt{eq\text{-}pair}\colon \mathrm{Eq}_\Sigma(s,t) \to (s = t)$ by repeated Σ-induction. For pairs (a,b) and (a',b') we must define

$$\mathtt{eq\text{-}pair}\colon \Sigma_{p:a=a'} \mathtt{tr}_B(p,b) = b' \to ((a,b) = (a',b'))$$

which we again do by Σ-induction. It suffices to define a function in the type

$$\Pi_{p:a=a'} (\mathtt{tr}_B(p,b) = b') \to ((a,b) = (a',b')).$$

By double path induction we may first assume b' is $\mathtt{tr}_B(p,b)$ and the second path is $\mathtt{refl}$. Then by path induction we may assume that a' is a and p is refl in which case we send this pair $(\mathtt{refl}_a, \mathtt{refl}_b)$ to $\mathtt{refl}_{(a,b)} : ((a,b) = (a,b))$.

To see that $\mathtt{eq\text{-}pair}$ is a section of $\mathtt{pair\text{-}eq}$ we require identifications

$$\mathtt{pair\text{-}eq}(\mathtt{eq\text{-}pair}(p,q)) = (p,q)$$

for each $(p, q) : \Sigma_{p:a=a'} \mathtt{tr}_B(p, b) = b'$. By a double path induction it suffices to identify

$$\mathtt{pair\text{-}eq}(\mathtt{eq\text{-}pair}(\mathtt{refl}_a, \mathtt{refl}_b)) = (\mathtt{refl}_a, \mathtt{refl}_b)$$

and the left-hand side is judgmentally equal to the right-hand side, so we may use $\mathtt{refl}_{(\mathtt{refl}_a, \mathtt{refl}_b)}$.

To see that `eq-pair` is a retraction of `pair-eq` we require identifications

$$\mathtt{eq\text{-}pair}(\mathtt{pair\text{-}eq}(p)) = p$$

for each $p : s = t$. By path induction we may take t to be s and p to be $\mathtt{refl}_s$. Then we require

$$\mathtt{eq\text{-}pair}(\mathtt{refl}_{\mathtt{pr}_1(s)}, \mathtt{refl}_{\mathtt{pr}_2(t)}) = \mathtt{refl}_s.$$

By Σ-induction we may consider the case of a pair (a, b) in which case we require

$$\mathtt{eq\text{-}pair}(\mathtt{refl}_a, \mathtt{refl}_b) = \mathtt{refl}_{(a,b)}.$$

Since the left-hand side we defined to be the right-hand side, we may use `refl`. □

October 6: Contractibility

Contractible types are singletons up to homotopy. In this section, we'll see several characterizations of contractibility and then relativize our discussion to include so-called "contractible maps."

Contractible types. The term "contractibility" is inspired by the homotopical interpretation of type theory but it can also be thought of as an expression of "uniqueness": we say that a type A is contractible just when it contains a unique element in a sense encoded by the Curry-Howard interpretation:

defn. A type A is **contractible** if it comes equipped with a term in the type

$$\text{is-contr}(A) \coloneqq \Sigma_{c:A} \Pi_{x:A} c = x.$$

Given $(c, C) : \text{is-contr}(A)$ we refer to $c : A$ as the **center of contraction** and $C : \Pi_{x:A} c = x$ as the **contraction** or **contracting homotopy**.

Q. This terminology suggests that C is a homotopy between some pair of functions. Which functions?

ex. The unit type is easily seen to be contractible. We take $\star : \mathbb{1}$ as the center of contraction and define $C : \Pi_{x:\mathbb{1}} \star = x$ by singleton induction: in the case where x is $\star$, we define $C(\star) = \mathtt{refl}_\star$.

We've met another example of a contractible type already.

Theorem. *For any $a : A$ the type*

$$\Sigma_{x:A} a = x$$

is contractible.

Proof. We take $(a, \mathtt{refl}_a) : \Sigma_{x:A} a = x$ to be the center of contraction. In our proof of the uniqueness of reflexivity we constructed the required contracting homotopy. □

Singleton induction. Contractible types satisfy an induction principle similar to singleton types.

defn. Suppose A comes with a term $a : A$. Then we say A satisfies **singleton induction** if for every type family B over A the map

$$\mathtt{ev\text{-}pt} : (\Pi_{x:A} B(x)) \to B(a)$$

defined by $\mathtt{ev\text{-}pt} f \coloneqq f(a)$ has a section. The data of this section is then given by a function and a homotopy

$$\mathtt{ind\text{-}sing}_a : B(a) \to \Pi_{x:A} B(x) \quad \text{and} \quad \mathtt{comp\text{-}sing}_a : \mathtt{ev\text{-}pt} \circ \mathtt{ind\text{-}sing}_a \sim \mathtt{id}.$$

The singleton induction principle is almost the same as the induction principle for the unit type except for one point of difference: the "computation rule" for singleton induction is stated using an *identification* rather than a judgmental equality. Note in particular, that $\star : \mathbb{1}$ satisfies singleton induction with $\lambda x.\mathtt{refl}_x$ as the required homotopy.

Theorem. *For a type A the following are logically equivalent.*

(i) A is contractible.
(ii) A comes equipped with a term $a : A$ satisfying singleton induction.

Proof. Suppose we have $(a, C) : \text{is-contr}(A) :\equiv \sum_{c:A} \prod_{x:A} c = x$. We may replace the homotopy $C : \mathtt{const}_a \sim \mathtt{id}_A$ by a new homotopy C' defined by $C'(x) = C(a)^{-1} \cdot C(x)$. We then have an identification $\mathtt{left\text{-}inv} : C'(a) = \mathtt{refl}_a$. So without loss of generality we suppose $(a, C) : \text{is-contr}(A)$ comes with a term $p : C(a) = \mathtt{refl}_a$.

Consider a type family B over A. For each $b : B(a)$ our goal is to define $\mathtt{ind\text{-}sing}_a(b) : \prod_{x:A} B(x)$, which we do by

$$\mathtt{ind\text{-}sing}_a(b) :\equiv \mathtt{tr}_B(C(x), b) : B(x).$$

It remains to construct an identification $\mathtt{ind\text{-}sing}_a(b, a) = b$. We have a function

$$\lambda(q : a = x).\mathtt{tr}_B(q, b)$$

that evaluates at the path $C(a)$ to give $\mathtt{ind\text{-}sing}_a(b)$ and evalutes at the path $\mathtt{refl}_a$ to give b. When we apply this function to the path $p : C(a) = \mathtt{refl}_a$ we get the desired identification.

Conversely, suppose $a : A$ satisfies singleton induction. To prove that A is contractible with center of contraction a we apply singleton induction to the type family $B(x) :\equiv a = x$ to obtain

$$\mathtt{ind\text{-}sing}_a : a = a \to \Pi_{x:A} a = x.$$

Now $\mathtt{ind\text{-}sing}_a(\mathtt{refl}_a)$ is a contracting homotopy. □

Fibers.

defn. Let $f\colon A \to B$ be a function between types and consider $b : B$. The **fiber** of f over b is the type

$$\text{fib}_f(b) :\equiv \Sigma_{a:A} f(a) = b.$$

That is the fiber is what in other contexts might get called the "homotopy fiber": it is comprised of those terms $a : A$ whose images $f(a) : B$ are identified with b.

It will be useful to identify the identity type of the fiber, which we do by following our usual strategy.

defn. Let $f : A \to B$ and $b : B$. Given two terms $(a, p), (a', p') : \text{fib}_f(b)$ define a type

$$\text{Eq-fib}_f((a, p), (a', p')) :\equiv \Sigma_{\alpha:a=a'} p = \mathtt{ap}_f(\alpha) \cdot p'.$$

Note $\text{Eq-fib}_f : \text{fib}_f(b) \to \text{fib}_f(b) \to \mathcal{U}$ is reflexive since we have

$$\lambda(a, p).(\mathtt{refl}_a, \mathtt{refl}_p)\colon \Pi_{(a,p):\text{fib}_f(b)} \text{Eq-fib}_f((a, p), (a, p)).$$

Proposition. *Consider $f\colon A \to B$ and $b : B$. The canonical map*

$$(a, p) =_{\text{fib}_f(b)} (a', p') \to \text{Eq-fib}_f((a, p), (a', p'))$$

induced by the reflexivity term is an equivalence for any pair of points $(a, p), (a', p') : \text{fib}_f(b)$.

Proof. For the inverse equivalence, we need a function

$$\left(\Sigma_{\alpha:a=a'} p = \mathtt{ap}_f(\alpha) \cdot p'\right) \to (a, p) =_{\text{fib}_f(b)} (a', p').$$

By Σ-induction it suffices to consider the image of a pair $\alpha : a = a'$ and $\beta : p = \mathtt{ap}_f(\alpha) \cdot p'$. By path induction we may take p to be $\mathtt{ap}_f(\alpha) \cdot p'$ and β to be $\mathtt{refl}$. By path induction again, we may take a to be a' and α to be $\mathtt{refl}$. Since $\mathtt{ap}_f(\mathtt{refl}) \cdot p'$ is judgmentally equal to p' we may use $\mathtt{refl}$ as our identification between $(a', \mathtt{ap}_f(\mathtt{refl}) \cdot p')$ and (a', p'). The construction of the homotopies witnessing the left and right inverses are similar. □

Contractible maps and equivalences.

defn. We say that a function $f\colon A \to B$ is **contractible** if it comes equipped with a term of type

$$\text{is-contr}(f) :\equiv \Pi_{b:B} \text{is-contr}(\text{fib}_f(b)).$$

Theorem. *Any contractible map is an equivalence.*

Proof. Suppose $f\colon A \to B$ is contractible. The center of contraction $(g(b), G(b))$ in each fiber gives rise to a dependent function

$$\lambda b.(g(b), G(b))\colon \Pi_{b:B}\mathrm{fib}_f(b).$$

In particular $g : B \to A$ defines a map and $G\colon f \circ g \sim \mathtt{id}_B$ defines a homotopy. So g is a section of f.

It remains to show that g is also a retraction of f by defining a term in the type $g \circ f \sim \mathtt{id}_A$. For each $a : A$ we have an identification $p : f(g(f(a))) = f(a)$ since g is a section of f. So $(g(f(a)), p) : \mathrm{fib}_f(f(a))$. Since this type is contractible there is an identification $\beta : (g(f(a)), p) = (a, \mathtt{refl}_{f(a)})$. The base path $\mathtt{ap}_{\mathtt{pr}_1}(\beta) : g(f(a)) = a$ gives the desired identification. □

In fact, equivalences necessarily also define contractible maps, which we show in a few steps. Recall an **invertible map** is a map $f\colon A \to B$ equipped with $g\colon B \to A$ and homotopies $G : f \circ g \sim \mathtt{id}$ and $H : g \circ f \sim \mathtt{id}$. Then the whiskered homotopies $G \circ f$ and $f \circ H$ both have the type $f \circ g \circ f \sim f$.

defn. A map $f\colon A \to B$ is **coherently invertible** if it comes with

$$g : B \to A, \quad G : f \circ g \sim \mathtt{id}, \quad H : g \circ f \sim \mathtt{id}, \quad K : G \circ f \sim f \circ H.$$

Proposition. *Any coherently invertible map has contractible fibers.*

Proof. Given $f\colon A \to B$ together with

$$g : B \to A, \quad G : f \circ g \sim \mathtt{id}, \quad H : g \circ f \sim \mathtt{id}, \quad K : G \circ f \sim f \circ H.$$

we wish to show that $\mathrm{fib}_f(b)$ is contractible for any $b : B$. We take $(g(b), G(b)) : \mathrm{fib}_f(b)$ as the center of contraction.

For the contracting homotopy it suffices to define a term in the equivalent type

$$\prod_{a:A} \prod_{p:f(a)=b} \text{Eq-fib}_f((g(b), G(b)), (a, p)).$$

By path induction on p it suffices to define a term in the type

$$\prod_{a:A} \text{Eq-fib}_f((g(f(a)), G(f(a))), (a, \mathtt{refl}_{f(a)})).$$

By the definition of Eq-fib_f this means that we need, for each $a : A$, a path $H(a) : g(f(a)) = a$ and a further identification

$$G(f(a)) = \mathtt{ap}_f(H(a)) \cdot \mathtt{refl}_{f(a)}.$$

We have $K(a) : G(f(a)) = \mathtt{ap}_f(H(a))$ so we get the identification we want by composing with `right-unit-htpy`. □

Our next goal is to show that any invertible map $f\colon A \to B$ equipped with $g\colon B \to A$ and homotopies $G : f \circ g \sim \mathtt{id}$ and $H : g \circ f \sim \mathtt{id}$ can be improved to a coherently invertible map at the cost of replacing G with a new homotopy $G' : f \circ g \sim \mathtt{id}$ satisfying the coherence $K : f \circ H \sim G' \circ f$. The upshot is that we have a map

$$\text{has-inverse}(f) \to \text{is-coh-invertible}(f).$$

The construction is by messy path algebra that you can read about in [R, §10.4].

Corollary. *For any type A and term $a : A$ the type*

$$\Sigma_{x:A} x = a$$

is contractible.

Proof. This type is the fiber of the identity function $\mathtt{id}_A\colon A \to A$ over $a : A$. Since $\mathtt{id}_A$ is an equivalence, this type must be contractible. □

Soon we'll have a second proof: we'll be able to use the equivalence $\mathtt{inv}\colon (a = x) \to (x = a)$ to define an equivalence $\lambda x.\mathtt{inv}\colon \Sigma_{x:A} a = x \to \Sigma_{x:A} x = a$. It follows easily that any type equivalent to a contractible type is contractible.

Families of equivalences. For any family of maps $f\colon \prod_{x:A} B(x) \to C(x)$ there is a map

$$\mathtt{tot}(f) : \Sigma_{x:A} B(x) \to \Sigma_{x:A} C(x)$$

defined by $\lambda(x,y).(x, f(x,y))$.

Theorem. *For any family of maps $f\colon \prod_{x:A} B(x) \to C(x)$ the following are logically equivalent:*

*(i) The family f is a **family of equivalences**: for each $x : A$ the map $f(x)$ is an equivalence.*
(ii) The map $\mathtt{tot}(f)$ is an equivalence.

Proof. Recall equivalences are contractible maps: meaning maps whose fibers are all contractible. So it suffices to show that $f(x)$ is a contractible map for each x if and only if $\mathtt{tot}(f)$ is a contractible map. For this, we must show for each $x : A$ and $c : C(x)$ that $\mathrm{fib}_{f(x)}(c)$ is contractible if and only if $\mathrm{fib}_{\mathrm{tot}(f)}(x,c)$ is contractible. But in fact these fibers are always equivalent, as the following lemma shows. □

Lemma. *For any family of maps $f\colon \prod_{x:A} B(x) \to C(x)$ and any term $t : \sum_{x:A} C(x)$ there is an equivalence*

$$\mathrm{fib}_{\mathtt{tot}(f)}(t) \simeq \mathrm{fib}_{f(\mathtt{pr}_1(t))}\mathtt{pr}_2(t).$$

Proof. For all $t : \Sigma_{x:A} C(x)$ define $\phi(t)\colon \mathrm{fib}_{\mathtt{tot}(f)}(t) \to \mathrm{fib}_{f(\mathtt{pr}_1(t))}\mathtt{pr}_2(t)$ by pattern matching by taking $(x, f(x,y), (x,y), \mathtt{refl})$ to $(y, \mathtt{refl})$. To see that $\phi(t)$ is an equivalence for each $t : \Sigma_{x:A} C(x)$ we construct an inverse by pattern matching

$$\phi(x, f(x,y), y, \mathtt{refl}) \coloneqq ((x,y), \mathtt{refl})$$

and homotopies by pattern matching in which case both homotopies reduce to $\mathtt{refl}$. □

Now consider a closely related situation where we are given a map $f\colon A \to B$ and a family $C\colon B \to \mathcal{U}$. We have a map

$$\lambda(x,z).(f(x), z) : \Sigma_{x:A} C(f(x)) \to \Sigma_{y:B} C(y).$$

Again, by the same style of argument, if f is an equivalence then this map is an equivalence (because the fibers are equivalent), but in this case the converse does not hold: consider $\mathtt{true} : \mathbb{1} \to \mathrm{bool}$ and the type family $\mathtt{false} = b : \mathrm{bool} \to \mathcal{U}$.

Nevertheless we can use the one-sided implication to extend the previous theorem as follows. Given $f\colon A \to B$ and a family of maps $g : \prod_{x:A} C(x) \to D(f(x))$ where C is a type family over A and D is a type family over B, we say that g is a **family of maps over** f. Define

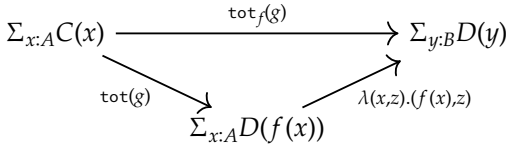
$$\mathtt{tot}_f(g) : \Sigma_{x:A} C(x) \to \Sigma_{y:B} D(y)$$

by $\mathtt{tot}_f(g)(x,z) \coloneqq (f(x), g(x,z))$.

Theorem. *Suppose g is a family of maps over f and f is an equivalence. Then the following are logically equivalent:*

(i) The family of maps g over f is a family of equivalences.
(ii) The map $\mathtt{tot}_f(g)$ is an equivalence.

Proof. We have a commuting triangle of maps



Since f is an equivalence, the bottom right map is an equivalence. The equivalences are closed under the 2-of-3 property (meaning if any two of a composable pair and their composite are equivalences so is the third map). Thus $\mathtt{tot}(g)$ is an equivalence if and only if $\mathtt{tot}_f(g)$ is an equivalence. And by the previous theorem, the first condition asserts that g is a family of equivalences. □

The fundamental theorem. The fundamental theorem of identity types provides necessary and sufficient conditions for a type family $B\colon A \to \mathcal{U}$ and terms $a : A$ and $b : B(a)$ to define a family of equivalences $\prod_{x:A}(a = x) \simeq B(x)$ by $(a, \mathtt{refl}) \mapsto b$. In fact, we'll see we can be a bit less particular about how exactly the family of equivalences is defined.

defn. Given a type A and a term $a : A$ a **(unary) identity system** on A at a is given by a type family $B : A \to \mathcal{U}$ and a term $b : B(a)$ so that for any family of types $P\colon \Sigma_{x:A}B(x) \to \mathcal{U}$ the function

$$\mathtt{ev}_{a,b}\colon \prod_{x:A}\prod_{y:B(x)}P(x,y) \to P(a,b)$$

has a section.

That is, if (B, b) is an identity system at (A, a) and P is a family of types over $x : A$ and $y : B(x)$ then for each $p : P(a, b)$ there is a dependent function $f : \prod_{x:A}\prod_{y:B(x)}P(x, y)$ so that $f(a, b) = p$. This is a variant of the path induction principal where the computation rule is given by an identification.

Theorem (fundamental theorem of identity types)**.** *Let A be a type, let $a : A$, and let $B : A \to \mathcal{U}$. Then the following are logically equivalent for any family of maps*

$$f : \prod_{x:A}(a = x) \to B(x).$$

(i) f is a family of equivalences.
(ii) The total space $\Sigma_{x:A}B(x)$ is contractible.
(iii) The family B is an identity system.

In particular, for any $b : B(a)$ the canonical map

$$\mathtt{path\text{-}ind}_a(b)\colon \prod_{x:A}(a = x) \to B(x)$$

is a family of equivalences if and only if $\Sigma_{x:A}B(x)$ is contractible.

Proof. By our theorem characterizing families of equivalences f is a family of equivalences iff $\mathtt{tot}(f)$ induces an equivalence

$$(\Sigma_{x:A}(a = x)) \simeq (\Sigma_{x:A}B(x))\,.$$

The left-hand type is contractible so this is the case if and only if $\Sigma_{x:A}B(x)$ is contractible. This proves the equivalence of (i) and (ii).

