Introduction to Homotopy Type Theory

Egbert Rijke

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

Contents

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

I

Martin-Löf's Dependent Type Theory

Dependent type theory is a formal system to organize all mathematical objects, structure, and knowledge. Dependent type theory is about types, or more generally dependent types, and their elements. There are many ways to think about type theory, types, and its elements. Types can be interpreted as sets, i.e., there is an interpretation of type theory into Zermelo-Fraenkel set theory, but there are some important differences between type theory and set theory, and the interpretation of types as sets has significant limitations. One of the differences is that in type theory, every element comes equipped with its type. We will write $a : A$ for the judgment that a is an element of type A. This leads us to a second important difference between type theory and set theory. Set theory is axiomatized in the formal system of first order logic, whereas type theory is its own formal system. Types and their elements are constructed by following the rules of this formal system, and the only way to construct an element is to construct it as an element of a previously constructed type. The expression $a : A$ is therefore not considered to be a proposition, i.e., something which one can assert about an arbitrary element and an arbitrary type, but it is considered to be a judgment, i.e., an assessment that is part of the construction of the element $a : A$.

In type theory there is a much stronger focus on equality of elements than there is in set theory. It is said that a type is not fully understood until (i) one understands how to construct an element of the type and (ii) one understands precisely how to show that two elements of the type are equal. Equality in type theory is governed by the identity type. Unlike in classical set theory, where equality is a decidable proposition of first order logic, the *type* $x = y$ of identifications of two elements $x, y : A$ is itself a type, and therefore it could posses intricate further structure.

Dependent type theory is built up in several stages. At the first stage we give structural rules, which express the general theory of type dependency. There is

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

no ambient deductive system of first order logic in type theory. Type theory is its own deductive system, and the structural rules are at the heart of this system. The basic operations that are governed by the structural rules are substitution and weakening operations. After the general system of dependent type theory has been set up, we introduce the ways in which we can form types. The most fundamental class of types are dependent function types, or Π-types. They are used for practically everything. Next, we introduce the type of natural numbers, where we use type-dependency to formulate a type-theoretic version of the induction principle. By the type-theoretic nature of this induction principle, it can be used in two ways: it can be used to construct the many familiar operations on $\mathbb{N}$, such as addition and multiplication, and it can also be used to prove properties about those operations.

The next idea is that we can consider induction principles for many other types as well. This leads to the idea of more general inductive types. In Section 4 we introduce the unit type, the empty type, the booleans, coproducts, dependent pair types, and cartesian products. All of these are examples of inductive types, and their induction principles can be used to construct the basic operations on them, as well as to prove properties about those operations.

Then we come to the most characteristic ingredient of Martin Löf's dependent type theory: the identity type. The identity type $x =_A y$ is an example of a *dependent* type, because it is indexed by $x, y : A$, and it is inductively generated by the reflexivity element $\mathsf{refl}_x : x =_A x$. The catch is, however, that the identity type $x =_A y$ is just another type, and it could potentially have many different elements.

The last class of types that we introduce are universes. Universes are type families that are closed under the operations of type theory: Π-types, Σ-types, identity types, and so on. Universes play a fundamental role in the theory. One important reason for introducing universes is that they can be used to define type families over inductive types via their induction principles. For example, this allows us to define the ordering relations $\leq$ and $<$ on the natural numbers. We will also use the universes to show the Peano axioms asserting that $\mathsf{succ}_{\mathbb{N}}$ is injective, and that $0_{\mathbb{N}}$ is not a successor.

In the final two sections of this chapter, we start developing mathematics in type theory. In Section 7.2 we study the Curry-Howard interpretation, and use it to develop modular arithmetic in type theory. In Section 8 we study the concept of decidability, and use it to obtain basic theorems in elementary number theory, such as the well-ordering theorem, the construction of the greatest common divisor, and the infinitude of primes. Both of these sections can be viewed as tutorials in type theory, designed to give you some practical experience with

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

type theory before diving into the intricacies of the univalent foundations for mathematics.

1 Dependent type theory

Dependent type theory is a system of inference rules that can be combined to make *derivations*. In these derivations, the goal is often to construct a term of a certain type. Such a term can be a function if the type of the constructed term is a function type; a proof of a property if the type of the constructed term is a proposition; an identification if the type of the constructed term is an identity type, and so on. In some respect, a type is just a collection of mathematical objects and constructing terms of a type is the everyday mathematical task or challenge. The system of inference rules that we call type theory offers a principled way of engaging in mathematical activity.