For the equivalence of (ii) and (iii) consider the commutative triangle:

$$\begin{array}{ccc} \prod_{t:\Sigma_{x:A}B(x)}P(t) & \xrightarrow{\mathtt{ev\text{-}pair}} & \prod_{x:A}\prod_{y:B(x)}P(x,y) \\ & {\scriptstyle \mathtt{ev\text{-}pt}_{(a,b)}} \searrow \quad \swarrow {\scriptstyle \mathtt{ev}_{(a,b)}} & \\ & P(a,b) & \end{array}$$

By Σ-induction, the top map has a section. It follows that the left map has a section if and only if the right map has a section. The left-hand section is the universal property called singleton induction that is satisfied if and only if the type $\Sigma_{x:A}B(x)$ is contractible, while the right-hand section is the universal property of an identity system. This proves the equivalence of (ii) and (iii). □

Equality on $\mathbb{N}$. As our first application recall the observational equality type family $\mathrm{Eq}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} \to \mathcal{U}$ satisfying

$$\mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) \doteq \mathbb{1} \quad \mathrm{Eq}_{\mathbb{N}}(\mathtt{succ}_{\mathbb{N}}(n), 0_{\mathbb{N}}) \doteq \varnothing \quad \mathrm{Eq}_{\mathbb{N}}(0, \mathtt{succ}_{\mathbb{N}}(n)) \doteq \varnothing \quad \mathrm{Eq}_{\mathbb{N}}(\mathtt{succ}_{\mathbb{N}}(m), \mathtt{succ}_{\mathbb{N}}(n)) \doteq \mathrm{Eq}_{\mathbb{N}}(m, n).$$

We previously showed that this type family is logically equivalent to the identity type family for the natural numbers, but we can do better. Using the reflexivity term $\mathtt{refl\text{-}Eq}_{\mathbb{N}} : \prod_{n:\mathbb{N}} \mathrm{Eq}_{\mathbb{N}}(n, n)$ we have a canonical map $(m = n) \to \mathrm{Eq}_{\mathbb{N}}(m, n)$ defined by path induction.

Theorem. *For all $m, n : \mathbb{N}$ the canonical map*

$$(m = n) \to \mathrm{Eq}_{\mathbb{N}}(m, n)$$

is an equivalence.

Proof. It suffices to show for each $m : \mathbb{N}$ that the type

$$\Sigma_{n:\mathbb{N}} \mathrm{Eq}_{\mathbb{N}}(m, n)$$

is contractible. We take $(m, \mathtt{refl\text{-}Eq}_{\mathbb{N}}(m))$ as the center of contraction.

The contracting homotopy

$$\gamma(m) : \Pi_{n:\mathbb{N}} \Pi_{e:\mathrm{Eq}_{\mathbb{N}}(m,n)} (m, \mathtt{refl\text{-}Eq}_{\mathbb{N}}(m)) = (n, e)$$

is defined by induction on $m, n : \mathbb{N}$ from the base case $\gamma(0, 0, \star) :\equiv \mathtt{refl}$. If either m or n is 0 and the other is a successor we can define this using ex-falso.

In the inductive step we seek an identification $\gamma(m+1, n+1, e) : (m+1, \mathtt{refl\text{-}Eq}_{\mathbb{N}}(m+1)) = (n+1, e)$. To define this we use the map

$$\lambda(n, e).(n+1, e) : \Sigma_{n:\mathbb{N}} \mathrm{Eq}_{\mathbb{N}}(m, n) \to \Sigma_{n:\mathbb{N}} \mathrm{Eq}_{\mathbb{N}}(m+1, n).$$

Since this map carries $(m, \mathtt{refl\text{-}Eq}_{\mathbb{N}}(m))$ to $(m+1, \mathtt{refl\text{-}Eq}_{\mathbb{N}}(m+1))$ we can apply the map to the identification (m, n, e) to get the identification we seek. □

Embeddings. Our next application will show that equivalences are embeddings, defined as follows:

defn. An **embedding** is a map $f\colon A \to B$ that satisfies the property that

$$\mathtt{ap}_f : (x = y) \to (f(x) = f(y))$$

is an equivalence for every $x, y : A$.

Write

$$\text{is-emb}(f) :\equiv \Pi_{x,y:A} \text{is-equiv}(\mathtt{ap}_f : (x = y) \to (f(x) = f(y))).$$

Theorem. *Equivalences are embeddings.*

Proof. Suppose $e : A \simeq B$ is an equivalence and $x : A$. We wish to show that

$$\mathtt{ap}_e : (x = y) \to (e(x) = e(y))$$

is an equivalence for every $y : A$. For this it suffices to show that $\Sigma_{y:A} e(x) = e(y)$ is contractible. We have an equivalence

$$\Sigma_{y:A} e(x) = e(y) \simeq \Sigma_{y:A} e(y) = e(x)$$

and the latter type is the fiber $\mathrm{fib}_e(e(x))$. Since e is an equivalence this fiber is contractible so the result follows. □

Disjointness of coproducts. For a third application, we characterize the identity types of coproducts.

Theorem. *Let A and B be types. Then for any $a, a' : A$ and $b, b' : B$ there are equivalences*

$$(\mathtt{inl}(a) = \mathtt{inl}(a')) \simeq (a = a')$$
$$(\mathtt{inl}(a) = \mathtt{inr}(b)) \simeq \varnothing$$
$$(\mathtt{inr}(b) = \mathtt{inl}(a)) \simeq \varnothing$$
$$(\mathtt{inr}(b) = \mathtt{inr}(b')) \simeq (b = b')$$

We follow our usual strategy first defining a type family

$$\mathrm{Eq}\text{-}+_{A,B} : (A + B) \to (A + B) \to \mathcal{U}$$

by induction by

$$\mathrm{Eq}\text{-}+_{A,B}(\mathtt{inl}(a), \mathtt{inl}(a')) \simeq (a = a')$$
$$\mathrm{Eq}\text{-}+_{A,B}(\mathtt{inl}(a), \mathtt{inr}(b)) \simeq \varnothing$$
$$\mathrm{Eq}\text{-}+_{A,B}(\mathtt{inr}(b), \mathtt{inr}(a)) \simeq \varnothing$$
$$\mathrm{Eq}\text{-}+_{A,B}(\mathtt{inr}(b), \mathtt{inr}(b')) \simeq (b = b')$$

Again by induction there is a term $\mathtt{Eq\text{-}+\text{-}refl} : \Pi_{z:A+B} \mathrm{Eq}\text{-}+_{A,B}(z, z)$ defined by $\mathtt{refl}$ in both cases. Thus there is a map

$$\mathtt{Eq\text{-}+\text{-}eq} : \Pi_{s,t:A+B} (s = t) \to \mathrm{Eq}\text{-}+_{A,B}(s, t)$$

defined by path induction.

Proposition. *For any $s : A + B$ the total space*

$$\Sigma_{t:A+B}\mathrm{Eq}\text{-}+_{A,B}(s,t)$$

is contractible.

Proof. By induction on s we have to consider two cases, of which we prove just one: that

$$\Sigma_{t:A+B}\mathrm{Eq}\text{-}+_{A,B}(\mathtt{inl}(a),t)$$

is contractible. From distributivity of dependent pairs of coproducts we have

$$\Sigma_{t:A+B}\mathrm{Eq}\text{-}+_{A,B}(\mathtt{inl}(a),t) \simeq \Sigma_{x:A}\mathrm{Eq}\text{-}+_{A,B}(\mathtt{inl}(a),\mathtt{inl}(x)) + \Sigma_{b:B}\mathrm{Eq}\text{-}+_{A,B}(\mathtt{inl}(a),\mathtt{inr}(b)) \simeq \Sigma_{x:A}(a = \mathtt{inl}(x)) + \Sigma_{b:B}\varnothing$$

$$\simeq \Sigma_{x:A}(a = \mathtt{inl}(x)) + \varnothing \simeq \Sigma_{x:A}(a = \mathtt{inl}(x)).$$

This last type is contractible so the first type is as claimed. □

This establishes the desired family of equivalences of types.

The structure identity principle. The notion of identity system $B : A \to \mathcal{U}$ and $b : B(a)$ over a type A and term $a : A$ can be extended to a notion of dependent identity system. A **dependent identity system** over (B, b) is given by a type family $C : A \to \mathcal{U}$ together with

$$D : \Pi_{x:A}B(x) \to C(x) \to \mathcal{U}$$

and a term $c : C(a)$ so that $z \mapsto D(a, b, z)$ is an identity system at c. We leave the details to [R, §11.6].

October 13: Propositions and sets

We now formally study propositions in homotopy type theory.

Propositions.

defn. A type A is a **proposition** if all of its identity types are contractible: i.e., if it comes with a term of type

$$\text{is-prop}(A) :\equiv \Pi_{x,y:A}\text{is-contr}(x = y).$$

Given a universe $\mathcal{U}$ we define $\mathrm{Prop}_{\mathcal{U}}$ to be the type of all small propositions:

$$\mathrm{Prop}_{\mathcal{U}} :\equiv \Sigma_{X:\mathcal{U}}\text{is-prop}(X).$$

ex. We have shown that identity types of contractible types are contractible. Thus contractible types are propositions.

ex. The empty type is also a proposition, for ex-falso inhabits

$$\Pi_{x,y:\varnothing}\text{is-contr}(x = y).$$

There are many equivalent ways to assert that a type is a proposition.

Proposition. *For a type A, the following are logically equivalent:*

(i) A is a proposition.
(ii) Any two terms of type A can be identified: i.e., there is a dependent function in the type

$$\Pi_{x,y:A}x = y.$$

(iii) The type A is contractible as soon as it is inhabited: i.e., there is a term of type $A \to \text{is-contr}(A)$.
(iv) The map $\mathtt{const}_\star : A \to \mathbb{1}$ is an embedding.

Proof. (i) clearly implies (ii), by using the center of contraction. Assuming (ii) we have $p : \Pi_{x,y:A}x = y$. Note for any x that $p(x) : \Pi_{y:A}x = y$ is a contracting homotopy onto x. Thus, we have a function

$$\lambda x.(x, p(x)) : A \to \text{is-contr}(A).$$

Next assume (iii) and consider a function $c : A \to \text{is-contr}(A)$. To prove

$$\Pi_{x,y:A}\text{is-equiv}(\mathtt{ap}_{\mathtt{const}_\star} : (x = y) \to (\star = \star))$$

it suffices to prove

$$A \to \left(\prod_{x,y:A} \text{is-equiv}(\mathsf{ap}_{\mathsf{const}_\star} : (x = y) \to (\star = \star))\right)$$

because then we can use this function f applied to one of the two terms $x, y : A$ to get the data we want. But now our new goal allows us to assume we have a term $a : A$ and so it follows from our assumption that A is contractible. Thus $\mathsf{const}_\star : A \to \mathbb{1}$ is an equivalence and in particular an embedding. This proves (iv).

Finally, $\mathsf{const}_\star : A \to \mathbb{1}$ is an embedding, then the types $(x = y)$ and $(\star = \star)$ are equivalent. The latter type is contractible so the former must be as well. This proves that (iv) implies (i). □

A useful feature of propositions is that logical equivalences become equivalences:

Proposition. *For propositions P and Q*

$$(P \simeq Q) \leftrightarrow (P \leftrightarrow Q).$$

Proof. Clearly we have $(P \simeq Q) \to (P \leftrightarrow Q)$ so the content is in the converse. Given $f : P \to Q$ and $g : Q \to P$ we obtain homotopies $f \circ g \sim \mathsf{id}$ and $g \circ f \sim \mathsf{id}$ by the fact that any two elements in P and Q can be identified. Thus, f and g are equivalence inverses. □

Subtypes. Now that we know about propositions we can say that a type family $P\colon A \to \mathcal{U}$ is a predicate if for each $a : A$, $P(a)$ is a proposition. In other words, predicates are type families $P\colon A \to \text{Prop}_{\mathcal{U}}$. Other terminology is commonly in use in this situation.

defn. A type family B over A is a **subtype** of A if for each $x : A$, $B(x)$ is a proposition. In this situation we say that $B(x)$ is a **property** of $x : A$.

We'll show that for subtypes B over A the map $\mathsf{pr}_1 \colon \Sigma_{x:A} B(x) \to A$ is an embedding. It follows, then, that $(x, y) = (x', y')$ if and only if $x =_A x'$. To prove this first observe:

Lemma. *Suppose $e : A \simeq B$. Then*

$$\text{is-prop}A \leftrightarrow \text{is-prop}B.$$

Proof. Since e is an equivalence, e is an embedding, meaning that $\mathsf{ap}_e : (x =_A y) \to (e(x) =_B e(y))$ is an equivalence. Now if B is contractible then $(e(x) =_B e(y))$ is contractible which then implies that $(x =_A y)$ is contractible. Thus $\text{is-prop}(B) \to \text{is-prop}(A)$. The converse is proven similarly using the inverse equivalence to e. □

Theorem. *For $f : A \to B$ the following are logically equivalent:*

(i) f is an embedding.
(ii) $\text{fib}_f(b)$ is a proposition for all $b : B$

Proof. By the fundamental theorem of identity types, f is an embedding if and only if $\Sigma_{x:A} f(x) =_B f(y)$ is contractible for each $y : A$. Thus f is an embedding iff $\text{fib}_f(f(y))$ is contractible for each $y : A$. If $b : B$ and $p : f(y) = b$ then transport defines an equivalence

$$\text{fib}_f(f(y)) \simeq \text{fib}_f(b).$$

Thus f is an embedding iff $\text{fib}_f(b)$ is contractible for each $b : B$ equipped with $p : f(y) = b$ for some $y : A$. This latter condition may be re-expressed as

$$\text{fib}_f(b) \to \text{is-contr}(\text{fib}_f(b)),$$

which is logically equivalent to the assertion that each $\text{fib}_f(b)$ is a proposition. □

Corollary. *For any family $B : A \to \mathcal{U}$ the following are logically equivalent:*

(i) $\mathsf{pr}_1 \colon \Sigma_{x:A} B(x) \to A$ is an embedding.
(ii) $B(x)$ is a proposition for each $x : A$.

Proof. Since $\text{fib}_{\mathsf{pr}_1}(x) \simeq B(x)$ this follows immediately from the previous theorem. □

Sets.

defn. A type A is a **set** if its identity types are propositions:

$$\text{is-set}(A) :\equiv \prod_{x,y:A} \text{is-prop}(x = y).$$

ex. The type of natural numbers is a set, since we have $(m = n) \simeq \text{Eq}_{\mathbb{N}}(m, n)$ and, by induction, the latter types are propositions.

Theorem. *For a type A the following are logically equivalent:*

(i) *A is a set.*
(ii) *A satisfies **axiom K**: that is, A comes with a term in the type*

$$\text{axiom-K}(A) :\equiv \prod_{x:A} \prod_{p:x=x} \mathtt{refl}_x = p.$$

Proof. If A is a set then $x = x$ is a proposition so any two terms in it can be identified.

Conversely, if axiom-K holds then for any $p, q : x = y$ we can identify $p \cdot q^{-1}$ and $\mathtt{refl}_x$ and it follows that $p = q$. This proves that $x = y$ is a proposition so A must be a set. □

The following result can be used to prove that a type A is a set.

Theorem. *Let A be a type and suppose $R \colon A \to A \to \mathcal{U}$ satisifes:*

(i) *Each $R(x, y)$ is a proposition.*
(ii) *R is reflexive, witnessed by $\rho : \prod_{x:A} R(x, x)$.*
(iii) *There is a map $R(x, y) \to (x = y)$ for all $x, y : A$.*

Then any family of maps $\prod_{x,y:A}(x = y) \to R(x, y)$ is a family of equivalences and A must be a set.

Proof. By hypothesis we have terms $f : \prod_{x,y:A} R(x, y) \to (x = y)$ and also $\mathtt{path\text{-}ind}(\rho) : \prod_{x,y}(x = y) \to R(x, y)$. Since each $R(x, y)$ is a proposition we have a homotopy $\mathtt{path\text{-}ind}(\rho)(x, y) \circ f(x, y) \sim \mathtt{id}_{R(x,y)}$ proving that $R(x, y)$ is a retract of $x = y$. Thus, $\Sigma_{y:A} R(x, y)$ is a retract of $\Sigma_{y:A} x = y$. Since the latter type is contractible the former must be too. Thus any family of maps $\prod_{y:A}(x = y) \to R(x, y)$ is a family of equivalences (since its totalization is a map between contractible types and thus an equivalence).

But now we know that the identity types of A are propositions so A must be a set. □

Recall a type A has decidable equality if the identity type $x =_A y$ is decidable for every $x, y : A$, meaning

$$\prod_{x,y:A}(x = y) + \neg(x = y).$$

Theorem (Hedberg)**.** *Any type with decidable equality is a set.*

Proof. Let $d : \prod_{x,y:A}(x = y) + \neg(x = y)$ be a witness to the fact that A has decidable equality and let $\mathcal{U}$ be a universe containing A.

Define a type family $R'(x, y) : ((x = y) + \neg(x = y)) \to \mathcal{U}$ by

$$R'(x, y, \mathtt{inl}(p)) :\equiv \mathbb{1} \quad R'(x, y, \mathtt{inr}(p)) :\equiv \varnothing.$$

Note that this is a family of propositions. Now define $R(x, y) :\equiv R'(x, y, d(x, y))$. This defines a family of propositions $R : A \to A \to \mathcal{U}$. This is a reflexive binary relation so the apply the previous theorem to conclude that A is a set we must only show that R implies identity.

Since R is defined to be an instance of R' it suffices to construct a function for each $q : (x = y) + \neg(x = y)$ that proves $f(q) : R'(x, y, q) \to (x = y)$. We have this by

$$f(\mathtt{inl}(p, r)) :\equiv p \qquad f(\mathtt{inr}(p), r) :\equiv \mathtt{ex\text{-}falso}(r).$$ □

October 18: General Truncation Levels & Function extensionality

General truncation levels. So far we have defined:

$$\begin{aligned} \text{is-contr}(A) &:\equiv \Sigma_{a:A} \prod_{x:A} a = x \\ \text{is-prop}(A) &:\equiv \prod_{x,y:A} \text{is-contr}(x = y) \\ \text{is-set}(A) &:\equiv \prod_{x,y:A} \text{is-prop}(x = y) \end{aligned}$$

These define the first few layers of the hierarchy of truncation levels. This hierarchy starts at level -2 with the contractible types and continues at level -1 with the propositions. This makes level 0 the sets, which are typically thought of as "0-dimensional."

Let $\mathbb{T}$ be the inductive type with constructors $-2_{\mathbb{T}} : \mathbb{T}$ and $\mathsf{succ}_{\mathbb{T}} : \mathbb{T} \to \mathbb{T}$. The natural inclusion $i\colon \mathbb{N} \to \mathbb{T}$ is defined recursively by $i(0_{\mathbb{N}}) :\equiv \mathsf{succ}_{\mathbb{T}}(\mathsf{succ}_{\mathbb{T}}(-2_{\mathbb{T}}))$ and $i(\mathsf{succ}_{\mathbb{N}}(n)) :\equiv \mathsf{succ}_{\mathbb{T}}(i(n))$. We abbreviate by writing $-2, -1, 0, 1, 2, \ldots$ for the first few terms of $\mathbb{T}$ when the context is clear.

defn. Define is-trunc $: \mathbb{T} \to \mathcal{U} \to \mathcal{U}$ recursively by

$$\text{is-trunc}_{-2}(A) :\equiv \text{is-contr}(A) \qquad \text{is-trunc}_{k+1}(A) :\equiv \prod_{x,y:A} \text{is-trunc}_k(x =_A y).$$

When $\text{is-trunc}_k(A)$ holds we say A is k-**truncated** or is a k-**type**. You can prove, inductively in $k : \mathbb{T}$, that this is logically independent of the universe being used to define $\text{is-trunc}_k(A)$.

For $k \geqslant 0$, we may also say that A is a **proper** k-**type** if $\text{is-trunk}_k(A)$ holds but $\text{is-trunk}_{k-1}(A)$ does not.

defn. A map $f\colon A \to B$ is k-**truncated** if its fibers are k-truncated.

Given a universe $\mathcal{U}$, we may also define a universe of k-truncated types by

$$\mathcal{U}^k :\equiv \Sigma_{X:\mathcal{U}} \text{is-trunc}_k(X).$$

The truncation levels are successively contained in one another:

Proposition. *If A is a k-type then A is also a $k+1$-type.*

Proof. We use induction on $k : \mathbb{T}$. In the base case, we have shown already that contractible types are propositions. For the inductive step, note that if any k-type is a $k+1$-type then this applies to show that the identity types of a $k+1$-type, which are known to be k-types, are also $k+1$-types. This proves that any $k+1$-type is a $k+2$-type. □

In particular:

Corollary. *If A is a k-type its identity types are also k-types.*

General truncation levels are stable under equivalence:

Proposition. *If $e : A \simeq B$ and B is a k-type so is A.*

Proof. We know this for contractible types, which is the base case. For the inductive step, $e : A \simeq B$ provides an equivalence $\mathsf{ap}_e : (x = y) \to (e(x) = e(y))$ for any $x, y : A$. If B is a $k+1$-type its identity types are k-types so the inductive hypothesis implies that $(x = y)$ is also a k-type. This proves that A is a $k+1$-type. □

A similar argument shows:

Corollary. *If $f\colon A \to B$ is an embedding and B is a $k+1$-type, then so is A.*

Our final theorem generalizes the result that shows that a map is an embedding if and only if its fibers are propositions.

Theorem. *For $f\colon A \to B$ the following are logically equivalent:*

(i) f is $(k+1)$-truncated.
(ii) For each $x, y : A$, $\mathsf{ap}_f : (x = y) \to (f(x) = f(y))$ is k-truncated.

Proof. Both directions use the characterization of identity types of fibers:

$$((x, p) =_{\text{fib}_f(b)} (y, q)) \simeq \Sigma_{\alpha:x=y} p = \mathsf{ap}_f(\alpha) \cdot q.$$

The first statement is about identity types of fibers so consider $s, t : \text{fib}_f(b)$. We claim there is an equivalence

$$(s = t) \simeq \text{fib}_{\mathsf{ap}_f}(\mathsf{pr}_2(s) \cdot \mathsf{pr}_2(t)^{-1}).$$

By Σ-induction we can construct this for pairs $(x, p), (y, q) : \text{fib}_f(b)$ for which we calculate

$$\begin{aligned}((x, p) = (y, q)) &\simeq \Sigma_{\alpha:x=y} p = \mathsf{ap}_f(\alpha) \cdot q \\ &\simeq \Sigma_{\alpha:x=y} \mathsf{ap}_f(\alpha) \cdot q = p \\ &\simeq \Sigma_{\alpha:x=y} \mathsf{ap}_f(\alpha) = p \cdot q^{-1} \\ &=: \text{fib}_{\mathsf{ap}_f}(p \cdot q^{-1}).\end{aligned}$$

It follows that if $\mathtt{ap}_f$ is k-truncated then so each identity type $(s = t)$ is equivalent to a k-truncated type and thus f is $k+1$-truncated.

For the converse, we have an equivalence between $\mathrm{fib}_{\mathtt{ap}_f}(p)$ and the identity type $(x,p) = (y, \mathtt{refl}_{f(y)})$ between terms in $\mathrm{fib}_f(f(y))$. So if f is $(k+1)$-truncated these fibers are $k+1$-truncated and thus their identity types are k-types. This proves that the fiber $\mathrm{fib}_{\mathtt{ap}_f}(p)$ is k-truncated. □

Function extensionality. The function extensionality principle characterizes the identity type of an arbitrary dependent function type, asserting that the type $f = g$ of identifications between dependent functions $f, g : \prod_{x:A} B(x)$ is equivalent to the type of homotopies $f \sim g$. It has several equivalent forms:

Proposition. *For a type family $B : A \to \mathcal{U}$ the following are logically equivalent:*

*(i) The **function extensionality principle holds** for $f, g : \prod_{x:A} B(x)$: the family of maps*

$$\mathtt{htpy\text{-}eq} : (f = g) \to (f \sim g)$$

defined by sending `refl` *to* `refl-htpy` *is a family of equivalences.*

(ii) For any $f : \prod_{x:A} B(x)$, the total space

$$\Sigma_{g:\prod_{x:A} B(x)} f \sim g$$

is contractible.

*(iii) The principle of **homotopy induction** holds: for any family of types P depending on $f, g : \prod_{x:A} B(x)$ and $H : f \sim g$ the evaluation function*

$$\mathtt{ev}\colon \left(\prod_{f,g:\prod_{x:A} B(x)} \prod_{H:f\sim g} P(f,g,H)\right) \to \prod_{f:\prod_{x:A} B(x)} P(f,f,\mathtt{refl\text{-}htpy}_f)$$

has a section.

Proof. This follows by applying the fundamental theorem of identity types to the type $\prod_{x:A} B(x)$, term $f : \prod_{x:A} B(x)$, and type family $g : \prod_{x:A} B(x) \vdash f \sim g$ `type` . □

A fourth equivalent condition is more surprising because it appears to express only a weak function extensionality principle.

Theorem. *For any universe $\mathcal{U}$ the following are logically equivalent:*

*(i) The **function extensionality principle** holds in $\mathcal{U}$: for any type family B over A and dependent functions the map*

$$\mathtt{htpy\text{-}eq} : (f = g) \to (f \sim g)$$

is an equivalence.

*(ii) The **weak function extensionality principle** holds in $\mathcal{U}$: for any type family B over A one has*

$$\left(\prod_{x:A} \text{is-contr}(B(x))\right) \to \text{is-contr}\left(\prod_{x:A} B(x)\right).$$

Proof. Assume (i) and suppose each fiber $B(x)$ is contractible with center of contraction $c(x)$ and contracting homotopy $C_x : \prod_{y:B(x)} c_x = y$. Define $c = \lambda x.c(x)$ to be the center of contraction of $\prod_{x:A} B(x)$. For the contraction we require a term of type

$$\prod_{f:\prod_{x:A} B(x)} c = f.$$

By function extensionality we have a map $(c \sim f) \to (c = f)$ so it suffices to construct a term of type $c \sim f :\equiv \prod_{x:A} c(x) = f(x)$ and $\lambda x.C_x(f(x))$ is just such a term.

For the converse, assume (ii). By the previous result it suffices to show that the type

$$\Sigma_{g:\prod_{x:A} B(x)} f \sim g$$

is contractible. Note we have a section-retraction pair:

$$\left(\Sigma_{g:\prod_{x:A} B(x)} f \sim g\right) \xrightarrow{s} \left(\prod_{x:A} \Sigma_{y:B(x)} f(x) = y\right) \xrightarrow{r} \left(\Sigma_{g:\prod_{x:A} B(x)} f \sim g\right)$$

defined by

$$s :\equiv \lambda(g,H).\lambda x.(g(x), H(x)) \quad \text{and} \quad r :\equiv \lambda p.(\lambda x.\mathtt{pr}_1(p(x)), \lambda x.\mathtt{pr}_2(p(x))).$$

The composite is homotopic to the identity function by the computation rules for Σ and Π-types. Here the central type is a product of contractible types so must be contractible by (ii). Since retracts of contractible types are contractible, the claim follows. □

Henceforth, we will assume the function extensionality principle as an axiom:

Axiom (function extensionality)**.** For any type family B over A and any pair of dependent functions $f, g : \Pi_{x:A} B(x)$ the map

$$\mathtt{htpy\text{-}eq} : (f = g) \to (f \sim g)$$

is an equivalence, with inverse `eq-htpy`.

That is, we add the following rule to type theory:

$$\frac{\Gamma, x : A \vdash B(x)\ \mathtt{type} \qquad \Gamma \vdash f : \Pi_{x:A} B(x) \qquad \Gamma \vdash g : \Pi_{x:A} B(x)}{\Gamma \vdash \mathtt{funext} : \text{is-equiv}(\mathtt{htpy\text{-}eq}_{f,g})}$$

There are myriad consequences of the function extensionality axiom. Firstly:

Theorem. *For any type family B over A one has*

$$(\Pi_{x:A} \text{is-trunc}_k(B(x))) \to \text{is-trunc}_k(\Pi_{x:A} B(x)).$$

Proof. The theorem states that k-types are closed under arbitrary dependent products. We prove this by induction on $k \geqslant -2$. The base case is the weak function extensionality principle.

For the inductive step assume k-types are closed under products and consider a family B of $(k+1)$-types. To show that $\Pi_{x:A} B(x)$ is $(k+1)$-truncated we must show that $f = g$ is k-truncated for every $f, g : \Pi_{x:A}$. By function extensionality this is equivalent to the type $f \sim g :\equiv \Pi_{x:A} f(x) = g(x)$ which is defined to be a dependent product of k-truncated types and thus is k-truncated by hypothesis. Since k-truncated types are closed under equivalences, the result follows. □

For a non-dependent family we conclude that:

Corollary. *Suppose B is a k-type. Then the type of functions $A \to B$ is a k-type for any type A.*

In particular, $\neg A$ is a proposition for any type A!

October 20: Universal properties

We can understand the function extensionality axiom as providing a characterization of the identity type of a Π-type: up to equivalence, for $f, g : \Pi_{x:A} B(x)$, the identity type $f = g$ is equivalent to the type of homotopies $f \sim g$.

The type theoretic axiom of choice. First, observe that Π-types distribute over Σ-types by the type theoretic axiom of choice.

Theorem. *For any family of types $x : A, y : B(x) \vdash C(x, y)$ `type` the map*

$$\mathtt{choice} : \left(\Pi_{x:A} \Sigma_{y:B(x)} C(x, y)\right) \to \left(\Sigma_{f:\Pi_{x:A} B(x)} \Pi_{x:A} C(x, f(x))\right)$$

defined by

$$\mathtt{choice}(h) :\equiv (\lambda x.\mathtt{pr}_1(h(x)), \lambda x.\mathtt{pr}_2(h(x)))$$

is an equivalence.

Consequently, whenever we have types A and B and a type family C over B there is an equivalence

$$\left(A \to \Sigma_{y:B} C(y)\right) \simeq \left(\Sigma_{f:A\to B} \Pi_{x:A} C(f(x))\right)$$

Proof. Define the inverse map $\mathtt{choice}^{-1}$ by

$$\mathtt{choice}^{-1}(f, g) :\equiv \lambda x.(f(x).g(x)).$$

For the first homotopy it suffices to define an identification $\mathtt{choice}(\mathtt{choice}^{-1}(f, g)) = (f, g)$. The left-hand side computes to

$$\mathtt{choice}(\mathtt{choice}^{-1}(f, g)) \doteq \mathtt{choice}(\lambda x.(f(x).g(x))) \doteq (\lambda x.f(x), \lambda x.g(x))$$

which is definitely equal to the right-hand side by the computation rules for function types.

For the second homotopy, we require an identification $\mathtt{choice}^{-1}(\mathtt{choice}(h)) = h$. The left-hand side computes to

$$\mathtt{choice}^{-1}(\lambda x.\mathtt{pr}_1(h(x)), \lambda x.\mathtt{pr}_2(h(x))) \doteq \lambda x.(\mathtt{pr}_1(h(x)), \mathtt{pr}_2(h(x))).$$

We do not have a definitional equality relating $h(x)$ and $(\mathtt{pr}_1(h(x)), \mathtt{pr}_2(h(x)))$ but in our characterization of the identity type of Σ-types we do have an identification between them called $\mathtt{eq\text{-}pair}(\mathtt{refl}, \mathtt{refl})$. By function extensionality, the homotopy $\lambda x.\mathtt{eq\text{-}pair}(\mathtt{refl}, \mathtt{refl}) : \mathtt{choice}^{-1}(\mathtt{choice} \sim h$ can be turned into an identification and thus a homotopy $\mathtt{choice}^{-1} \circ \mathtt{choice} \sim \mathtt{id}$. □

Universal properties. More generally, the function extensionality axiom allows us to prove universal properties, which characterize maps out of or into a given type, and characterize that type up to equivalence. Some examples follow.

In our first example, we consider the maps out of Σ-types. The universal property states that the map

$$\mathtt{ev\text{-}pair} : ((\Sigma_{x:A} B(x)) \to C) \to (\Pi_{x:A} B(x) \to C)$$

given by $f \mapsto \lambda x.\lambda y.f(x, y)$ is an equivalence for any type C. But the analogous result is true for type familyes C that depend on the type $\Sigma_{x:A} B(x)$ so we prove the result in that form.

Theorem (universal property of Σ-types)**.** *Let B be a type family over A and let C be a type family over $\Sigma_{x:A} B(x)$. Then the map*

$$\mathtt{ev\text{-}pair} : \left(\Sigma_{z:\Sigma_{x:A} B(x)} C(z)\right) \to \left(\Pi_{x:A} \Pi_{y:B(x)} C(x, y)\right)$$

given by $f \mapsto \lambda x.\lambda y.f(x, y)$ is an equivalence

Proof. The inverse map is given by the induction principle for Σ-types:

$$\mathtt{ind}_\Sigma : \left(\Pi_{x:A} \Pi_{y:B(x)} C(x, y)\right) \to \left(\Sigma_{z:\Sigma_{x:A} B(x)} C(z)\right).$$

By the computation rule for Σ types, we have the homotopy

$$\mathtt{refl\text{-}htpy} : \mathtt{ev\text{-}pair} \circ \mathtt{ind}_\Sigma \sim \mathtt{id},$$

which shows that $\mathtt{ind}_\Sigma$ is a section of $\mathtt{ev\text{-}pair}$.

Function extensionality is used to construct the other homotopy. To define a homotopy $\mathtt{ind}_\Sigma \circ \mathtt{ev\text{-}pair} \sim \mathtt{id}$ requires identifications $\mathtt{ind}_\Sigma(\lambda x.\lambda y.f(x, y)) = f$. By function extensionality it suffices to show that

$$\Pi_{z:\Sigma_{x:A} B(x)} \mathtt{ind}_\Sigma(\lambda x.\lambda y.f(x, y))(t) = f(t).$$

By Σ-induction it suffices to prove this for pairs in which case req require identifications

$$\mathtt{ind}_\Sigma(\lambda x.\lambda y.f(x, y))(a, b) = f(a, b),$$

but this holds definitionally by the computation rule for Σ types. □

In the non-dependent case we have as a corollary:

Corollary. *For types A and B and C,*

$$\mathtt{ev\text{-}pair} : (A \times B \to C) \to (A \to B \to C)$$

given by $f \mapsto \lambda a.\lambda b.f(a, b)$ is an equivalence.

The universal property of identity types. The universal property for identity types can be understood as an (undirected) type theoretic version of the Yoneda lemma. In the most familiar case, when B is a type family over A, it says that the map

$$\mathtt{ev\text{-}refl} : (\Pi_{x:A}(a = x) \to B(x)) \to B(a)$$

given by $f \mapsto f(a, \mathtt{refl}_a)$ is an equivalence. As before, though, it generalizes to a dependent version of the undirected Yoneda lemma, where the type family B is allowed to depend on $x : A$ and $p : a = x$.

Theorem. *Consider a type A, a term $a : A$, and a type family $B(x, p)$ over $x : A$ and $p : a = x$. Then the map*

$$\mathtt{ev\text{-}refl} : \left(\Pi_{x:A} \Pi_{p:a=x} B(x, p)\right) \to B(a, \mathtt{refl}_a)$$

defined by $f \mapsto f(a, \mathtt{refl}_a)$ is an equivalence.

Proof. The inverse map is

$$\texttt{path-ind}_a : B(a, \texttt{refl}_a) \to \left(\Pi_{x:A}\Pi_{p:a=x}B(x,p)\right),$$

which is a section by the computation rule of the path induction principle.

For the other homotopy $\texttt{path-ind}_a \circ \texttt{ev-refl} \simeq \texttt{id}$ let $f : \Pi_{x:A}\Pi_{p:a=x}B(x,p)$. To prove that $\texttt{path-ind}_a(f(a,\texttt{refl}_a)) = f$ we apply function extensionality twice so that it suffices to show that

$$\Pi_{x:A}\Pi_{p:a=x}\texttt{path-ind}_a(f(a,\texttt{refl}_a),x,p) = f(x,p).$$

This follows from path induction on p since $\texttt{path-ind}_a(f(a,\texttt{refl}_a),a,\texttt{refl}_a) \doteq f(a,\texttt{refl}_a)$ by the computation rule for path induction. □

Composing with equivalences. Another useful consequence is the fact that $f : A \to B$ is an equivalence if and only if precomposing with f is an equivalence.

Theorem. *For any map $f\colon A \to B$ the following are logically equivalent:*

(i) *f is an equivalence*
(ii) *For any type family P over B the map*

$$(\Pi_{b:B}P(b)) \to \left(\Pi_{a:A}P(f(a))\right)$$

given by $h \mapsto h \circ f$ is an equivalence.
(iii) *For any type C the map*

$$(B \to C) \to (A \to C)$$

given by $g \mapsto g \circ f$ is an equivalence.

Proof. (ii) immediately implies (iii) by choosing a constant family.

Assuming (iii) we can take $C = A$ and use the fact that the fibers of the equivalence

$$- \circ f : (B \to A) \to (A \to A)$$

are contractible to find a point $(h, H) : \text{fib}_{-\circ f}(\texttt{id}_A) \doteq \Sigma_{h:B\to A} h \circ f = \texttt{id}_A$. To see that h is also a section of f we choose $C = B$ and use the fiber of the equivalence

$$- \circ f : (B \to B) \to (A \to B)$$

over f. We have $(\texttt{id}_B, \texttt{refl}_f)$ in this fiber but also the point $(f \circ h, fH)$ where fH is the name for the identification derived from the whiskered homotopy $f \cdot H : f \circ h \circ f \sim f$. Since the ifber is contractible, we must have an identification $f \circ h = \texttt{id}_B$ as desired.

Thus, it remains to prove that (i) implies (ii) which is the hard part. The first step is to promote the equivalence f to a coherently invertible equivalence, involving $g : B \to A$, homopies G and H, and a higher homotopy $K : G \cdot f \sim f \cdot H$. We leave the details to [R, 13.4.1]. □

The strong induction principle of $\mathbb{N}$. A final application of function extensionality is to prove the strong induction principle for the natural numbers. We give the statement and leave the proof to [R, §13.5]. Function extensionality is needed to give the computation rules.

Theorem. *Consider a type family P over $\mathbb{N}$ with $p_0 : P(0)$ and*

$$p_S : \Pi_{n:\mathbb{N}}\left(\Pi_{m:\mathbb{N}}(m \leqslant n) \to P(m)\right) \to P(n+1).$$

Then there is a dependent function

$$\texttt{strong-ind}_{\mathbb{N}}(p_0, p_S) : \Pi_{n:\mathbb{N}}P(n)$$

so that $\texttt{strong-ind}_{\mathbb{N}}(p_0, p_S, 0) = 0$ *and*

$$\texttt{strong-ind}_{\mathbb{N}}(p_0, p_S, n+1) = p_S(n, \lambda m.\lambda p.\texttt{strong-ind}_{\mathbb{N}}(p_0, p_s, m)).$$

October 25: Propositional truncation

There is a distinction made in mathematics between *properties* and extra *structure*. For instance, we can ask whether a given function $f\colon A \to B$ is surjective or not. In type theory, this is the case just when

$$\Pi_{b:B}\Sigma_{a:A} f(a) = b.$$

But we've seen that there is an equivalence of types

$$\left(\Pi_{b:B}\Sigma_{a:A} f(a) = b\right) \simeq \left(\Sigma_{s:A\to B}\Pi_{b:B} f(s(b)) = b\right).$$

In particular, the data provided by a term in this type provides an explicit section $s : B \to A$ which is additional structure.

To correctly capture the mere property of being surjective, we need a way to assert the *proposition* that a type is inhabited without providing the *data* of a specific inhabitant. The proposition that a type A is inhabited is called the **propositional truncation** of A.

The universal property of propositional truncations. The propositional truncation of a type A is a type $\|A\|$ equipped with a map $\eta\colon A \to \|A\|$ with a universal property to be described. The map ensures that if $a : A$ then the proposition $\|A\|$ that A is inhabited holds.

defn. Let A be a type and let $f\colon A \to P$ be a map whose codomain is a proposition. We say that f is a **propositional truncation** of A if for every proposition Q the map

$$- \circ f : (P \to Q) \to (A \to Q)$$

is an equivalence.

This definition describes the **universal property of the propositional truncation**. It can be reformulated as follows. Note the fiber of the map

$$- \circ f : (P \to Q) \to (A \to Q)$$

over $g : A \to Q$ is the type

$$\sum_{h:P\to Q} h \circ f = g.$$

Thus, f satisfies the universal property of the propositional truncation if and only if these fibers are contractible, meaning that for each map $g : A \to Q$ into a proposition there is a unique map $h : P \to Q$ for wich $h \circ f = g$, a circumstance we might summarize by saying that every map $g : A \to Q$ into a proposition **extends uniquely along** f as indicated

$$\begin{array}{ccc} A & & \\ {\scriptstyle f}\downarrow & \searrow^{g} & \\ P & \xrightarrow[\exists!]{} & Q \end{array}$$

Remark. By (weak) function extensionality, the types $(P \to Q)$ and $(A \to Q)$ are propositions, since Q is a proposition. Recall that equivalences between propositions are just logical equivalences. Thus, to prove that

$$- \circ f : (P \to Q) \to (A \to Q)$$

is an equivalence, it suffices to construct a function

$$(A \to Q) \to (P \to Q)$$

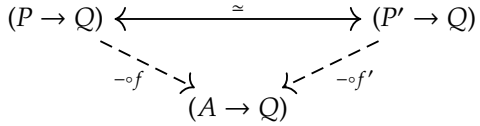
for every proposition Q.

Proposition. *Let A be a type and consider two maps $f\colon A \to P$ and $f'\colon A \to P'$ into propositions. If any two of the following hold so does the third:*

(i) f is a propositional truncation
(ii) f' is a propositional truncation
(iii) There is a (unique) equivalence $P \simeq P'$, commuting with the maps from A.

Proof. Given (i) and (ii) the universal properties induce maps $P \to P'$ and $P' \to P$ under A that are necessarily an inverse equivalence. This proves (iii).

Given (iii), we have an equivalence between $(P \to Q)$ and $(P' \to Q)$ and moreover the equivalence $P \leftrightarrow P'$ commutes with the maps from A, since by function extensionality any two terms in the types $A \to P$ and $A \to P'$ must be identifiable. But now by the 2-of-3 property for equivalences, if either of the two dashed maps below is an equivalence both are.