1.1 Judgments and contexts in type theory

A mathematical argument or construction consists of a sequence of deductive steps, each one using finitely many premises in order to get to the next stage in the proof or construction. Such steps can be represented by **inference rules**, which are written in the form

$$\frac{\mathcal{H}_1 \quad \mathcal{H}_2 \quad \ldots \quad \mathcal{H}_n}{C.}$$

Inference rules contain above the horizontal line a finite list $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n$ of *judgments* for the premises, and below the horizontal line a single judgment C for the conclusion. The system of dependent type theory is described by a set of such inference rules.

A straightforward example of an inference rule that we will encounter in Section 2 when we introduce function types, is the inference rule

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash f : A \to B}{\Gamma \vdash f(a) : B.}$$

This rule asserts that in any context Γ we may use a term $a : A$ and a function $f : A \to B$ to obtain a term $f(a) : B$. Each of the expressions

$$\Gamma \vdash a : A$$
$$\Gamma \vdash f : A \to B$$
$$\Gamma \vdash f(a) : B$$

are examples of judgments.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

Definition 1.1.1 There are four kinds of **judgments** in Martin-Löf's dependent type theory:

(i) *A is a (well-formed)* ***type*** *in context* Γ. We express this judgment as

$$\Gamma \vdash A \text{ type}.$$

(ii) *A and B are* ***judgmentally equal types*** *in context* Γ. We express this judgment as

$$\Gamma \vdash A \doteq B \text{ type}.$$

(iii) *a is a (well-formed)* ***term*** *of type A in context* Γ. We express this judgment as

$$\Gamma \vdash a : A.$$

(iv) *a and b are* ***judgmentally equal terms*** *of type A in context* Γ. We express this judgment as

$$\Gamma \vdash a \doteq b : A.$$

We see that any judgment is of the form $\Gamma \vdash \mathcal{J}$, consisting of a *context* Γ and a *judgment thesis* $\mathcal{J}$ asserting either that A is a type, that A and B are equal types, that a is a term of type A, or that a and b are equal terms of type A. The role of a context is to declare what hypothetical terms are assumed, along with their types. Hypothetical terms are commonly called **variables**.

Definition 1.1.2 A **context** is a finite list of variable declarations

$$x_1 : A_1,\ x_2 : A_2(x_1),\ \ldots,\ x_n : A_n(x_1, \ldots, x_{n-1}) \tag{1.1.1}$$

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

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

using the inference rules of type theory. We may use variable names other than $x_1, \ldots, x_n$, as long as no variable is declared more than once.

The condition in Definition 1.1.2 that each of the hypothetical terms is assigned a well-formed type, is checked recursively. In other words, to check that a list of variable declarations as in Eq. (1.1.1) is a context, one starts on the left and works their way to the right, verifying that each hypothetical term x_k is assigned a well-formed type.

Note that there is a context of length 0, the **empty context**, which declares no variables. This context satisfies the requirement in Definition 1.1.2 vacuously. A list of variable declarations $x_1 : A_1$ of length one is a context if and only if A_1

is a type in the empty context. We will soon encounter the type $\mathbb{N}$ of natural numbers, which is an example of a type in the empty context.

The next case is that a list of variable declarations $x_1 : A_1,\ x_2 : A_2(x_1)$ of length two is a context if and only if A_1 is a type in the empty context, and $A_2(x_1)$ is a type in context $x_1 : A_1$. This process repeats itself for longer contexts.

1.2 Type families

It is a feature of *dependent* type theory that all judgments are context dependent, and indeed that even the types of the variables in a context may depend on any previously declared variables. For example, if n is a natural number and we know from the context that n is prime, then we don't have enough information yet to decide whether or not n is odd. However, if we also know from the context that $n + 2$ is prime, then we can derive that n must be odd. Context dependency is everywhere – not only in mathematics, but also in language and in everyday life – and it gives rise to the notion of *type families* and their *sections*.

Definition 1.2.1 Consider a type A in context Γ. A **family** of types over A in context Γ is a type $B(x)$ in context $\Gamma, x : A$. In other words, in the situation where

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

we say that B is a family of types over A in context Γ. Alternatively, we say that $B(x)$ is a type **indexed** by $x : A$, in context Γ.

We think of a type family B over A in context Γ as a type $B(x)$ varying along $x : A$. A basic example of a type family occurs when we introduce *identity types* in Section 5. They are introduced as follows:

$$\frac{\Gamma \vdash a : A}{\Gamma,\ x : A \vdash a = x\ \text{type}.}$$

This rule asserts that given a term $a : A$ in context Γ, we may form the type $a = x$ in context $\Gamma,\ x : A$. The type $a = x$ in context $\Gamma,\ x : A$ is an example of a type family over A in context Γ.