Thus when (iii) holds (i) holds if and only if (ii) holds. □

It is tempting to think that a type is inhabited if and only if it is non-empty. That it, it is tempting to wonder whether the canonical map $\lambda x \to \mathsf{ev}_x : A \to \neg\neg A$ is a propositional truncation, since after all $\neg\neg A$ is always a proposition. One can verify that any map $A \to \neg\neg Q$ extends to a map $\neg\neg A \to \neg\neg Q$ proving that the map

$$(\neg\neg A \to \neg\neg Q) \to (A \to \neg\neg Q)$$

is an equivalence. However, this universal property with respect to doubly negated propositions cannot be extended to general propositions. Indeed, propositional truncations are not guaranteed to exist in Martin Löf's dependent type theory; see "Notions of anonymous existence in Martin-Löf type theory" by Altenkirch, Coquand, Escardó, and Kraus for a discussion. However, if we add a new rule to the type theory, we can guarantee their existence.

Propositional truncations as higher inductive types. The propositional truncation of a general type A can be constructed as an instance of something called a **higher inductive type**. Higher inductive types are similar to ordinary inductive types but have an additional feature that constructors can be used to generate identifications. As in ordinary inductive types, there are **point constructors** which introduce new terms, but now these are supplemented by **path constructors** which introduce new identifications.

In the case of the propositional truncation, for any type A, $\|A\|$ is the type given by one point constructor $\eta\colon A \to \|A\|$ and one path constructor $\alpha\colon \prod_{x,y:\|A\|} x = y$. It follows immediately from the existence of the term α that $\|A\|$ is a proposition.

The induction principle for the propositional truncation tells us how to construct terms $h : \prod_{t:\|A\|} Q(t)$ for any family of types (not only families of propositions). It says that for any family of types Q over $\|A\|$ if we have

$$f : \prod_{a:A} Q(\eta(a))$$

and if we can construct identifications $\mathsf{tr}_Q(\alpha(x,y),u) = v$ for all $x, y : \|A\|$ and all $u : Q(x)$ and $v : Q(y)$ then we obtain a dependent function $h : \prod_{t:\|A\|} Q(t)$ equipped with a homotopy $h \circ \eta \sim f$. This homotopy should be thought of as a homotopical version of the computation rule for ordinary inductive types.

Remark. Although the induction principle for propositional truncations does not a priori require that the type family is a family of propositions, once the second required term

$$\beta : \prod_{x,y:\|A\|} \prod_{u:Q(x)} \prod_{v:Q(y)} \mathsf{tr}_Q(\alpha(x,y),u) = v$$

exists it follows that Q is a family of propositions. Recall that transporting along a path defines an equivalence and in particular an embedding. Thus

$$(\mathsf{tr}_Q(\alpha(x,y),u) = \mathsf{tr}_Q(\alpha(x,y),w)) \simeq (u = w)$$

for any $u, w : Q(x)$. Taking $v = \mathsf{tr}_Q(\alpha(x,y),w))$, we see that β defines a term in the left-hand type so it follows that any $u, w : Q(x)$ are identifiable. Thus $Q(x)$ must be a proposition.

Since the induction principle of the propositional truncation is only applicable in families of propositions there are no interesting computation rules: there was no need to mention the homotopy $h \circ \eta \sim f$. After all any identification in a proposition just holds!

Theorem. *The map $\eta\colon A \to \|A\|$ satisfies the universal property of propositional truncation.*

Proof. It suffices to construct a map

$$(A \to Q) \to (\|A\| \to Q)$$

for any proposition Q, which we do by the induction principle of propositional truncation. To construct a term in $\|A\| \to Q$ we need a term $f : A \to Q$ and also have to prove that $\mathtt{tr}_Q(\alpha(x,y),u) = v$ holds for any $u, v : Q$ and any $x, y : \|A\|$. But Q is a proposition so this is automatic. Thus the map that sends $f : A \to Q$ to the induced function $\|A\| \to Q$ defines the required function $(A \to Q) \to (\|A\| \to Q)$. □

Next we show that $\|-\|$ acts functorially on functions.

Proposition. *For any pair of types A and B there is a map*

$$\|-\| \colon (A \to B) \to (\|A\| \to \|B\|)$$

satisfying $\|\mathtt{id}\| \sim \mathtt{id}$ and $\|g \circ f\| \sim \|g\| \circ \|f\|$.

Proof. For any $f : A \to B$, $\|f\|$ may be defined to be the unique extension

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle\eta}\downarrow & & \downarrow{\scriptstyle\eta} \\ \|A\| & \xrightarrow[\|f\|]{} & \|B\| \end{array}$$

Note by definition that $\mathtt{id}_{\|A\|}$ and $\|g\| \circ \|f\|$ are similarly extensions of $\mathtt{id}_A$ and $g \circ f$ along η, and thus must agree up to homotopy with $\|\mathtt{id}_A\|$ and $\|g \circ f\|$ by uniqueness. □

Logic in type theory. Propositional truncations can be used to extend the interpretation of logic in type theory via the Curry-Howard correspondence. This refines our previous discussion by replacing structures with properties.

defn. Given two propositions P and Q we define their **disjunction** to be

$$P \vee Q :\equiv \|P + Q\|.$$

defn. Given a family of propositions P over a type A, define

$$\exists_{x:A} P(x) :\equiv \|\Sigma_{x:A} P(x)\|.$$

One can verify the expected logical equivalences involving these notions. For instance:

Proposition. *For any family of propositions P over A there is a dependent function*

$$\epsilon : \Pi_{a:A} \left(P(a) \to \exists_{x:A} P(x)\right).$$

Furthermore, for any proposition Q we have

$$((\exists_{x:A} P(x)) \to Q) \leftrightarrow (\Pi_{x:A} P(x) \to Q).$$

Proof. Define $\epsilon(a,p) :\equiv \eta(a,p)$. Now consider the following composition of maps

$$((\exists_{x:A} P(x)) \to Q) \to ((\Sigma_{x:A} P(x)) \to Q) \to (\Pi_{x:A} P(x) \to Q).$$

The first map is an equivalence by the uinversal property of the propositional truncation while the second is an equivalence by the universal property of Σ-types. □

Mapping from propositional truncations into sets. In general it is tricky to map out of propositional truncations into general types but some tricks may help. To define a map $\|A\| \to X$ one could search for a type family P over X so that $\Sigma_{x:X} P(x)$ is a proposition. Then the universal property of propositional truncation can be used to define a map $\|A\| \to \Sigma_{x:X} P(x)$ and compositing with $\mathtt{pr}_1$ then defines map to X.

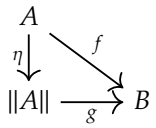
When X is a set there is another strategy. The propositional truncation $\|A\|$ can be thought of as a quotient of the type A by the equivalence relation that identifies any two terms in A together. We will prove that to extend a map $f : A \to X$ into a set to a map $\|A\| \to X$ it suffices to define identifications $f(x) = f(y)$ for all $x, y : A$.

defn. A map $f\colon A \to B$ is **weakly constant** if it comes with a term in the type

$$\Pi_{x,y:A} f(x) = f(y).$$

If B has a term b, then weakly constant maps are homotopic to constant maps $\mathtt{const}_b$ but in general there is no requirement for B to have any terms.

Lemma. *Consider a commutative triangle where B is an arbitrary type*



then g is weakly constant.

Kraus observed that any weakly constant map $f : A \to B$ into a set B extends uniquely to a map $\|A\| \to B$. Thus to define a map $\|A\| \to B$ into a set it suffices to define $f : A \to B$ and show that it is weakly constant.

Theorem (Kraus)**.** *For any type A and set B the map*

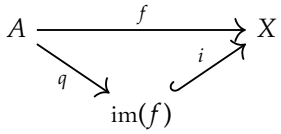
$$(\|A\| \to B) \to \Sigma_{f:A\to B}\Pi_{x,y:A} f(x) = f(y)$$

given by $g \mapsto (g \circ \eta, \lambda x.\lambda y.\mathsf{ap}(\alpha(x,y)))$ is an equivalence.

For proof see [R, §14.4].

October 27: The image factorization

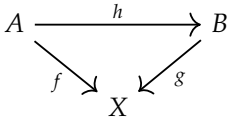
Propositional truncation can be used to define the **image** of a map $f\colon A \to X$, which can be thought of as the smallest subtype of X that contains all of the values of f. More precisely, we'll define a commutative triangle



in which i is an embedding.

The image of a map.

defn. Let $f\colon A \to X$ and $g\colon B \to X$ be maps. A **morphism** from f to g over X is a map $h\colon A \to B$ together with a homotopy $H\colon f \sim g \circ h$ witnessing commutativity of the following triangle



Thus we define the type

$$\hom_X(f,g) :\equiv \Sigma_{h:A\to B} f \sim g \circ h.$$

Composition of morphisms over X is defined by

$$(k,K) \circ (h,H) :\equiv (k \circ h, H \cdot (K \cdot h)).$$

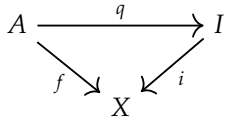
Lemma. *For any $f\colon A \to X$ and any $m : B \to X$ the type,* $\hom_X(f,m)$ *is a proposition.*

Proof. We have an equivalence

$$\hom_X(f,m) :\equiv \Sigma_{h:A\to B} f \sim m \circ h :\equiv \Sigma_{h:A\to B}\Pi_{a:A} f(a) = m(h(a)) \simeq \Pi_{a:A}\Sigma_{b:B} f(a) = m(b) \simeq \Pi_{a:A}\mathrm{fib}_m(f(a)).$$

Since m is an embedding its fibers are propositions. By weak function extensionality, it follows that $\hom_X(f,m)$ is a proposition. □

Proposition. *Consider a commutative triangle*



with $H : f \sim i \circ q$. We say that i satisfies the ***universal property of the image of*** *f if either of the following logically equivalent conditions hold:*

(i) The precomposition function

$$- \circ (q, H) : \hom_X(i, m) \to \hom_X(f, m)$$

is an equivalence for any embedding $m \colon B \hookrightarrow X$.

(ii) For every embedding $m : B \to X$ there is a map

$$\hom_X(f, m) \to \hom_X(i, m).$$

Proof. Since $\hom_X(i, m)$ and $\hom_X(f, m)$ are propositions any map between them is an equivalence if and only if there is exists any map in the other direction. □

Propositional truncation can be used to construct the image of a map:

defn. For any map $f \colon A \to X$ we define the **image** of f to be the type

$$\mathrm{im}(f) \coloneqq \Sigma_{x:X} \| \mathrm{fib}_f(x) \|$$

The **image inclusion** $i_f \colon \mathrm{im}(f) \to X$ is the first projection $\mathtt{pr}_1$. The map $q_f \colon A \to \mathrm{im}(f)$ is given by

$$q_f(a) \coloneqq (f(a), \eta(a, \mathtt{refl}_{f(a)})).$$

The homotopy $I_f : f \sim i_f \circ q_f$ is given by $I_f(a) \coloneqq \mathtt{refl}_{f(a)}$.

We now verify that this construction has the required properties.

Lemma. *The image inclusion $i_f \colon \mathrm{im}(f) \to X$ is an embedding.*

Proof. The fiber of $\mathtt{pr}_1 \colon \Sigma_{x:X} \| \mathrm{fib}_f(x) \| \to X$ is equivalent to $\| \mathrm{fib}_f(x) \|$, which is a proposition. Thus $i_f \coloneqq \mathtt{pr}_1$ is an embedding. □

Theorem [R, 15.1.7] verifies the following:

Theorem. *For any $f \colon A \to X$ the image inclusion $i_f \colon \mathrm{im}(f) \to X$ satisfies the universal property.*

Surjective maps. As mentioned above, our previous notion of surjectivity of a map $f \colon A \to B$ is more properly "split surjectivity" as

$$\left(\Pi_{b:B} \Sigma_{a:A} f(a) = b \right) \simeq \left(\Sigma_{s:A \to B} \Pi_{a:A} f(s(a)) = b \right) =: \left(\Sigma_{s:A \to B} f \circ s \sim \mathtt{id}_B \right).$$

The traditional notion is more closely analogous to:

defn. A map $f \colon A \to B$ is **surjective** if there is a term in the type

$$\text{is-surj}(f) \coloneqq \Pi_{b:B} \| \mathrm{fib}_f(b) \|.$$

Note that having a section is stronger than surjectivity because we have a map $\mathrm{fib}_f(b) \to \| \mathrm{fib}_f(b) \|$ but don't typically have a map in the reverse direction.

Surjective maps also have a universal property recorded by the following theorem. See [R] for a proof:

Proposition. *For a map $f \colon A \to B$, the following are logically equivalent:*

(i) $f \colon A \to B$ is surjective.

(ii) For any family P of propositions over B,

$$- \circ f : (\Pi_{b:B} P(b)) \to \left(\Pi_{a:A} P(f(a)) \right)$$

is an equivalence.

It follows that:

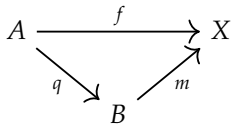
Corollary. *For any $f \colon A \to P$ into a proposition, the following are logically equivalent:*

(i) $f\colon A \to P$ *is a propositional truncation of* A.
(ii) $f\colon A \to P$ *is surjective.*

Proof. The second equivalent condition above is the universal property of propositional truncation, except without the hypothesis that B itself is a proposition. □

We now see that the map q in the image factorization is surjective:

Theorem. *Consider a commutative triangle*

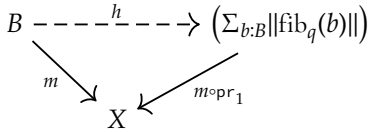


in which m *is an embedding. Then* m *is the image inclusion of* f *if and only if* q *is surjective.*

Proof. First suppose m is the image inclusion of f. In that case the function

$$\left(\Sigma_{b:B}\|\mathrm{fib}_q(b)\|\right) \xrightarrow{\mathtt{pr}_1} B \xrightarrow{m} X$$

is an embedding as a composite of embeddings. By the universal property of m there is a unique map



making the triangle commute. Thus $m \circ (\mathtt{pr}_1 \circ h) \sim m$ but also $m \circ \mathtt{id} \sim m$. By the uniqueness in the universal property of m, it follows that $\mathtt{pr}_1 \circ h \sim \mathtt{id}$. Thus h is a section of the projection map. In particular, this defines a dependent function in

$$\Pi_{b:B}\|\mathrm{fib}_q(b)\|$$

proving that q is surjective.

Conversely, if q is surjective, we can prove that m is the image of f by constructing an equivalence

$$\hom_X(f, m') \to \hom_x(m, m')$$

for any embedding $m'\colon B' \to X$. Using the equivalence noted above, we calculate

$$\hom_X(m, m') \simeq \Pi_{b:B}\mathrm{fib}_{m'}(m(b)) \simeq \Pi_{a:A}\mathrm{fib}_{m'}(m(q(a))) \simeq \Pi_{a:A}\mathrm{fib}_{m'}(f(a)) \simeq \hom_X(f, m'),$$

where the second equivalence uses the hypothesis that q is surjective and the third equivalence follows from the homotopy $f \sim m \circ q$. □

Corollary. *Every map factors uniquely as a surjective map followed by an embedding.*

Proof. If $f\colon A \to X$ admits two such factorizations $I : f \sim i \circ q$ and $I' : f \sim i' \circ q'$ then both embeddings i and i' have the universal property of the image of f. It follows that there is an equivalence $(e, H) : \hom_X(i, i')$ and moreover $(e, H) \circ (q, I) = (q', I')$ □

Cantor's diagonalization argument.

defn. For any type X and universe $\mathcal{U}$ define the $\mathcal{U}$-power set of X to be

$$P_{\mathcal{U}}(X) :\equiv X \to \mathrm{Prop}_{\mathcal{U}}$$

using the universe of propositions in $\mathcal{U}$.

Theorem. *For any type* X *and universe* $\mathcal{U}$ *there is no surjective function* $f\colon X \to P_{\mathcal{U}}(X)$.

Proof. We're asked to prove a negation so we may consider a function $f\colon X \to (X \to \mathrm{Prop}_{\mathcal{U}})$ and suppose that f is surjective. Following Cantor's diagonalization argument, define the subset $P\colon X \to \mathrm{Prop}_{\mathcal{U}}$ by

$$P(x) :\equiv \neg(f(x,x)).$$

Since f is assumed to be surjective and our goal is to reach a contradiction, it suffices to show that

$$\|\Sigma_{x:X} f(x) = P\| \to \varnothing.$$

Since $\varnothing$ is a proposition, by the universal property of propositional truncation it is equivalent to show that

$$\left(\Sigma_{x:X} f(x) = P\right) \to \varnothing.$$

So we consider $x : X$ and an identification $f(x) = P$. From the identification it follows that the propositions $f(x,y)$ and $P(y)$ are logically equivalent for all $y : Y$. In particular $f(x,x)$ is logically equivalent to $P(x)$ but since $P(x) :\equiv \neg f(x,x)$ we have a logical equivalence between $f(x,x)$ and $\neg(f(x,x))$. This gives our desired contradiction. □

November 1: The univalence axiom

The univalence axiom characterizes the identity type of a universe, asserting that identification between types is equivalent to equivalence between types. We shall see that there are several equivalent ways to state the univalence axiom arising from the fundamental theorem of identity types.

Equivalent forms of the univalence axiom. The univalence axiom asserts that the identity type family $A =_{\mathcal{U}} B$ is equivalent to the family of equivalences $A \simeq B$. Immediately from our universal property of identity type families, we have three equivalent forms of this result:

Theorem. *For a universe $\mathcal{U}$ the following are equivalent:*

(i) *$\mathcal{U}$ is **univalent**: for any $A, B : \mathcal{U}$, the map*

$$\texttt{equiv-eq} : (A = B) \to (A \simeq B)$$

defined by $\texttt{equiv-eq}(\texttt{refl}) :\equiv \texttt{id}$ *is an equivalence.*

(ii) *The type*

$$\Sigma_{B:\mathcal{U}} A \simeq B$$

is contractible for each $A : \mathcal{U}$.

(iii) *$\mathcal{U}$ satisfies the principle of **equivalence induction**: for any $A : \mathcal{U}$, and type family $P(X, e)$ indexed by $X : \mathcal{U}$ and $e : A \simeq X$, the evaluation map*

$$(\Pi_{X:\mathcal{U}} \Pi_{e:A\simeq X} P(X,e)) \to P(A, \texttt{id})$$

given by $f \mapsto f(A, \texttt{id})$ has a seciton.

Henceforth, we will assume:

Axiom (univalence)**.** All postulated universes are univalent. We write $\texttt{eq-equiv} : (A \simeq B) \to (A = B)$ for the inverse of $\texttt{equiv-eq}$.

Our first consequence of univalence is **propositional extensionality**, which proves that logically equivalent propositions can be identified. To prove this, we first demonstrate that univalence also characterizes the identity type of any subuniverse.

Proposition. *Let $\mathcal{U}$ be a univalent universe and let P be family of propositions over $\mathcal{U}$. Then the family of maps*

$$\texttt{equiv-eq} : (A = B) \to (\texttt{pr}_1(A) \simeq \texttt{pr}_1(B))$$

indexed by $A, B : \Sigma_{X:\mathcal{U}} P(X)$ given by $\texttt{equiv-eq}(\texttt{refl}) :\equiv \texttt{id}$ *is an equivalence.*

Proof. We've seen that for any family of propositions, the projection map $\texttt{pr}_1$ is an embedding. Therefore we see that the asserted map is the composite of the equivalences:

$$A =_{\Sigma_{X:\mathcal{U}} P(X)} B \xrightarrow{\texttt{ap}_{\texttt{pr}_1}} (\texttt{pr}_1(A) =_{\mathcal{U}} \texttt{pr}_1(B)) \xrightarrow{\texttt{equiv-eq}} (\texttt{pr}_1(A) \simeq \texttt{pr}_1(B)).$$ □

Notation. Given $A : \Sigma_{X:\mathcal{U}} P(X)$ it's common to above notation and write A for $\mathtt{pr}_1(A)$. A similar convention makes sense for terms in any subtype $z : \Sigma_{x:A} P(x)$ where P is a family of propositions over A. Technically speaking the terms in $\Sigma_{x:A} P(x)$ should be regarded as pairs $(x : A, y : P(x))$ but since y is a term in a proposition no data is lost by forgetting exactly what term is declared beyond simply remembering that $P(x)$ is true for the particular term $x : A$.

Important examples of subuniverses include the universe $\mathrm{Prop}_{\mathcal{U}}$ or universes $\mathcal{U}^{\leqslant k}$ of types at any truncation level.

Theorem. *Propositions satisfy* ***proposition extensionality****: for any propositions P and Q the canonical map*

$$\mathtt{iff\text{-}eq} : (P = Q) \to (P \leftrightarrows Q)$$

defined by $\mathtt{iff\text{-}eq}(\mathtt{refl}) = (\mathtt{id}, \mathtt{id})$ *is an equivalence. Moreover, the type of propositions is a set.*

Proof. By function extensionality

$$\text{is-prop}X \coloneqq \Pi_{x,y:X} x = y$$

is a proposition, so the previous result applies to show that the natural map $(P = Q) \to (P \simeq Q)$ is an equivalence. We've previously defined a logical equivalence $(P \simeq Q) \leftrightarrows (P \leftrightarrows Q)$ which we can now upgrade to an equivalence since by function extensionality both types are propositions.[8] Thus

$$(P = Q) \simeq (P \simeq Q) \simeq (P \leftrightarrows Q).$$

□

Corollary. *The type of decidable propositions in any universe is equivalent to* bool*.*

Proof. Since Σ distributes over coproducts we have

$$\Sigma_{P:\mathrm{Prop}_{\mathcal{U}}} \text{is-decidable}(P) \coloneqq \Sigma_{P:\mathrm{Prop}_{\mathcal{U}}} P + \neg P \simeq \left(\Sigma_{P:\mathrm{Prop}_{\mathcal{U}}} P\right) + \left(\Sigma_{Q:\mathrm{Prop}_{\mathcal{U}}} \neg Q\right).$$

We claim that both components are contractible. For the former, we use $(\mathbb{1}, \star)$ as the center of contraction, while for the latter we use $(\varnothing, \mathtt{id})$. For the contractions, it suffices to show that $\mathbb{1} = P$ for any proposition P with $p : P$ and that $\varnothing = Q$ for any proposition Q with $q : \neg Q$. Both identifications follow easily from propositional extensionality. □

Univalence implies function extensionality. We've added two big axioms to Martin-Löf's dependent type theory: univalence and function extensionality. We'll see now that the latter follows from the former using a lemma.

Lemma. *For any equivalence $e : X \simeq Y$ in a univalent universe $\mathcal{U}$ and type A, the map*

$$e \circ - : (A \to X) \to (A \to Y)$$

is an equivalence.

Proof. Since $\mathcal{U}$ is univalent it satisfies equivalence induction. Consider the type family $P(X, e) \coloneqq \text{is-equiv}(e \circ -)$. To prove this, it suffices to show that $\mathtt{id} \circ - : (A \to X) \to (A \to X)$ is an equivalence. But this map is just the identity map so it follows. □

We'll also use a lemma that we've cited already that does not require univalence, which we now pause to prove:

Lemma. *For any type family B over A the fiber of the map $\mathtt{pr}_1 : \Sigma_{x:A} B(x) \to A$ over $a : A$ is equivalent to $B(a)$.*

Proof. The comparison maps are defined by

$$B(a) \xrightarrow{\ s\ } \Sigma_{z:\Sigma_{x:A} B(x)} \mathtt{pr}_1(z) = a \xrightarrow{\ r\ } B(a)$$

$$b \longmapsto ((a, b), \mathtt{refl}_a)$$

$$((x, y), p) \longmapsto \mathtt{tr}_B(p, y)$$

Note $r(s(b)) \doteq \mathtt{tr}_B(\mathtt{refl}_a, b)) \doteq b$ so $\mathtt{refl}$ provides a homotopy $r \circ s \sim \mathtt{id}_{B(a)}$.

We calculate that

$$s(r((x, y), p)) \doteq s(\mathtt{tr}_B(p, y)) \doteq ((a, \mathtt{tr}_B(p, y)), \mathtt{refl}_a)$$

so for the other homotopy we require a path from $((x, y), p)$ to $((a, \mathtt{tr}_B(p, y)), \mathtt{refl}_a)$ for all a, x, y, and p. By path induction on p it suffices to define this in the case where a is x and p is $\mathtt{refl}_x$, in which case the reflexivity path will do. □

[8] In general, $A \simeq B$ is a k-type if both A and B are k-types.

Theorem. *For any universe $\mathcal{U}$, univalence implies function extensionality.*

Proof. It suffices to show that univalence implies weak function extensionality, so consider a family $B\colon A \to \mathcal{U}$ of contractible types. It follows that $\mathtt{pr}_1\colon \Sigma_{x:A}B(x) \to A$ is an equivalence. We'd like to show that $\Pi_{x:A}B(x)$ is contractible.

By the lemma we know that

$$\mathtt{pr}_1 \circ -\colon (A \to \Sigma_{x:A}B(x)) \to (A \to A)$$

is an equivalence, so in particular the fiber at $\mathtt{id}_A$ must be contractible. We'll show that $\Pi_{x:A}B(x)$ is a retract of this fiber

$$\Sigma_{f:A\to\Sigma_{x:A}B(x)}\mathtt{pr}_1 \circ f = \mathtt{id}_A.$$

Note that terms in $\Pi_{x:A}B(x)$ are sections of the type family, while terms in $\Sigma_{f:A\to\Sigma_{x:A}B(x)}\mathtt{pr}_1 \circ f = \mathtt{id}_A$ are sections up to homotopy.

The section is defined by

$$i(f) :\equiv (\lambda x, (x, f(x)), \mathtt{refl}_{\mathtt{id}}).$$

For the retraction consider $h : A \to \Sigma_{x:A}B(x)$ with $p : \mathtt{pr}_1 \circ h = \mathtt{id}$. We then have $\mathtt{htpy\text{-}eq}(p) : \mathtt{pr}_1 \circ h \simeq \mathtt{id}$ and $\mathtt{pr}_2(h(x)) : B(\mathtt{pr}_1(h(x)))$. We define

$$r((h,p),x) :\equiv \mathtt{tr}_B(\mathtt{htpy\text{-}eq}(p,x), \mathtt{pr}_2(h(x))).$$

It remains to construct a homotopy $H : r \circ i \simeq \mathtt{id}$ but in this case we may compute

$$r(i(f)) \doteq r(\lambda x, (x, f(x)), \mathtt{refl}_{\mathtt{id}}) \doteq \mathtt{tr}_B(\mathtt{htpy\text{-}eq}(\mathtt{refl}, x), \mathtt{pr}_2(x, f(x))) \doteq \mathtt{tr}_B(\mathtt{refl}, f(x)) \doteq f(x).$$

Thus we may define $H(f) \doteq \mathtt{refl}$. □

Maps and families of types. An important consequence of univalence is the following:

Theorem. *For any type A and any univalent universe $\mathcal{U}$ containing A the map*

$$(\Sigma_{X:\mathcal{U}}X \to A) \to (A \to \mathcal{U})$$

given by $(X, f) \mapsto \mathrm{fib}_f$ is an equivalence.

Proof. The map in the converse direction is

$$B \mapsto \left(\Sigma_{x:A}B(x), \mathtt{pr}_1\right).$$

To see that this map is a section, we must prove that $\mathrm{fib}_{\mathtt{pr}_1} = B$ for any $B\colon A \to \mathcal{U}$. By function extensionality and the univalence axiom, this is equivalent to showing that

$$\Pi_{x:A}\mathrm{fib}_{\mathtt{pr}_1}(x) \simeq B(x),$$

which is the result of the lemma in the previous section that does not require univalence.

We must also verify that

$$(X, f) = \left(\Sigma_{a:A}\mathrm{fib}_f(a), \mathtt{pr}_1\right).$$

For this, first observe that the identity type $(X, f) = (Y, g)$ in the type $\Sigma_{X:\mathcal{U}}X \to A$ is equivalent to the type of pairs (e, f) given by $e : X \simeq Y$ and a homotopy $f \sim g \circ e$. This follows from the universal property of identity types using the fact that the type

$$\Sigma_{Y:\mathcal{U}}\Sigma_{g:\, Y\to A}\Sigma_{e:X\simeq Y} f \sim g \circ e$$

is contractible by a result we skipped called the "structure identity principle." Using this result it suffices to construct an equivalence $e\colon X \simeq \Sigma_{a:A}\mathrm{fib}_f(a)$ and homotopy $f \sim \mathtt{pr}_1 \circ e$. Intuitively, this follows from the observation that

$$\Sigma_{a:A}\mathrm{fib}_f(a) :\equiv \Sigma_{a:A}\Sigma_{x:X} f(x) = a \simeq \Sigma_{x:X}\Sigma_{a:A} f(x) = a \simeq \Sigma_{x:X}\mathbb{1} \simeq X.$$

The full details are left to the exercises. □

We can extend this result as follows. For any family of types P indexed by $\mathcal{U}$ write

$$\mathcal{U}_P :\equiv \Sigma_{X:\mathcal{U}}P(X).$$

If P is a family of propositions this is a subuniverse of $\mathcal{U}$ but we can consider this construction more generally.

Theorem. *For any type A, any univalent universe $\mathcal{U}$ containing A, and any family of types P indexed by $\mathcal{U}$, the map*

$$\left(\Sigma_{X:\mathcal{U}}\Sigma_{f:X\to A}\Pi_{a:A}P(\mathrm{fib}_f(a))\right) \to (A \to \mathcal{U}_P)$$

given by $(X, f, p) \mapsto \lambda a(\mathrm{fib}_f(a)p(a))$ is an equivalence.

Proof. The map is homotopic to the composite of equivalences

$$\Sigma_{X:\mathcal{U}}\Sigma_{f:X\to A}\Pi_{a:A}P(\mathrm{fib}_f(a)) \simeq \Sigma_{(X,f):\Sigma_{X:\mathcal{U}}X\to A}\Pi_{a:A}P(\mathrm{fib}_f(a)) \simeq \Sigma_{B:A\to\mathcal{U}}\Pi_{a:A}P(B(a)) \simeq A \to \Sigma_{X:\mathcal{U}}P(X). \qquad \square$$

As a special case, we have an equivalence between families of propositions and embeddings.

Corollary. *For any type A and any univalent universe $\mathcal{U}$ containing A the map*

$$(\Sigma_{X:\mathcal{U}}X \hookrightarrow A) \to (A \to \mathrm{Prop}_{\mathcal{U}})$$

given by $(X, f) \mapsto \mathrm{fib}_f$ is an equivalence.

November 3: Classical mathematics with the univalence axiom

The univalence axiom asserts that identifications $A =_{\mathcal{U}} B$ in the universe of types are equivalent to equivalences $A \simeq B$ between types. This interpretation is incompatible with an interpretation of type theory in which types are interpreted by sets and identity types are interpreted by the equality relation on those sets. In particular, if $\mathcal{U}$ is interpreted by a set then for any $A : \mathcal{U}$ the set intepreting $A =_{\mathcal{U}} A$ has exactly one element (the reflexivity term). But

$$A \simeq A \coloneqq \Sigma_{f:A\to A}\Pi_{a:A}\text{is-contr}\left(\Sigma_{x:A}f(x) = a\right)$$

is the set of bijective functions $f\colon A \to A$. In particular, if the set interpreting A has more than one element $A = A$ and $A \simeq A$ cannot be equivalent.

That said, the univalent foundation system *is* consistent with classical mathematics: provided one allows the existence of types, such as $\mathcal{U}$, which are not mere sets. We'll now explore this compatibility in more detail.

Classical mathematics with the univalence axiom. The univalence axiom *is* consistent with the axiom of choice provided that care is taken to make choice about sets.

To reason about this concretely, consider the type

$$\mathbb{F}_2 \coloneqq \Sigma_{X:\mathcal{U}}\|\mathrm{Fin}_2 \simeq X\|$$

of **2-element types**. Here $\mathrm{Fin}_2 \coloneqq \mathbb{1} + \mathbb{1}$ is the standard 2-element type. We write $i(\star)$ for $\mathtt{inl}(\star) : \mathrm{Fin}_2$ and $\star$ for $\mathtt{inr}(\star) : \mathrm{Fin}_2$.

Proposition. *The canonical family of maps*

$$\mathsf{ev}_\star : (\mathrm{Fin}_2 \simeq X) \to X$$

is a family of equivalences. Consequently, the type

$$\Sigma_{X:\mathbb{F}_2}X$$

of pointed 2-element types is contractible.

Proof. By the univalence axiom, the type

$$\Sigma_{X:\mathbb{F}_2}\mathrm{Fin}_2 \simeq X$$

is contractible. To see that $\Sigma_{X:\mathbb{F}_2}X$ is contractible, it suffices to show the maps $\mathsf{ev}_\star : (\mathrm{Fin}_2 \simeq X) \to X$ define a family of equivalences, since then they induce an equivalence between the Σ-types.

Since $X : \mathbb{F}_2$ we have an assumption of the form $\|\mathrm{Fin}_2 \simeq X\|$. Since our goal is to prove a proposition, namely

$$\text{is-equiv}(\mathsf{ev}_\star : (\mathrm{Fin}_2 \simeq X) \to X)$$

we may assume we have a term $\alpha : \mathrm{Fin}_2 \simeq X$. Now we can proceed by equivalence induction on α, meaning it suffices to assume X is Fin_2 and α is the identity equivalence, in which case our goal is to show that

$$\mathsf{ev}_\star : (\mathrm{Fin}_2 \simeq \mathrm{Fin}_2) \to \mathrm{Fin}_2$$

is an equivalence. In this case we can define an inverse equivalence $g : \mathrm{Fin}_2 \to (\mathrm{Fin}_2 \simeq \mathrm{Fin}_2)$ by $g(\star) \coloneqq \mathtt{id}$ and $g(i(\star)) \coloneqq \mathsf{succ}_2$ (the non-identity equivalence). It is straightforward to verify that these maps are inverse equivalences. $\square$

Consequently:

Corollary. *There is no dependent function*

$$\Pi_{X:\mathbb{F}_2} X$$

that chooses an element of every 2-element set.

Proof. By the previous proposition we have an equivalence

$$\left(\Pi_{X:\mathbb{F}_2} \mathrm{Fin}_2 = X\right) \simeq \left(\Pi_{X:\mathbb{F}_2} X\right).$$

Note the left-hand side is the type of contracting homotopies for $\mathbb{F}_2$ using $\mathrm{Fin}_2 : \mathbb{F}_2$ as the center of contraction. So it suffices to show that $\mathbb{F}_2$ is not contractible (which can only be the case if this type has no terms). In the previous proposition, we proved that $(\mathrm{Fin}_2 \simeq \mathrm{Fin}_2) \simeq \mathrm{Fin}_2$. Thus, by univalence, the identity type $\mathrm{Fin}_2 = \mathrm{Fin}_2$ of the term $\mathrm{Fin}_2 : \mathbb{F}_2$ is not contractible. Thus $\mathbb{F}_2$ must not be contractible. □

As a corollary, we can conclude more generally that there is not way to construct a term of an arbitrary inhabited type.

Theorem. *If $\mathcal{U}$ is a univalent universe, then there is no function*

$$\Pi_{A:\mathcal{U}} \|A\| \to A.$$

Proof. If we had such a term f we could restrict to the type of 2-element types to obtain a function

$$\Pi_{A:\mathbb{F}_2} \|A\| \to A.$$

Since every 2-element type A is inhabited there is a term of type $\|A\|$. To see this, assume $\|\mathrm{Fin}_2 \simeq A\|$. To produce a term of type $\|A\|$ it suffices to assume that $e : \mathrm{Fin}_2 \simeq A$. Now $e(\star) : A$ and $\eta(e(\star)) : \|A\|$.

Thus, by evaluating at this term $\eta(e(\star)) : \|A\|$, we obtain a function

$$\Pi_{A:\mathbb{F}_2} A$$

which we've just shown is impossible. □

The statement

$$\Pi_{A:\mathcal{U}} \|A\| \to A$$

is called the **principle of global choice**. We've just shown that this principle is incompatible with the univalence axiom: in other words, we cannot obtain a term of type A from the assumption $\|A\|$ that A is inhabited. What goes wrong? If we did have such a function $\texttt{global-choice} : \Pi_{A:\mathcal{U}} \|A\| \to A$ it would necessarily be invariant under identifications in $\mathcal{U}$: $\mathsf{apd}_{\texttt{global-choice}}$ is a dependent function that takes a path $p : A = B$ to an identification between the transport of the function $\texttt{global-choice} : \|A\| \to A$ along p and the function $\texttt{global-choice} : \|B\| \to B$.

By univalence, anything that respects identification between types (which is to say, every construction in homotopy type theory) must also be invariant under equivalences between types. But no choices of an element of A can be invariant under all automorphisms of A (unless A is contractible). For instance, when $A :\equiv \mathrm{Fin}_2$, there is no fixed point of the equivalence $\mathsf{succ}_2 : \mathrm{Fin}_2 \simeq \mathrm{Fin}_2$.

In a sense the principle of global choice is taking the wrong point of view on the traditional axiom of choice, which is really an axiom about *sets*. The inconsistency of the global choice function is tied to the inconsistency in defining any sections of the type family $\lambda X.X : \mathbb{F}_2 \to \mathcal{U}$ of inhabited types. Here the fibers of this type family, the 2-element types X, are sets, but the base type $\mathbb{F}_2$ is not a set but rather a 1-type. The axiom of choice is more properly an assertion that only applies to type families $B\colon A \to \mathcal{U}$ in which both the base type A and the fibers $B(x)$ are sets.

defn. The **axiom of choice** asserts that for any family B of inhabited sets indexed by a set A the type of sections of B is also inhabited:

$$\mathsf{AC}_{\mathcal{U}} :\equiv \Pi_{A:\mathrm{Set}_{\mathcal{U}}} \Pi_{B:A\to\mathrm{Set}_{\mathcal{U}}} \left(\Pi_{x:A} \|B(x)\|\right) \to \|\Pi_{x:A} B(x)\|.$$

The consistency of this axiom with univalence is established by Voevodsky's model of homotopy type theory in the category of simplicial sets (when the sets are taken to be classical sets). This version of the axiom of choice asserts essentially that every surjective function between sets has a section.

Similar care has to be taken with the type theoretic formulation of the law of the excluded middle. Recall for a type X,

$$\text{is-decidable}(X) :\equiv X + \neg X.$$

With univalence, it is inconsistent to assume that every type is decidable:

Theorem. *There is no dependent function*

$$\Pi_{X:\mathcal{U}}\text{is-decidable}(X).$$

Proof. If such a function existed, it would restrict to a dependent function

$$d : \Pi_{X:\mathbb{F}_2}\text{is-decidable}(X).$$

Since each 2-element type is inhabited $\neg(X) \to \varnothing$, so is-decidable$(X) \to X$. Thus from d we would obtain a dependent function in $\Pi_{X:\mathbb{F}_2}X$, which we've seen does not exist. □

The remedy is similar to the discussion above. We've argued that the axiom of choice is really an axiom about *sets*. Similarly, the law of excluded middle is really an axiom about *propositions*, and it is consistent with univalence to assume that every proposition is decidable.

defn. The **law of excluded middle** asserts that every proposition is decidable

$$\mathtt{LEM}_{\mathcal{U}} : \Pi_{P:\text{Prop}_{\mathcal{U}}}\text{is-decidable}(P).$$

Again, the consistency of this axiom with univalence is established by Voevodsky's model of homotopy type theory in the category of simplicial sets, in which case the propositions are the subobjects of the one-point simplicial set.

We will not assume either $\mathtt{AC}_{\mathcal{U}}$ or $\mathtt{LEM}_{\mathcal{U}}$ going forward but nevertheless it is reassuring to know that these assumptions are consistent.

November 8: Groups in univalent mathematics

In order to demonstrate a typical way to use the univalence axiom, we tour the theory of groups in univalent mathematics. By something called the *structure identity principle*, we will see that univalence implies that isomorphic groups can be *identified*.

The type of all groups. We introduce the group axioms in stages. Recall a type A is a set if

$$\Pi_{x,y:A}\Pi_{p,q:x=y}\text{is-contr}p = q.$$

defn. A **semi-group** consists of a set G equipped with a term of type has-associative-mul(G), which is the type of pairs $(\mu_G, \mathtt{assoc}_G)$ comprised of

$$\mu_G : G \to G \to G$$

anda homotopy

$$\mathtt{assoc}_G : \Pi_{x,y,z:G}\mu_G(\mu_G(x,y),z) = \mu_G(x,\mu_G(y,z)).$$

We write

$$\text{Semi-Group} \coloneqq \Sigma_{G:\text{Set}_{\mathcal{U}}}\Sigma_{\mu_G:G\to G\to G}\Pi_{x,y,z:G}\mu_G(\mu_G(x,y),z) = \mu_G(x,\mu_G(y,z)).$$

defn. A semi-group G is a **unital semi-group** or a **monoid** if it comes equipped with a **unit** $e_G : G$ satisfying left and right unit laws:

$$\mathtt{left\text{-}unit}_G : \Pi_{y:G}\mu_G(e_G,y) = y \qquad \mathtt{right\text{-}unit}_G : \Pi_{x:G}\mu_G(x,e_G) = x.$$

We write

$$\text{is-unital}(G) \coloneqq \Sigma_{e_G:G}\left(\Pi_{y:G}\mu_G(e_G,y) = y\right) \times \left(\Pi_{x:G}\mu_G(x,e_G) = x\right).$$

In classical mathematics, the unit of a semi-group is unique once it exists. In univalent mathematics, this is expressed by the following result:

Lemma. *For a semi-group G, the type* is-unital(G) *is a proposition.*

In other words, being unital is a *property* of semi-groups rather than *structure* on semi-groups.