Definition 1.2.2 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 $\Gamma, x : A$, i.e., in 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. Alternatively, we say that $b(x)$ is a term of type $B(x)$ **indexed** by $x : A$ in context Γ.

Note that in the above situations A, B, and b also depend on the variables declared in the context Γ, even though we have not explicitly mentioned them. It is indeed common practice to not mention every variable in the context Γ in such situations.

1.3 Inference rules

We are now ready to present the system of inference rules that underlies dependent type theory. These rules are known as the **structural rules** of type theory, since they establish the basic mathematical framework for type dependency. There are five sets of inference rules:

(i) Rules postulating that judgmental equality is an equivalence relation.
(ii) Variable conversion rules.
(iii) Substitution rules.
(iv) Weakening rules.
(v) The generic element.

Judgmental equality is an equivalence relation

The rules postulating that judgmental equality on types and on terms is an equivalence relation simply postulate that these relations are reflexive, symmetric, and transitive:

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

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a \doteq a : A} \qquad \frac{\Gamma \vdash a \doteq b : A}{\Gamma \vdash b \doteq a : A} \qquad \frac{\Gamma \vdash a \doteq b : A \quad \Gamma \vdash b \doteq c : A}{\Gamma \vdash a \doteq c : A.}$$

Variable conversion rules

The **variable conversion rules** are rules postulating that we can convert the type of a variable to a judgmentally equal type. The first variable conversion rule states that

$$\frac{\Gamma \vdash A \doteq A' \text{ type} \qquad \Gamma,\ x : A,\ \Delta \vdash B(x) \text{ type}}{\Gamma,\ x : A',\ \Delta \vdash B(x) \text{ type}.}$$

In this conversion rule, the context $\Gamma,\ x : A,\ \Delta$ is just any extension of the context $\Gamma,\ x : A$, i.e., it is a context of the form

$$x_1 : A_1,\ \ldots,\ x_{n-1} : A_{n-1},\ x : A,\ x_{n+1} : A_{n+1},\ \ldots,\ x_{n+m} : A_{n+m}.$$

Similarly, there are variable conversion rules for judgmental equality of types, for terms, and for judgmental equality of terms. To avoid having to state

essentially the same rule four times, we state all four variable conversion rules at once using a *generic judgment thesis* $\mathcal{J}$, which can be any of the four kinds described in Definition 1.1.1:

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

An analogous *term conversion rule*, stated in Exercise 1.1, converting the type of a term to a judgmentally equal type, is derivable using the rules from the rules presented in this section.

Substitution

Consider a term $f(x) : B(x)$ indexed by $x : A$ in context Γ, and suppose we also have a term $a : A$. Then we can simultaneously substitute a for all occurrences of x in $f(x)$ to obtain a new term $f[a/x]$, which has type $B[a/x]$. A precise definition of substitution requires us to get too deep into the theory of the syntax of type theory, but a mathematician is of course no stranger to substitution. For example, substituting 0 for x in the polynomial

$$1 + x + x^2 + x^3$$

results in the number $1 + 0 + 0^2 + 0^3$, which can be computed to the value 1.

Type theoretic substitution is similar. Type theoretic substitution is in fact a bit more general than what we have described above. Suppose we have a type

$$\Gamma,\ x : A,\ y_1 : B_1,\ \ldots,\ y_n : B_n \vdash C \text{ type}$$

and a term $a : A$ in context Γ. Then we can simultaneously substitute a for all occurrences of x in the types $B_1, \ldots, B_n$ and C, to obtain

$$\Gamma,\ y_1 : B_1[a/x],\ \ldots,\ y_n : B_n[a/x] \vdash C[a/x] \text{ type}.$$

Note that the variables $y_1, \ldots, y_n$ are assigned new types after performing the substitution of a for x. Similarly, we can substitute a for x in a term $c : C$ to obtain the term $c[a/x] : C[a/x]$, and we can substitute a for x in a judgmental equality thesis, either of types or terms, by simply substituting on both sides of the equation. The **substitution rule** are therefore stated using a generic judgment $\mathcal{J}$:

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

Furthermore, we add two more 'congruence rules' for substitution, postulating that substitution by judgmentally equal terms results in judgmentally equal types and terms:

$$\frac{\Gamma \vdash a \doteq a' : A \qquad \Gamma,\ x : A,\ \Delta \vdash B \text{ type}}{\Gamma,\ \Delta[a/x] \vdash B[a/x] \doteq B[a'/x] \text{ type}}$$

$$\frac{\Gamma \vdash a \doteq a' : A \qquad \Gamma,\ x : A,\ \Delta \vdash b : B}{\Gamma,\ \Delta[a/x] \vdash b[a/x] \doteq b[a'/x] : B[a/x].}$$

To see that these rules make sense, we observe that both $B[a/x]$ and $B[a'/x]$ are types in context $\Delta[a/x]$, provided that $a \doteq a'$. This is immediate by recursion on the length of Δ.