Proof. Since a semi-group G is a set, the types of the left and right unit laws are propositions. Therefore it suffices to show that any two terms $e, e' : G$ satisfying the left and right unit laws can be identified. This is easy using the right-unit law for e' and the left-unit law for e:

$$\mathtt{inv}(\mathtt{right\text{-}unit}_G(e)) \cdot \mathtt{left\text{-}unit}_G(e') : e = \mu_G(e,e') = e'.$$ □

defn. Let G be a unital semi-group. We say G **has inverses** if it comes with an operation $x \mapsto x^{-1} : G \to G$ satisfying the left and right inverse laws:

$$\texttt{left-inv}_G : \prod_{x:G} \mu_G(x^{-1}, x) = e_G \qquad \texttt{right-inv}_G : \prod_{x:G} \mu_G(x, x^{-1}) = e_G.$$

We write

$$\text{is-group}(G) :\equiv \Sigma_{e_G:\text{is-unital}(G)} \Sigma_{x \mapsto x^{-1}:G \to G} \left(\prod_{x:G} \mu_G(x^{-1}, x) = e_G\right) \times \left(\prod_{x:G} \mu_G(x, x^{-1}) = e_G\right).$$

Lemma. *For any semi-group G, the type* is-group(G) *is a proposition.*

Proof. We have seen already that is-unital(G) is a proposition so it suffices to show that

$$\Sigma_{x \mapsto x^{-1}:G \to G} \left(\prod_{x:G} \mu_G(x^{-1}, x) = e\right) \times \left(\prod_{x:G} \mu_G(x, x^{-1}) = e\right)$$

is a proposition for any $e :$ is-unital(G). Since semi-groups are sets, the types of the inverse laws are propositions, so it suffices to show that any two operations satisfying the inverse laws are homotopic. To that end consider $x \mapsto x^{-1}$ and $x \mapsto \bar{x}^{-1}$. Then we have

$$x^{-1} = \mu_G(e, x^{-1}) = \mu_G(\mu_G(\bar{x}^{-1}, x), x^{-1}) = \mu_G(\bar{x}^{-1}, \mu_G(x, x^{-1}) = \mu_G(\bar{x}^{-1}, e) = \bar{x}^{-1}. \qquad \square$$

defn. A **group** is a unital semi-group with inverses. We write

$$\text{Group} :\equiv \Sigma_{G:\text{Set}_{\mathcal{U}}} \Sigma_{\mu_G:G \to G \to G} \Sigma_{\texttt{assoc}_G:\prod_{x,y,z:G} \mu_G(\mu_G(x,y),z) = \mu_G(x,\mu_G(y,z))} \text{is-group}(G).$$

for the type of all groups in $\mathcal{U}$.

Recall a groupoid is a "group with many objects." This suggests that one-object groupoids give examples of groups:

ex. An important example of groups consist of **loop spaces** $x =_X x$ for any 1-type X and any $x : X$. We write $\Omega(X, x)$ for the loop space. Since X is a -1 type $\Omega(X, x) :\equiv x =_X x$ is a set. Then we have

$$\texttt{refl}_x : \Omega(X, x) \qquad \texttt{inv} : \Omega(X, x) \to \Omega(X, x) \qquad \texttt{concat} : \Omega(X, x) \to \Omega(X, x) \to \Omega(X, x)$$

satisfying all of the required laws, as special cases of the groupoid laws for identity types.

ex. The type of integers $\mathbb{Z} :\equiv \mathbb{N} + (\mathbb{1} + \mathbb{N})$ can be given the structure of a group with addition as the group operation. The proof that $\mathbb{Z}$ is a set is similar to the proof that $\mathbb{N}$ is a set. The group laws were shown in the exercises.

ex. A final example is given by an **automorphism group** of a set. Given a set X define

$$\text{Aut}(X) :\equiv (X \simeq X).$$

The group operation is composition of equivalences and the unit is the identity function. While composition os functions is strictly associative and strictly unital, composition of equivalences only satisfies the group laws up to identification because this also requires a composite of the equivalence data.

As an important special case we define the **symmetric groups**

$$S_n :\equiv \text{Aut}(\text{Fin}_n).$$

Group homomorphisms.

defn. Let G and H be semi-groups. A **homomorphism** of semi-groups is a pair (f, μ_f) comprised of a function $f\colon G \to H$ between their underlying types and a term

$$\mu_f : \prod_{x,y:G} f(\mu_G(x, y)) = \mu_H(f(x), f(y))$$

witnessing that f preserves the binary operations. We write

$$\hom(G, H) :\equiv \Sigma_{f:G \to H} \prod_{x,y:G} f(\mu_G(x, y)) = \mu_H(f(x), f(y))$$

for the type of all semi-group homomorphisms.

Since it is a property for a function to preserve the multiplication of a semi-group, it follows that the identity type between two terms $f, f' : \hom(G, H)$ is equivalent to the type of homotopies between their underlying functions. In particular $\hom(G, H)$ is a set.

ex. The **identity homomorphism** on a semi-group G is given by $\texttt{id} : G \to G$ and

$$\lambda x.\lambda y.\texttt{refl}_{\mu_G(x,y)} : \prod_{x,y:G} \mu_G(x, y) = \mu_G(x, y).$$

Remark. If $f\colon G \to H$ and $g\colon H \to K$ are semi-group homomorphisms, their composite $g \circ f\colon G \to K$ is also a semi-group homomorphism using the composite identification

$$g(f(\mu_G(x,y)) = g(\mu_H(f(x), f(y)) = \mu_K(g(f(x)), g(f(y))).$$

The following coherence laws can be verified

$$\mathtt{id} \circ f = f \qquad g \circ \mathtt{id} = g \qquad (h \circ g) \circ f = h \circ (g \circ f),$$

using the fact that identity types of semi-group homomorphisms are equivalent to the identity types between the underlying functions.

defn. Let G and H be groups. A **homomorphism** of groups from G to H is defined to be a semi-group homomorphism between their underlying semi-groups. We will write

$$\hom(G,H)$$

for the type of all group homomorphisms.

Why this definition? On the one hand, you might remember that the classical definition of group homomorphism only requires compatibility with multiplication: the preservation of units and inverses comes for free. If you recall the proof of this, you'll be lead to define, for instance, a composite identification:

$$e_H = \mu_H(f(e_G), f(e_G)^{-1}) = \mu_H(f(\mu_G(e_G, e_G)), f(e_G)^{-1}) = \mu_H(\mu_H(f(e_G), f(e_G)), f(e_G)^{-1})$$
$$= \mu_H(f(e_G), \mu_H(f(e_G), f(e_G)^{-1}) = \mu_H(f(e_G), e_H) = f(e_G).$$

Since the identity types in a set are propositions, there is no data defined by an explicit identification beyond its mere existence.

Isomorphic groups are equal.

defn. A homomorphism of semi-groups $h : \hom(G,H)$ is said to be an **isomorphism** if it comes with a term of type is-iso(h) consisting of triples (h^{-1}, p, q) given by $h^{-1} : \hom(H,G)$ and

$$p : h^{-1} \circ h = \mathtt{id}_G \qquad \text{and} \qquad q : h \circ h^{-1} = \mathtt{id}_H$$

witnessing that h^{-1} satisfies the inverse laws.

Write

$$G \cong H \coloneqq \Sigma_{h:\hom(G,H)} \Sigma_{k:\hom(H,G)} (k \circ h = \mathtt{id}_G) \times (h \circ k = \mathtt{id}_H).$$

If h is an isomorphism its inverse is unique. In other words, being an isomorphism is a property:

Lemma. *For any semi-group homomorphism $h : \hom(G,H)$, the type* is-iso(h) *is a proposition. Thus, for any two semi-groups $G \cong H$ is a set.*

Proof. Suppose $k, k' : \hom(H,G)$ are two inverses of h. Since the type of semi-group homomorphisms is a set, we have that $h \circ k = \mathtt{id}$ and $k \circ h = \mathtt{id}$ are propositions, and similarly for k', so it suffices to check that $k = k'$. We've observed that this identity type is equivalent to the type of homotopies $k \sim k'$ and we can construct one by the usual argument: applying k to the inverse of the coherence $h \circ k' = \mathtt{id}$ at $y : H$ and concatenating with the coherence $k \circ h = \mathtt{id}$ at $k'(y) : G$ yields:

$$k(y) = k(h(k'(y))) = k'(y).$$

□

Lemma. *A semi-group homomorphism $h : \hom(G,H)$ is an isomorphism if and only if its underlying map is an equivalence. Consequently there is an equivalence*

$$(G \cong H) \simeq \Sigma_{e:G \simeq H} \Pi_{x,y:G} e(\mu_G(x,y)) = \mu_H(e(x), e(y)).$$

Proof. If $h : \hom(G,H)$ is an isomorphism, then the inverse semi-group homomorphism also provides an inverse to the underlying map of h. Thus h must be an equivalence.

For the converse, suppose $h : \hom(G,H)$ is a semi-group homomorphism whose underlying map $h : G \to H$ is an equivalence, with equivalence inverse $k : H \to G$. Then for all $x, y : G$

$$h(\mu_G(k(x), k(y))) = \mu_H(h(k(x)), h(k(y))) = \mu_H(x,y) = hk(\mu_H(x,y)).$$

Since h is an equivalence it follows that $\mu_G(k(x), k(y)) = k(\mu_H(x,y))$ so k is a semi-group homomorphism. The homotopies of the equivalence provide the homotopies of the group isomorphism. □

defn. Let G and H be semi-groups. We define the map

$$\texttt{iso-eq} : (G = H) \to (G \cong H)$$

by path induction taking $\texttt{refl}_G$ to $\texttt{id}_G$.

Theorem. *The map*

$$\texttt{iso-eq} : (G = H) \to (G \cong H)$$

is an equivalence for any two semi-groups G and H.

Proof. By the fundamental theory of identity types, it suffices to show that

$$\Sigma_{G' : \text{Semi-Group}} G \cong G'$$

is contractible. This is equivalent to the type

$$\Sigma_{G' : \text{Semi-Group}} \Sigma_{e : G \simeq G'} \Pi_{x,y:G} e(\mu_G(x, y)) = \mu_{G'}(e(x), e(y)).$$

Since

$$\text{Semi-Group} \coloneqq \Sigma_{G : \text{Set}_{\mathcal{U}}} \text{has-associative-mul}(G),$$

we have

$$\left(\Sigma_{G' : \text{Semi-Group}} \Sigma_{e : G \simeq G'} \Pi_{x,y:G} e(\mu_G(x, y)) = \mu_{G'}(e(x), e(y))\right)$$
$$\simeq \left(\Sigma_{G' : \text{Set}_{\mathcal{U}}} \Sigma_{e : G \simeq G'} \Sigma_{\mu_{G'} : \text{has-associative-mul}(G')} \Pi_{x,y:G} e(\mu_G(x, y)) = \mu_{G'}(e(x), e(y))\right).$$

Thus, it suffices to show that the type $\Sigma_{G' : \text{Set}_{\mathcal{U}}} G \simeq G'$ is contractible, which follows by the univalence axiom, and uses $(G, \texttt{id}_G)$ as the center of contraction, and then show that the type

$$\Sigma_{\mu' : \text{has-associative-mul}(G)} \Pi_{x,y:G} \mu_G(x, y) = \mu_{G'}(x, y)$$

is contractible. This holds by function extensionality, with $(\mu_G, \texttt{refl})$ as the center of contraction. □

Corollary. *The type* Semi-Group *is a 1-type.*

Proof. We've just shown that $(G = H) \simeq (G \cong H)$ for semi-groups G and H. It's straightforward to see, using the fact that G and H are sets, that $G \cong H$ is a set. □

The results for groups follow similarly though the types $G =_{\text{Semi-Group}} H$ and $G =_{\text{Group}} H$ are not equal.

defn. Let G and H be groups. We define the map

$$\texttt{iso-eq} : (G = H) \to (G \cong H)$$

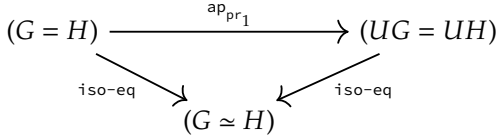
by path induction taking $\texttt{refl}_G$ to $\texttt{id}_G$.

Theorem. *The map*

$$\texttt{iso-eq} : (G = H) \to (G \cong H)$$

is an equivalence for any two groups G and H.

Proof. Let G and H be group and write UG and UH for their underlying semi-groups. Then we have a commutative triangle



Since being a group is a property of semi-groups, the projection map $\texttt{pr}_1 : \text{Group} \to \text{Semi-Group}$ is an embedding. Thus the top map is an equivalence, as is the right map by the previous theorem. The result follows. □

Corollary. *The type of groups is a 1-type.*

These results follow a general pattern. The **Structure Identity Principle** states that any property of set-level structures (e.g., posets, groups, rings, fields) definable in Univalent Foundations is invariant under isomorphism. More specifically, identifications of structures coincide with isomorphisms. There is some subtlety in the correct formulation of this result which we illustrate next time by considering categories in univalent mathematics.

Traditional group theory fits very comfortably into the set-theoretical foundations of mathematics but category theory is less comfortable there. A key problem is that most of category theory is invariant under weaker notions of "sameness" than equality, such as isomorphism in ac category of equivalence of categories, in a way that set theory fails to capture. But this is the same sort of problem that the univalence axiom solves for types. Thus, in univalent foundations, it makes sense to consider a notion of "category" in which equality of objects is identified with isomorphism in a similar way.

We'll actually introduce two definitions, which following [UF] are called *precategories* and *categories*, the latter of which might be thought of as the "univalent" categories.

An illustration of the difference between precategories and categories comes from the behavior of equivalences between such notions. In classical mathematics the statement that "every fully faithful and essentially surjective functor is an equivalence of categories" is equivalent to the axiom of choice. For precategories, there is no consistent axiom of choice, which can make it true. For categories, it is provable without any axiom of choice. We won't have time to prove this here but see [UF, §9.4], which also explores a third notion of *strict categories*.

Precategories and categories.

defn. A **precategory** A consists of:

- A type A_0 of objects. We write $a : A$ to mean $a : A_0$.
- For each $a, b : A$, a set $\hom_A(a, b)$ of **arrows** or **morphisms**
- For each $a : A$ a morphism $1_a : \hom_A(a, a)$.
- For each $a, b, c : A$ a function
$$\hom_A(b, c) \to \hom_A(a, b) \to \hom_A(a, c)$$
denoted by $g \mapsto f \mapsto g \circ f$.
- For each $a, b : A$ and $f : \hom_A(a, b)$ proofs that $f = 1_b \circ f$ and $f = f \circ 1_a$.
- For each $a, b, c, d : A$ and $f : \hom_A(a, b)$, $g : \hom_A(b, c)$ and $h : \hom_A(c, d)$ a proof that $h \circ (g \circ f) = (h \circ g) \circ f$.

The problem with precategories is there are a priori two distinct notions of sameness for terms $a, b : A$, one given by the identity type and the other given by the categorical notion of isomorphism:

defn. A term $f : \hom_A(a, b)$ is an **isomorphism** if there is a term of type
$$\text{is-iso}(f) :\equiv \Sigma_{g:\hom_A(b,a)}(g \circ f = 1_a) \times (f \circ g = 1_b).$$

For instance, 1_a is an isomorphism with 1_a and the unit coherences defining the required term of is-iso(f).

Lemma. *For any $f : \hom_A(a, b)$, the type* is-iso(f) *is a proposition. Thus the type*
$$a \cong b :\equiv \Sigma_{f:\hom_A(a,b)}\text{is-iso}(f)$$
is a set.

Proof. Given $(g, \eta, \epsilon), (g', \eta', \epsilon') :$ is-iso(f) we must construct an identification between them. Since hom-sets are sets, their identity types are propositions and it suffices to construct an identification $g = g'$. For this we have
$$g' = 1_a \circ g' = (g \circ f) \circ g' = g \circ (f \circ g') = g.$$
Thus is-iso $: \hom_A(a, b) \to \text{Prop}_{\mathcal{U}}$ is a family of propositions, so $\mathtt{pr}_1 : a \cong b \to \hom_A(a, b)$ is an embedding. Since $\hom_A(a, b)$ is a set, $a \cong b$ must be as well. □

When f is an isomorphism, is-iso(f) is contractible. In particular, we may write $f^{-1} : \hom_A(b, a)$ for its uniquely determined inverse morphism.

defn. If A is a precategory and $a, b : A$ there is a map
$$\mathtt{iso\text{-}id} : (a = b) \to (a \cong b)$$
defined by path induction by $\mathtt{refl}_a \mapsto \mathtt{id}_a$.

A category is a precategory with a unified notion of equivalence.

defn. A **category** is a precategory so that for all a, b the map $\mathtt{iso\text{-}id} : (a = b) \to (a \cong b)$ is an equivalence.

Corollary. *In any category, the type of objects is a 1-type.*

Proof. Since $a \cong b$ is a set, $a = b$ is a set, making A into a 1-type. □

ex. The type $\mathrm{Set}_{\mathcal{U}}$ is a category with $\hom(A, B) :\equiv A \to B$ in any universe large enough to contain $\mathrm{Set}_{\mathcal{U}}$. Here we have $(A \cong B) \simeq (A \simeq B) \simeq (A = B)$ by the univalence axiom.

ex. The types SemiGroup and Group are also categories in sufficiently large universes with the hom-sets given by the types of (semi-)group homomorphisms.

ex. If X is any 1-type, there is a category with X as its type of objects and $\hom(x, y) :\equiv (x = y)$. In this category every morphism is an isomorphism, so X might be called a **groupoid**.

ex. A precategory A in which each set $\hom_A(a, b)$ is a proposition is called a **preorder**. Equivalently, this data is given by a type A_0 equipped with a proposition-valued relation $\leqslant$ that is reflexive ($a \leqslant a$) and transitive ($b \leqslant c) \to (a \leqslant b) \to (a \leqslant c$).

A unique morphism $f : \hom_A(a, b) \simeq a \leqslant b$ is an isomorphism just when $b \leqslant a$. Thus $a \cong b$ is the proposition $(a \leqslant b) \times (b \leqslant a)$. Thus, we see that a preorder is category just when $a = b$ is a proposition and the relation $\leqslant$ is anti-symmetric: $(a \leqslant b) \times (b \leqslant a) \to (a = b)$.

ex. If A is a category then A_0 is a set if and only if $a \cong b$ is a proposition. In particular, every automorphism in $a \cong a$ must be an identity. In fact, it's typically understood to mean that every *isomorphism* in $a \cong b$ is an identity, meaning it corresponds to $\mathtt{id}_a : a \cong a$ under the transport equivalence $(a \cong a) \simeq (a \cong b)$ induced by the corresponding path in $a = b$. This is a strong condition on a category that goes by the name of **gaunt**.

Functors and natural transformations.

defn. For precategories A and B a **functor** $F\colon A \to B$ consists of

- A function $F_0 : A_0 \to B_0$ generally also denoted by F.
- For each $a, b : A$, a function $F_{a,b} : \hom_A(a, b) \to \hom_B(Fa, Fb)$.
- Proofs, for each $a : A$, that $F(1_a) = 1_{Fa}$.
- Proofs, for each $a, b, c : A$, $f : \hom_A(a, b)$ and $g : \hom_A(b, c)$ that $F(g \circ f) = Fg \circ Ff$.

defn. For functors $F, G : A \to B$ between precategories a **natural transformation** $\gamma\colon F \to G$ consists of

- A dependent function $\gamma : \prod_{a:A} \hom_B(Fa, Ga)$ defining the components of the natural transformation.
- For each $a, b : A$ and $f : \hom_A(a, b)$, a proof that $Gf \circ \gamma_a = \gamma_b \circ Ff$, the naturality axiom.

defn. For precategories A and B there is a pre-category B^A defined by taking $(B^A)_0$ to be the type of functors $A \to B$ and $\hom_{B^A}(F, G)$ to be the type of natural transformations from F to G. Note that since the hom types in precategories are sets, two natural transformations $\gamma, \gamma' : F \to G$ are equal just when their components are equal. In particular, by function extensionality, the type of natural transformations is a set.

The identity element 1_F is the natural transformation whose components $(1_F)_a :\equiv 1_{Fa}$ are identities. Similarly, composition of natural transformations is componentwise.

The traditional categorical proof shows:

Lemma. *A natural transformation $\gamma\colon F \to G$ is an isomorphism if and only if each of its components are isomorphisms.* □

Using this, we may show

Theorem. *If A is a precategory and B is a category then B^A is a category.*

Proof. For functors $F, G : A \to B$ we must show that $\mathtt{iso\text{-}id} : (F = G) \to (F \cong G)$ is an equivalence. To define its inverse suppose $\gamma : F \cong G$. By the lemma this means we have an isomorphism $\gamma_a : \hom_B(Fa, Ga)$ for every $a : A$. Since B is a category, we have identities $\mathtt{id\text{-}iso}(\gamma_a) : Fa =_B Ga$. By function extensionality this proves that $\bar{\gamma} : F_0 =_{A_0 \to B_0} G_0$.

Our next task is to show that $F_{a,b} : \hom_A(a, b) \to \hom_B(Fa, Fb)$ equals $G_{a,b} : \hom_A(a, b) \to \hom_B(Ga, Gb)$ after transporting the former along $\bar{\gamma}$. In fact this is all that remains to show since the final axioms of a functor are propositions. By a lemma below, for $f : \hom_A(a, b)$ the transport of $Ff : \hom_B(Fa, Fb)$ along $\bar{\gamma}$ is equal to the composite $\gamma_b \circ Ff \circ \gamma_a^{-1} : \hom_B(Ga, Gb)$. By naturality and is-iso(γ_a), we have

$$\gamma_b \circ Ff \circ \gamma_a^{-1} = Gf \circ \gamma_a \circ \gamma_a^{-1} = Gf$$

as required.

This defines a function $(F \cong G) \to (F = G)$. A further argument is needed to prove that this function is an inverse equivalence to $\mathtt{iso\text{-}id} : (F = G) \to (F \cong G)$. See [UF, 9.2.5]. □

Lemma. *For a precategory A, $f : \hom_A(a,b)$ and $p : a = a'$ and $q : b = b'$*

$$\mathtt{tr}_{\hom}((p,q),f) = \mathtt{iso\text{-}id}(q) \circ f \circ \mathtt{iso\text{-}id}(p)^{-1}.$$

Proof. By path induction, we may assume p is $\mathtt{refl}_a$ and q is $\mathtt{refl}_b$ in which case the transport of f is just f. We have $\mathtt{iso\text{-}id}(\mathtt{refl}) = \mathtt{id}$ and $f = \mathtt{id}_b \circ f \circ \mathtt{id}_a^{-1}$ so the result follows. □

Equivalence of categories. We now study the various ways in which a functor $F\colon A \to B$ can define an equivalence of categories. As with the question of when a function defines an equivalence of types, there is some subtlety in formulating a *proposition* is-equiv(F). One solution is to use the concept of an adjunction.

defn. A functor $F\colon A \to B$ between precategories is a **left adjoint** when there exists:

- a functor $G\colon B \to A$,
- a natural transformation $\eta\colon 1_A \to G \circ F$.
- a natural transformation $\epsilon\colon F \circ G \to 1_B$.
- proofs that $G\epsilon \cdot \eta G = \mathtt{id}_G$ and $\epsilon F \cdot F\eta = \mathtt{id}_F$.

This definition uses various concepts we have not defined: identity functors, composition of functors, whiskering of functors and natural transformations, and identity natural transformations.

Importantly:

Lemma. *If $F\colon A \to B$ is a functor and A is a category then* is-left-adjoint(F) *is a proposition.*

Proof. Given two terms $(G,\eta,\epsilon),(G',\eta',\epsilon') :$ is-left-adjoint(F) the standard category theoretic argument constructs a unique natural isomorphism $\gamma : G \to G'$ commuting with the natural transformations. If A is a category then A^B is a category so $\mathtt{id\text{-}iso} : G \cong G' \to G = G'$ converts γ into an identity so that the transport of η and ϵ along this path is identifiable with η' and ϵ'. □

defn. A functor $F\colon A \to B$ defines an **equivalence** of precategories if F is a left adjoint in which η and ϵ are isomorphisms.

We write $A \simeq B \coloneqq \Sigma_{F:A\to B}$is-equiv($F$), with the sum over functors between precategories, for the type of equivalences from A to B. We state the following results and refer to [UF] for their proofs.

Lemma. *For precategories A and B and a functor $F\colon A \to B$ the following types are equivalent:*

(i) is-equiv(F) *asserting that F is an equivalence of precategories.*

(ii) The type

$$\left(\Pi_{a,a':A}\text{is-equiv}(F_{a,a'})\right) \times (\Pi_{b:B}\Sigma_{a:A}Fa \cong b)$$

*asserting that F is **fully faithful**, meaning each $F_{a,a'} : \hom_A(a,a') \to \hom_B(Fa,Fa')$ is an equivalence, and **split essentially surjective**, meaning $\Pi_{b:B}\Sigma_{a:A}Fa \cong b$.*

If A is a category and F is fully faithful then for any $b : B$ the type $\Sigma_{a:A}Fa \cong b$ is a proposition: if given $(a,f),(a',f') : \Sigma_{a:A}Fa \cong b$ then $f'^{-1} \circ f\colon Fa \cong Fa'$ is an isomorphism. Since F is fully faithful then $a \cong a'$ and since A is a category $a = a'$ and one can verify that transport along this path identifies f with f'. In particular, in this context the notion of split essentially surjective could be replaced by the notion of essentially surjective, meaning $\Pi_{b:B}\|\Sigma_{a:A}Fa \cong b\|$, without change.

defn. A functor $F\colon A \to B$ is an **isomorphism** of precategories if F is fully faithful and $F_0 : A_0 \to B_0$ is an equivalence of types.

We write $A \cong B$ for the type of isomorphisms between precategories.

Lemma. *For categories A and B, a functor $F\colon A \to B$ is an equivalence if and only if it is an isomorphism of categories.*

By univalence:

Lemma. *If A and B are precategories then the function*

$$\mathtt{iso\text{-}id} : (A = B) \to (A \cong B)$$

defined by induction by $\mathtt{refl}_A \mapsto 1_A$ is an equivalence.

Thus:

Theorem. *If A and B are categories, then the function*

$$\texttt{equiv-id} : (A = B) \to (A \simeq B)$$

defined by induction by $\texttt{refl}_A \mapsto 1_A$ *is an equivalence.*

November 15: The real numbers in univalent mathematics

In this section we'll describe and compare two different approaches to constructing the real numbers in univalent mathematics following [UF, Chapter 10]. The first of these constructs the *Cauchy reals* using Cauchy sequences, while the second constructs the *Dedekind reals*, using Dedekind cuts. Our constructions will use some properties of the universe $\mathrm{Set}_{\mathcal{U}}$ of sets that we have not proven, which we will provide references for.

The rational numbers. To define the rational numbers we need to know how to form set quotients in homotopy type theory. Let A be a set and let $R \colon A \to A \to \mathrm{Prop}_{\mathcal{U}}$ be a propositional relation on A that is reflexive, symmetric, and transitive, as discussed above.

defn. Define A/R to be the higher inductive type generated by

(i) A function $q \colon A \to A/R$.
(ii) For each $a, b : A$ so that $R(a, b)$, an equality $q(a) = q(b)$.
(iii) The 0-truncation constructor: for all $x, y : A/R$ and $r, s : x = y$, an equality $r = s$.

Another equivalent construction is described in [R, §18.1]. For example, this construction gives us a new way to think about the type of integers $\mathbb{Z}$.

ex. We may define the integers $\mathbb{Z}$ as the set quotient $\mathbb{N} \times \mathbb{N}_{/\sim}$ where $\sim$ is the equivalence relation defined by

$$(a, b) \sim (c, d) :\equiv (a + d = b + c).$$

In this case, there are canonical representatives for equivalence classes: each $(a, b) : \mathbb{N} \times \mathbb{N}$ is equivalent to a pair of the form $(n, 0)$ or $(0, n)$ dependent on whether $a \geqslant b$ or not (and this relation is decidable). The function $r \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ defined by

$$r(a, b) :\equiv \begin{cases} (a - b, 0) & a \geqslant b \\ (0, b - a) & a < b \end{cases}$$

defines an **idempotent**, meaning $r \circ r = r$.

Lemma. *Suppose $\sim$ is an equivalence relation on A and there is an idempotent $r \colon A \to A$ for all $x, y : A$ so that $(r(x) = r(y)) \simeq (x \sim y)$. Then the type*

$$A/\sim :\equiv \Sigma_{x:A} r(x) = x$$

is the set quotient of A by $\sim$. In other words, there is a map $q \colon A \to A/\sim$ such that for every set B the type $(A/\sim) \to B$ is equivalent to

$$\Sigma_{g:A\to B} \Pi_{x,y:A} (x \sim y) \to (g(x) = g(y))$$

via precomposition with q.

See [UF, 6.10.8].

defn. Similarly we define the rationals $\mathbb{Q}$ as the quotient $(\mathbb{Z} \times \mathbb{N})/\sim$ by the equivalence relation

$$(u, a) \sim (v, b) :\equiv (u(b + 1) = v(a + 1)).$$

Here (u, a) represents the rational numbers $u/(a + 1)$, with the 1 added so we don't have to worry about division by 0.

Again we have an idempotent $r \colon \mathbb{Z} \times \mathbb{N} \to \mathbb{Z} \times \mathbb{N}$ that sends a pair (u, a) to $r(u, a) :\equiv (u', a')$ where $u'/(a' + 1)$ is the same fraction as $u/(a + 1)$ but in lowest terms. It follows that $\mathbb{Q}$ is a set with decidable equality and decidable order.

Dedekind reals. The Dedekind reals are defined to be the set of Dedekind cuts of $\mathbb{Q}$, which are defined as follows.

defn. A **Dedekind cut** consists of a pair $L : \mathbb{Q} \to \mathrm{Prop}_{\mathcal{U}}$ and $U : \mathbb{Q} \to \mathrm{Prop}_{\mathcal{U}}$ of predicates, called the **lower** and **upper cut**, respectively, which are:

(i) inhabited:
$$\|\Sigma_{q:\mathbb{Q}} L(q)\| \qquad \text{and} \qquad \|\Sigma_{r:\mathbb{Q}} U(r)\|$$
(ii) rounded: for all $q, r : \mathbb{Q}$
$$L(q) \leftrightarrow \|\Sigma_{r:\mathbb{Q}}(q < r) \times L(r)\| \qquad \text{and} \qquad U(r) \leftrightarrow \|\Sigma_{q:\mathbb{Q}}(q < r) \times U(r)\|.$$
(iii) disjoint:
$$\Pi_{q:\mathbb{Q}} \neg(L(q) \times U(q)).$$
(iv) located:
$$\Pi_{q,r:\mathbb{Q}}(q < r) \to L(q) \vee U(r).$$

Define is-Cut(L, U) to be the conjunction of these conditions. The type of **Dedekind reals** is
$$\mathbb{R}_d \coloneqq \Sigma_{L,U:\mathbb{Q}\to \mathrm{Prop}_{\mathcal{U}}} \text{is-Cut}(L, U).$$

Remark. The problem with using $\mathrm{Prop}_{\mathcal{U}}$ is the codomain for the predicates L and U is that it makes $\mathbb{R}_d$ a type in a higher universe $\mathcal{U}^+$. This problem can be dealt with in a number of ways. One solution is to add a **propositional resizing** axiom. Our axioms for universes imply that there is a map $\mathcal{U} \to \mathcal{U}^+$ that restricts to define a map $\mathrm{Prop}_{\mathcal{U}} \to \mathrm{Prop}_{\mathcal{U}^+}$. Propositional resizing asserts that this map is a universe, which means in practice that any proposition in $\mathcal{U}^+$ can be "resized" to one in the smaller universe $\mathcal{U}$. With propositional resizing, we may as well collapse all the way down and write $\Omega : \mathcal{U}_0$ for our base universe of propositions and write $L, U : \mathbb{Q} \to \Omega$.

Another option is to assert the law of excluded middle for propositions in which case it's possible to use $\Omega = \mathbb{1} + \mathbb{1}$ for the universe of propositions.

A third option is to just carefully keep track of universe levels and let $\mathbb{R}_d$ live where it lives. We adopt one of these solutions and won't worry about this anymore.

Note since is-Cut(L, U) is a proposition, $\mathbb{R}_d$ is a set. There is an embedding $\mathbb{Q} \to \mathbb{R}_d$ which associates each $q : \mathbb{Q}$ with the cut
$$L_q(r) \coloneqq (r < q) \qquad \text{and} \qquad U_q(r) \coloneqq (q < r).$$

The Cauchy reals. The Cauchy reals are defined to be the completion of $\mathbb{Q}$ under limits of Cauchy sequences. The classical construction first considers the set of all Cauchy sequences in $\mathbb{Q}$ and then forms a quotient under a suitable equivalence relation. To prove that this quotient is Cauchy complete one must consider a Cauchy sequence in the quotient, lift it to a Cauchy sequence of Cauchy sequences, and construct the limit using this lift. However, the lifting step requires the axiom of countable choice or the law of excluded middle, which we may wish to avoid. The standard way out of this conundrum is either to

(i) Form the Cauchy reals as a setoid: in this case, a Cauchy real is a Cauchy sequence of rationals, and the equivalence relation is carried through all constructions.
(ii) Accept the axiom of countable choice.
(iii) Use the Dedekind reals instead.

Homotopy type theory presents a fourth alternative using higher inductive types. Essentially we define the Cauchy reals to be the free complete metric space generated by $\mathbb{Q}$, which is formed by iteratively attaching limits to Cauchy sequences. In the presence of the axiom of choice, you can prove that this set must only be done once, but this gadget can be reasoned about even in the absence of a more explicit construction.

We will define the Cauchy reals $\mathbb{R}_c$ and a relation $\sim_\epsilon : \mathbb{R}_c \to \mathbb{R}_c \to \mathrm{Prop}_{\mathcal{U}}$ for any positive rational number $\epsilon : \mathbb{Q}_+$ whose intended meaning is $x \sim_\epsilon y$ if the distance between x and y is less that ϵ. This requires a *higher inductive-inductive definition*.

Remark. An inductive-inductive definition defines a type A and a type family $B : A \to \mathcal{U}$ simultaneously by means of an induction in which constructors of the $B(a)$s and of A itself can take inputs from the other types. These are closely related to inductive-recursive types, where the type family B is defined recursively using the constructors of A (which may take inputs from B). It is implemented in `agda` but still considered somewhat experimental.

defn. A function $x\colon \mathbb{N} \to \mathbb{Q}$ is a **Cauchy sequence** when it satisfies

$$\Pi_{\epsilon:\mathbb{Q}_+}\Sigma_{n:\mathbb{N}}\Pi_{m,k\geqslant n}|x_m - x_k| < \epsilon.$$

When this holds, then by the type theoretic axiom of choice there is a function $M : \mathbb{Q}_+ \to \mathbb{N}$ called the **modulus of convergence** that sends ϵ to a natural number $M(\epsilon)$ so that $m, k \geqslant M(\epsilon)$ implies that $|x_m - x_k| < \epsilon$. Note in particular that for any $\epsilon, \delta : \mathbb{Q}_+$ that $|x_{M(\epsilon/2)} - x_{M(\delta/2)}| < \epsilon + \delta$ so the map $\epsilon \mapsto x_{M(\epsilon/2)} : \mathbb{Q}_+ \to \mathbb{Q}$ carries the same information about the limit as the original Cauchy sequence did. We refer to this data as a **Cauchy approximation**.

defn. Let $\mathbb{R}_c$ and the relation $\sim\colon \mathbb{Q}_+ \times \mathbb{R}_c \times \mathbb{R}_c \to \mathcal{U}$ be the higher inductive-inductive type family generated by the following constructors:

- rational points: for any $q : \mathbb{Q}$ there is a real $\mathtt{rat}(q) : \mathbb{R}_c$.
- limit points: for any $x : \mathbb{Q}_+ \to \mathbb{R}_c$ such that

  $$\Pi_{\delta,\epsilon:\mathbb{Q}_+} x_\delta \sim_{\delta+\epsilon} x_\epsilon$$

  there is a point $\mathtt{lim}(x) : \mathbb{R}_c$. We call x a **Cauchy approximation**.
- paths: for $u, v : \mathbb{R}_c$ such that

  $$\Pi_{\epsilon:\mathbb{Q}_+} u \sim_\epsilon v$$

  there is a path $\mathtt{eq}_{\mathbb{R}_c}(u, v) : u =_{\mathbb{R}_c} v$.

Simultaneously, the type family $\sim\colon \mathbb{R}_c \to \mathbb{R}_c \to \mathbb{Q}_+ \to \mathcal{U}$ is generated by the following constructors for all $q, r : \mathbb{Q}$, $\delta, \epsilon, \eta : \mathbb{Q}_+$, $u, v : \mathbb{R}_c$, and Cauchy approximations x and y:

- for all q, r, ϵ if $-\epsilon < q - r < \epsilon$ then $\mathtt{rat}(q) \sim_\epsilon \mathtt{rat}(r)$
- for all q, y, ϵ, δ if $\mathtt{rat}(q) \sim_{\epsilon-\delta} y_\delta$ then $\mathtt{rat}(q) \sim_\epsilon \mathtt{lim}(y)$
- for all x, r, ϵ, δ if $x_\delta \sim_{\epsilon-\delta} \mathtt{rat}(r)$ then $\mathtt{lim}(x) \sim_\epsilon \mathtt{rat}(r)$
- for all $x, y, \epsilon, \delta, \eta$ if $x_\delta \sim_{\epsilon-\delta-\eta} y_\eta$ then $\mathtt{lim}(x) \sim_\epsilon \mathtt{lim}(y)$
- for all u, v, ϵ, if $\zeta, \xi : u \sim_\epsilon v$ then $\zeta = \xi$. This is propositional truncation on the relation $\sim$.

Comparison between the Cauchy and Dedekind reals.

Theorem. *There is an embedding of ordered fields $\mathbb{R}_c \to \mathbb{R}_d$ which fixes the rational numbers.*

There are two proofs, both discussed in [UF]. One strategy is to prove the universal property of $\mathbb{R}_d$, namely that it is the terminal archimedean ordered field F for which the strict order $<$ on F is a map $<\colon F \to F \to \Omega$, where Ω is whatever universe of propositions was used to define Dedekind cuts. Here the archimedian principle states that for all $x, y : F$ if $x < y$ then there merely exists some $q : \mathbb{Q}$ so that $x < q < y$. Then you show that $\mathbb{R}_c$ is also an archimediate ordered field with $<\colon \mathbb{R}_c \to \mathbb{R}_c \to \Omega$. Consequently, it may be realized as a subfield of $\mathbb{R}_d$.

An alternate strategy uses the universal property of $\mathbb{R}_c$, which states that the Cauchy reals embed into every Cauchy complete archimedian ordered field. One then shows that the archimedian ordered field $\mathbb{R}_d$ is Cauchy complete.

Without further assumptions, we do not expect $\mathbb{R}_c$ and $\mathbb{R}_d$ to coincide. However:

Lemma. *If for every $x : \mathbb{R}_d$ there merely exists*

$$c : \Pi_{q,r:\mathbb{Q}}(q < r) \to ((q < x) + (x < r))$$

then the Cauchy and Dedekind reals coincide.

Proof. It suffices to show that every Dedekind real merely is the limit of a Cauchy sequence of rational numbers.

Consider any $x : \mathbb{R}_d$. By the assumption that there merely exists a c as in the statement and by the inhabitation of cuts, there merely exists $a, b : \mathbb{Q}$ so that $a < x < b$. We construct

$$f : \mathbb{N} \to \Sigma_{(q,r):\mathbb{Q}\times\mathbb{Q}} q < r$$

by recursion. Set $f(0) = (a, b)$. Then if $f(n) := (q_n, r_n)$ with $q_n < r_n$ Define $s := (2q_n + r_n)/3$ and $t := (q_n + 2r_n)/3$ and use $c(s, t)$ to decide between $s < x$ and $x < t$. If it decides $s < x$ set $f(n + 1) := (s, r_n)$ and set $f(n + 1) := (q_n, t)$ otherwise.

By construction $q_n < x < r_n$ and $|q_n - r_n| \leqslant (2/3)^n \cdot |q_0 - r_0|$ for all $n : \mathbb{N}$. Thus both $q : \mathbb{N} \to \mathbb{Q}$ and $r\colon \mathbb{N} \to \mathbb{Q}$ define Cauchy sequence converging to the Dedekind cut x. We have shown that for every $x : \mathbb{R}_d$ there merely exists a Cauchy sequence converging to x. □

Corollary. *If excluded middle or countable choice holds then $\mathbb{R}_c$ and $\mathbb{R}_d$ are equivalent.*

Proof. If excluded middle holds then $(x < y) \to ((x < z) + (z < y))$ can be proved: either $x < z$ or $\neg(x < z)$. In the former case, we are done, while in the latter we have $z \leqslant x < y$. This allows us to apply the lemma.