Definition 1.3.1 When B is a family of types over A in context Γ, and if we have $a : A$, then we also say that $B[a/x]$ is the **fiber** of B at a. We will usually write $B(a)$ for the fiber of B at a.

When b is a section of the family B over A in context Γ, we call the term $b[a/x]$ the **value** of b at a. Again, we will usually write $b(a)$ for the value of b at a.

Weakening

If we are given a type A in context Γ, then any judgment made in a longer context $\Gamma,\ \Delta$ can also be made in the context $\Gamma,\ x : A,\ \Delta$, for a fresh variable x. The **weakening rule** asserts that weakening by a type A in context preserves well-formedness and judgmental equality of types and terms.

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

This process of expanding the context by a fresh variable of type A is called **weakening** (by A).

In the simplest situation where weakening applies, we have two types A and B in context Γ. Then we can weaken B by A as follows

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type}}{\Gamma,\ x : A \vdash B \text{ type}}\ W$$

in order to form the type B in context $\Gamma,\ x : A$. The type B in context $\Gamma,\ x : A$ is called the **constant family** B, or the **trivial family** B.

The generic elements

If we are given a type A in context Γ, then we can weaken A by itself to obtain that A is a type in context $\Gamma,\ x : A$. The rule for the **generic element** now asserts that any hypothetical term $x : A$ in the context $\Gamma,\ x : A$ is also a term of type A in context $\Gamma,\ x : A$.

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma,\ x : A \vdash x : A}\ \delta.$$

This rule is also known as the **variable rule**. One of the reasons for including the generic element is to make sure that the variables declared in a context—i.e., the hypothetical terms—are indeed *terms*. It also provides the *identity function* on the type A in context Γ.

1.4 Derivations

A derivation in type theory is a finite tree in which each node is a valid rule of inference. At the root of the tree we find the conclusion, and in the leaves of the tree we find the hypotheses. We give two examples of derivations: a derivation showing that any variable can be changed to a fresh one, and a derivation showing that any two variables that do not mutually depend on one another can be swapped in order.

Given a derivation with hypotheses $\mathcal{H}_1, \ldots, \mathcal{H}_n$ and conclusion $\mathcal{C}$, we can form a new inference rule

$$\frac{\mathcal{H}_1 \quad \cdots \quad \mathcal{H}_n}{\mathcal{C}.}$$

Such a rule is called **derivable**, because we have a derivation for it. In order to keep proof trees reasonably short and manageable, we use the convention that any derived rules can be used in future derivations.

Changing variables

Variables can always be changed to fresh variables. We show that this is the case by showing that the inference rule

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

is derivable, where x' is a variable that does not occur in the context $\Gamma,\ x : A,\ \Delta$.

Indeed, we have the following derivation using substitution, weakening, and the generic element:

$$\dfrac{\dfrac{\Gamma \vdash A\ \mathrm{type}}{\Gamma,\ x' : A \vdash x' : A}\ \delta \qquad \dfrac{\Gamma \vdash A\ \mathrm{type} \qquad \Gamma,\ x : A,\ \Delta \vdash \mathcal{J}}{\Gamma,\ x' : A,\ x : A,\ \Delta \vdash \mathcal{J}}\ W}{\Gamma,\ x' : A,\ \Delta[x'/x] \vdash \mathcal{J}[x'/x]}\ S.$$

In this derivation it is the application of the weakening rule where we have to check that x' does not occur in the context $\Gamma,\ x : A,\ \Delta$.

Interchanging variables

The **interchange rule** states that if we have two types A and B in context Γ, and we make a judgment in context $\Gamma,\ x : A,\ y : B,\ \Delta$, then we can make that same judgment in context $\Gamma,\ y : B,\ x : A,\ \Delta$ where the order of $x : A$ and $y : B$ is swapped. More formally, the interchange rule is the following inference rule

$$\frac{\Gamma \vdash B\ \mathrm{type} \qquad \Gamma,\ x : A,\ y : B,\ \Delta \vdash \mathcal{J}}{\Gamma,\ y : B,\ x : A,\ \Delta \vdash \mathcal{J}.}$$

Just as the rule for changing variables, we claim that the interchange rule is a derivable rule.