If countable choice holds then we use the fact that the set $S = \Sigma_{(q,r):\mathbb{Q}\times\mathbb{Q}} q < r$ is equivalent to $\mathbb{N}$. So we may apply countable choice to the statement

$$\Pi_{(q,r):S} \|(q < x) + (x < r)\|$$

which says that x is located. This is a curried form of the statement we need. □

Part 3. **Synthetic Homotopy Theory**

In the final part of the course we tour a very small portion of the synthetic homotopy theory that has been developed in homotopy type theory, starting by studying the circle.

NOVEMBER 17: THE CIRCLE

The induction principle of the circle. Geometrically, the circle can be built by attaching a loop to a point. This definition is efficiently captured by the following higher inductive type.

defn. The circle is the type S^1 freely generated by a term $\mathtt{base} : \mathrm{S}^1$ and a loop $\mathtt{loop} : \mathtt{base} = \mathtt{base}$.

Just like for ordinary inductive types, higher inductive types come with an induction principle that can be used to construct sections of type families $B : \mathrm{S}^1 \to \mathcal{U}$. Suppose given a section $f : \Pi_{x:\mathrm{S}^1} B(x)$. Then in particular this gives a term $f(\mathtt{base}) : B(\mathtt{base})$ and a path $\mathtt{apd}_f(\mathtt{loop}) : \mathtt{tr}_B(\mathtt{loop}, f(\mathtt{base})) =_{B\mathtt{base}} f(\mathtt{base})$. We can record this data via a map

defn. For any type family $B : \mathrm{S}^1 \to \mathcal{U}$, there is a map

$$\mathtt{dgen}_{\mathrm{S}^1} : \left(\Pi_{x:\mathrm{S}^1} B(x)\right) \to \left(\Sigma_{b:B(\mathtt{base})} \mathtt{tr}_B(\mathtt{loop}, b) = b\right)$$

given by $\mathtt{dgen}_{\mathrm{S}^1}(f) \coloneqq (f(\mathtt{base}), \mathtt{apd}_f(\mathtt{loop}))$. This is the dependent action of f on the generators $\mathtt{base}$ and $\mathtt{loop}$ of S^1.

defn. The **induction principle of the circle** provides for each type family $B : \mathrm{S}^1 \to \mathcal{U}$ a map

$$\mathtt{ind}_{\mathrm{S}^1} : \left(\Sigma_{b:B(\mathtt{base})} \mathtt{tr}_B(\mathtt{loop}, b) = b\right) \to \left(\Pi_{x:\mathrm{S}^1} B(x)\right)$$

and a homotopy $\mathtt{comp}_{\mathrm{S}^1} : \mathtt{dgen}_{\mathrm{S}^1} \circ \mathtt{ind}_{\mathrm{S}^1} \sim \mathtt{id}$ witnessing that $\mathtt{ind}_{\mathrm{S}^1}$ is a section of $\mathtt{dgen}_{\mathrm{S}^1}$.

The type of identifications $(b, p) = (b', p')$ in the type $\Sigma_{b:B(\mathtt{base})} \mathtt{tr}_B(\mathtt{loop}, b) = b$ is equivalent to the type of pairs (α, β) consisting of a path $\alpha : b = b'$ and a path β defining a homotopy between the square of paths

$$\begin{array}{ccc} \mathtt{tr}_B(\mathtt{loop}, b) & \overset{\mathtt{ap}_{\mathtt{tr}_B(\mathtt{loop})}(\alpha)}{=\!=\!=} & \mathtt{tr}_B(\mathtt{loop}, b') \\ {\scriptstyle p}\Big\| & \beta & \Big\|{\scriptstyle p'} \\ b & \underset{\alpha}{=\!=\!=} & b' \end{array}$$

Thus the induction principle of the circle says that for any $b : B(\mathtt{base})$ and $p : \mathtt{tr}_B(\mathtt{loop}, b) = b$ there is a function $f : \Pi_{x:\mathrm{S}^1} B(x)$ equipped with identifications $\alpha : f(\mathtt{base}) = b$ and β defining a homotopy in the square of paths

$$\begin{array}{ccc} \mathtt{tr}_B(\mathtt{loop}, f(\mathtt{base})) & \overset{\mathtt{ap}_{\mathtt{tr}_B(\mathtt{loop})}(\alpha)}{=\!=\!=} & \mathtt{tr}_B(\mathtt{loop}, b) \\ {\scriptstyle \mathtt{apd}_f(\mathtt{loop})}\Big\| & \beta & \Big\|{\scriptstyle p} \\ f(\mathtt{base}) & \underset{\alpha}{=\!=\!=} & b \end{array}$$

The dependent universal property of the circle. The dependent universal property of the circle states that $\mathtt{ind}_{\mathrm{S}^1}$ and $\mathtt{dgen}_{\mathrm{S}^1}$ define an inverse equivalence for any type family $B : \mathrm{S}^1 \to \mathcal{U}$.

Theorem. *For any type family* $B : \mathrm{S}^1 \to \mathcal{U}$*, the map*

$$\mathtt{dgen}_{\mathrm{S}^1} : \left(\Pi_{x:\mathrm{S}^1} B(x)\right) \to \left(\Sigma_{b:B(\mathtt{base})} \mathtt{tr}_B(\mathtt{loop}, b) = b\right)$$

given by $\mathtt{dgen}_{\mathrm{S}^1}(f) \coloneqq (f(\mathtt{base}), \mathtt{apd}_f(\mathtt{loop}))$ *is an equivalence.*

Proof. The inverse equivalence and one homotopy are given by $\mathtt{ind}_{S^1}$ and $\mathtt{comp}_{S^1}$. It remains only to construct the final homotopy

$$\mathtt{ind}_{S^1} \circ \mathtt{dgen}_{S^1} \sim \mathtt{id}.$$

Thus for any $f : \prod_{x:S^1} B(x)$ we require an identification $\mathtt{ind}_{S^1}(\mathtt{dgen}_{S^1}(f)) = f$. By function extensionality, it suffices to give identifications $\mathtt{ind}_{S^1}(\mathtt{dgen}_{S^1}(f(x))) = f(x)$ for each $x : S^1$. That is, we must define a term in the type

$$\prod_{x:S^1} \mathtt{ind}_{S^1}(\mathtt{dgen}_{S^1}(f(x))) = f(x).$$

By the induction principle of the circle it suffices to construct a pair of terms

$$\alpha : \mathtt{ind}_{S^1}(\mathtt{dgen}_{S^1}(f(\mathtt{base}))) = f(\mathtt{base})$$

and

$$\beta : \mathtt{tr}(\mathtt{loop}, \alpha) = \alpha$$

where the transport is in the family $x \mapsto \mathtt{ind}_{S^1}(\mathtt{dgen}_{S^1}(f(x))) = f(x)$. By the lemma below to construct the second path it suffices instead to define a homotopy β in the square

$$\begin{array}{ccc} \mathtt{tr}_B(\mathtt{loop}, f(\mathtt{base})) & \overset{\mathsf{ap}_{\mathtt{tr}_B(\mathtt{loop})}(\alpha)}{=\!=\!=} & \mathtt{tr}_B(\mathtt{loop}, b) \\ {\scriptstyle \mathsf{apd}_f(\mathtt{loop})}\Big\| & \beta & \Big\|{\scriptstyle p} \\ f(\mathtt{base}) & \underset{\alpha}{=\!=\!=} & b \end{array}$$

But this is exactly the data we're given in the computation rule for the circle. □

Lemma. *For any type family $B : A \to \mathcal{U}$, $f, g : \prod_{x:A} B(x)$, $p : x = x'$, $q : g(x) = f(x)$ and $r : g(x') = f(x')$ there is a function*

$$\left(\mathsf{apd}_g(p) \cdot r = \mathsf{ap}_{\mathtt{tr}_B(p)}(q) \cdot \mathsf{apd}_f(p)\right) \to \left(\mathtt{tr}_{x \mapsto g(x) = f(x)}(p, q) = r\right).$$

Proof. By path induction on p it suffices to define a function

$$\left(\mathtt{refl} \cdot r = q \cdot \mathtt{refl}\right) \to \left(q = r\right)$$

which can be done by combining composing with the unit homotopies with the function $\mathtt{inv} : (r = q) \to (q = r)$. □

Note that a corollary of the equivalence is that for each $b : B(\mathtt{base})$ and $p : \mathtt{tr}_B(\mathtt{loop}, b) = b$ the type of dependent functions $f : \prod_{x:S^1} B(x)$ equipped with $\alpha : f(\mathtt{base}) = b$ and β in the square above is contractible.

Another corollary of the dependent universal property of the circle is the non-dependent universal property:

Theorem. *For any type X, the action on generators*

$$\mathtt{gen}_{S^1} : (S^1 \to X) \to \Sigma_{x:X} x = x$$

given by $f \mapsto (f(\mathtt{base}), \mathsf{ap}_f(\mathtt{loop}))$ is an equivalence.

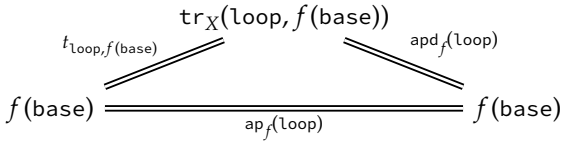
Proof. We will show that there is a commutative triangle

$$\begin{array}{ccc} & S^1 \to X & \\ {\scriptstyle \mathtt{gen}_{S^1}}\swarrow & & \searrow{\scriptstyle \mathtt{dgen}_{S^1}} \\ (\Sigma_{x:X} x = x) & \xrightarrow{\quad\simeq\quad} & (\Sigma_{x:X} \mathtt{tr}_X(\mathtt{loop}, x) = x) \end{array}$$

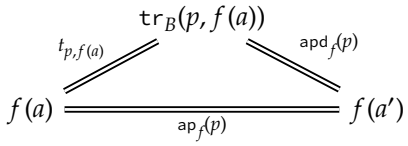
in which the bottom map is an equivalence. Here the transport is in the constant family over S^1.

We first must define the bottom map. For this note, that if B is a constant type family over A and $p : a =_A a'$ then for any $b : B$ there is an identification $t_{p,b} : \mathtt{tr}_B(p, b) = b$ defined by path induction on p by $\mathtt{refl}_a \mapsto \mathtt{refl}_b$. The bottom map is then defined by composition with $t_{p,b}$. Since composition with a path is an equivalence, this map is an equivalence, as claimed.

It remains to show that the triangle of maps commutes up to homotopy. Given $f\colon \mathrm{S}^1 \to X$, we must construct an identification in the triangle of paths



This holds for completely general reasons. For any $f : A \to B$ and $p : a =_A a'$ the triangle



commutes by path induction on p. □

Remark. Note that $\neg(\mathtt{loop} = \mathtt{refl}_{\mathtt{base}})$. Because if so then for any type X and any loop $p : x = x$ we would get a function $f : \mathrm{S}^1 \to X$ so that

$$p = \mathsf{ap}_f(\mathtt{loop}) = \mathsf{ap}_f(\mathtt{refl}_{\mathtt{base}}) = \mathtt{refl}_x,$$

which is "axiom K" and implies that X is a set. Since not all types are sets, we can't have $\mathtt{loop} = \mathtt{refl}_{\mathtt{base}}$.

Corollary. *For any loop $\ell : x = x$ in X the type of maps $f : \mathrm{S}^1 \to X$ equipped with an identification $\alpha : f(\mathtt{base}) = x$ and a homotopy β in the square*

$$\begin{array}{ccc} f(\mathtt{base}) & \overset{\alpha}{=\!=} & x \\ {\scriptstyle \mathsf{ap}_f(\mathtt{loop})}\Big\| & \beta & \Big\|{\scriptstyle \ell} \\ f(\mathtt{base}) & \underset{\alpha}{=\!=} & x \end{array}$$

is contractible.

Multiplication on the circle. Classically, the circle can be identified with the space of complex numbers at distance 1 from the origin. This space is a topological abelian group, with complex multiplication. We can define an analogous multiplication operation on the S^1 of homotopy type theory despite the fact this was constructed in a very different way.

defn. There is a binary operation $\mathtt{mul}_{\mathrm{S}^1}\colon \mathrm{S}^1 \to \mathrm{S}^1 \to \mathrm{S}^1$ defined by applying the universal property of the circle to the type $\mathrm{S}^1 \to \mathrm{S}^1$. Using the universal property of the circle we may construct a term $\mathtt{ind}_{\mathrm{S}^1}(b, p)$ in the type $\mathrm{S}^1 \to \mathrm{S}^1 \to \mathrm{S}^1$ from a term in the type

$$(b, p) : \Sigma_{e:\mathrm{S}^1 \to \mathrm{S}^1} e = e.$$

The computation rule provides identifications $\alpha : \mathtt{ind}_{\mathrm{S}^1}(b, p)(\mathtt{base}) = b$ and β. If we want $\mathtt{ind}_{\mathrm{S}^1}(b, p)$ to be $\mathtt{mul}_{\mathrm{S}^1}$ and $\mathtt{base} : \mathrm{S}^1$ to be the unit for the multiplication, then $\mathtt{mul}_{\mathrm{S}^1}(\mathtt{base})$ should be the identity function. Thus we take $\mathtt{id} : \mathrm{S}^1 \to \mathrm{S}^1$ for the term b and write $H : \mathtt{id} \sim \mathtt{id}$ for the homotopy, to be defined later, whose corresponding path $\mathtt{eq\text{-}htpy}(H) : \mathtt{id} = \mathtt{id}$ is the path p. We then define

$$\mathtt{mul}_{\mathrm{S}^1} :\equiv \mathtt{ind}_{\mathrm{S}^1}(\mathtt{id}, \mathtt{eq\text{-}htpy}(H)).$$

It takes some effort to define the correct homotopy H, for which we're guided by the computation rule which tells us that the function $\mathtt{mul}_{\mathrm{S}^1}$ is related to the generating data $(\mathtt{id}, \mathtt{eq\text{-}htpy}(H))$ by paths and homotopies

$$\begin{array}{ccc} \mathtt{mul}_{\mathrm{S}^1}(\mathtt{base}) & \overset{\mathtt{base\text{-}mul}_{\mathrm{S}^1}}{=\!=} & \mathtt{id} \\ {\scriptstyle \mathsf{ap}_{\mathtt{mul}_{\mathrm{S}^1}}(\mathtt{loop})}\Big\| & \mathtt{loop\text{-}mul}^1_{\mathrm{S}} & \Big\|{\scriptstyle \mathtt{eq\text{-}htpy}(H)} \\ \mathtt{mul}_{\mathrm{S}^1}(\mathtt{base}) & \underset{\mathtt{base\text{-}mul}_{\mathrm{S}^1}}{=\!=} & \mathtt{id} \end{array}$$

where $\texttt{base-mul}_{\mathbb{S}^1}$ is the path α above and $\texttt{loop-mul}_{\mathbb{S}^1}$ is β. Applying the dependent universal property of the circle we may define H to be the unique dependent function $H : \prod_{x:\mathbb{S}^1} x = x$ equipped with an identification $\alpha : H(\texttt{base}) = \texttt{loop}$ and an identification β in the square

$$\begin{array}{ccc} \texttt{tr}(\texttt{loop}, H(\texttt{base})) & \overset{\texttt{ap}_{\texttt{tr}(\texttt{loop})}(\alpha)}{=\!=\!=} & \texttt{tr}(\texttt{loop},\texttt{loop}) \\ {\scriptstyle \texttt{apd}_H(\texttt{loop})}\Big\| & \beta & \Big\|{\scriptstyle \gamma} \\ H(\texttt{base}) & \underset{\alpha}{=\!=\!=} & \texttt{loop} \end{array}$$

where the transport is in the family $x \mapsto x = x : \mathbb{S}^1 \to \mathcal{U}$. Now it just remains to define the path $\gamma : \texttt{tr}(\texttt{loop},\texttt{loop}) = \texttt{loop}$.

By the lemma in the previous section we have a function

$$(p \cdot r = q \cdot p) \to \left(\texttt{tr}_{x \mapsto x=x}(p,q) = r\right)$$

for any $p : \texttt{base} = x$, $q : \texttt{base} = \texttt{base}$, and $r : x = x$. In particular, we have a function

$$(\texttt{loop} \cdot \texttt{loop}) = \texttt{loop} \cdot \texttt{loop}) \to (\texttt{tr}_{x \mapsto x=x}(\texttt{loop},\texttt{loop}) = \texttt{loop})\,.$$

We apply this to $\texttt{refl}_{\texttt{loop}\cdot\texttt{loop}}$ to obtain the desired identification γ.

What just happened? By path induction on p, transport $\texttt{tr}_{x \mapsto x=x}(p,q)$ in the family $x \mapsto x = x$ of a term $q : a = a$ along a path $p : a = b$ is identifiable with the path $p^{-1} \cdot q \cdot p : b = b$. So γ may be identified with a path of paths $\gamma : \texttt{loop}^{-1} \cdot \texttt{loop} \cdot \texttt{loop} = \texttt{loop}$, namely the path defined by canceling the inverses.

It takes some effort to see that this function deserves to be called "complex multiplication."

As evidence, note first that the path $\texttt{base-mul}_{\mathbb{S}^1}$ proves the left unit law $\texttt{mul}_{\mathbb{S}^1}(\texttt{base}, x) = x$ for each $x : \mathbb{S}^1$.

Proposition. *The function* $\texttt{mul}_{\mathbb{S}^1}$ *satisfies the right unit law: we have*

$$\texttt{mul}_{\mathbb{S}^1}(x, \texttt{base}) = x$$

for all $x : \mathbb{S}^1$.

Proof. We prove this by induction on the circle. In the base case we have $\texttt{mul}_{\mathbb{S}^1}(\texttt{base},\texttt{base}) = \texttt{base}$ by the left unit law $\texttt{left-unit}_{\mathbb{S}^1}(\texttt{base})$. So it remains to show that $\texttt{tr}_P(\texttt{loop}, \texttt{left-unit}_{\mathbb{S}^1}(\texttt{base})) = \texttt{left-unit}_{\mathbb{S}^1}(\texttt{base})$ where $P : \mathbb{S}^1 \to \mathcal{U}$ is the family defined by $P(x) :\equiv \texttt{mul}_{\mathbb{S}^1}(x, \texttt{base}) = x$. To construct this identification it suffices to define a homotopy in the square

$$\begin{array}{ccc} \texttt{mul}_{\mathbb{S}^1}(\texttt{base},\texttt{base}) & \overset{\texttt{left-unit}_{\mathbb{S}^1}(\texttt{base})}{=\!=\!=} & \texttt{base} \\ {\scriptstyle \texttt{htpy-eq}(\texttt{ap}_{\texttt{mul}_{\mathbb{S}^1}}(\texttt{loop}))(\textit{base})}\Big\| & & \Big\|{\scriptstyle \texttt{loop}} \\ \texttt{mul}_{\mathbb{S}^1}(\texttt{base},\texttt{base}) & \underset{\texttt{left-unit}_{\mathbb{S}^1}(\texttt{base})}{=\!=\!=} & \texttt{base} \end{array}$$

From our construction of the homotopy H we have an identification $\alpha : H(\texttt{base}) = \texttt{loop}$ (which proves that H is not the trivial homotopy). Thus there is a homotopy in the previous square if and only if there is one in the square

$$\begin{array}{ccc} \texttt{mul}_{\mathbb{S}^1}(\texttt{base},\texttt{base}) & \overset{\texttt{left-unit}_{\mathbb{S}^1}(\texttt{base})}{=\!=\!=} & \texttt{base} \\ {\scriptstyle \texttt{htpy-eq}(\texttt{ap}_{\texttt{mul}_{\mathbb{S}^1}}(\texttt{loop}))(\textit{base})}\Big\| & & \Big\|{\scriptstyle H(\texttt{base})} \\ \texttt{mul}_{\mathbb{S}^1}(\texttt{base},\texttt{base}) & \underset{\texttt{left-unit}_{\mathbb{S}^1}(\texttt{base})}{=\!=\!=} & \texttt{base} \end{array}$$

and this can be constructed out of the term $\texttt{loop-mul}_{\mathbb{S}^1}$. □

November 29: The universal cover of the circle

December 1: Homotopy groups of types

December 6: Classifying types of groups

References

[R] Egbert Rijke, *Introduction to Homotopy Type Theory*, available from https://hott.zulipchat.com

[UF] *Homotopy Type Theory: Univalent Foundations of Mathematics*, the Univalent Foundations Program, Institute for Advanced Study, available from https://homotopytypetheory.org/book/

Dept. of Mathematics, Johns Hopkins University, 3400 N Charles St, Baltimore, MD 21218
Email address: eriehl@math.jhu.edu