The idea of the derivation for the interchange rule is as follows: If we have a judgment

$$\Gamma,\ x : A,\ y : B,\ \Delta \vdash \mathcal{J},$$

then we can change the variable y to a fresh variable y' and weaken the judgment to obtain the judgment

$$\Gamma,\ y : B,\ x : A,\ y' : B,\ \Delta[y'/y] \vdash \mathcal{J}[y'/y].$$

Now we can substitute y for y' to obtain the desired judgment $\Gamma,\ y : B,\ x : A,\ \Delta \vdash \mathcal{J}$. The formal derivation is as follows:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash B\ \mathrm{type}}{\Gamma,\ y : B \vdash y : B}\,\delta}{\Gamma,\ y : B,\ x : A \vdash y : B}\,W \qquad \dfrac{\Gamma \vdash B\ \mathrm{type} \qquad \dfrac{\Gamma,\ x : A,\ y : B,\ \Delta \vdash \mathcal{J}}{\Gamma,\ x : A,\ y' : B,\ \Delta[y'/y] \vdash \mathcal{J}[y'/y]}\,y'/y}{\Gamma,\ y : B,\ x : A,\ y' : B,\ \Delta[y'/y] \vdash \mathcal{J}[y'/y]}\,W}{\Gamma,\ y : B,\ x : A,\ \Delta \vdash \mathcal{J}}\,S.$$

Exercises

1.1 (a) Give a derivation for the following **term conversion rule**:

$$\frac{\Gamma \vdash A \doteq A'\ \mathrm{type} \qquad \Gamma \vdash a : A}{\Gamma \vdash a : A'.}$$

(b) Give a derivation for the following **congruence rule** for term conversion:

$$\frac{\Gamma \vdash A \doteq A'\ \mathrm{type} \qquad \Gamma \vdash a \doteq b : A}{\Gamma \vdash a \doteq b : A'.}$$

2 Dependent function types

A fundamental concept of dependent type theory is that of a dependent function. A dependent function is a function of which the type of the output may depend on the input. For example, when we concatenate a vector of length m with a vector of length n, we obtain a vector of length $m + n$. Depedent functions are a generalization of ordinary functions, because an ordinary function $f : A \to B$ is a function of which the output $f(x)$ has type B regardless of the value of x.

2.1 The rules for dependent function types

Consider a section b of a family B over A in context Γ, i.e., consider

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

From one point of view, such a section b is an operation or assignment $x \mapsto b(x)$, or a program, that takes as input $x : A$ and produces a term $b(x) : B(x)$. From a more mathematical point of view we see b as a choice of an element of each $B(x)$. In other words, we may see b as a function that takes $x : A$ to $b(x) : B(x)$. Note that the type $B(x)$ of the output may depend on $x : A$. The assignment $x \mapsto b(x)$ is in this sense a *dependent* function. The type of all such dependent functions is called the **dependent function type**, and we will write

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

for the type of dependent functions. There are four principal rules for Π-types:

(i) The *formation rule*, which tells us how we may form dependent function types.
(ii) The *introduction rule*, which tells us how to introduce new terms of dependent function types.
(iii) The *elimination rule*, which tells us how to use arbitrary terms of dependent function types.
(iv) The *computation rules*, which tell us how the introduction and elimination rules interact. These computation rules guarantee that every term of a dependent function type is indeed a dependent function taking the values by which it is defined.

In the cases of the formation rule, the introduction rule, and the elimination rule, we also need rules that assert that all the constructions respect judgmental equality. Those rules are called **congruence rules**, and they are part of the specification of dependent function types.

The Π-formation rule

The **Π-formation rule** tells us how Π-types are constructed. The idea of Π-types is that for any type family B of types over A, there is a type of dependent functions $\prod_{(x:A)} B(x)$, so the Π-formation rule is as follows:

$$\frac{\Gamma, x : A \vdash B(x)\ \mathrm{type}}{\Gamma \vdash \prod_{(x:A)} B(x)\ \mathrm{type}}\ \Pi.$$

This rule simply states that in order to form the type $\prod_{(x:A)} B(x)$ in context Γ, we must have a type family B over A in context Γ.

We also require that the the operation of forming dependent function types respects judgmental equality. This is postulated in the **congruence rule** for Π-types:

$$\frac{\Gamma \vdash A \doteq A'\ \mathrm{type} \qquad \Gamma, x : A \vdash B(x) \doteq B'(x)\ \mathrm{type}}{\Gamma \vdash \prod_{(x:A)} B(x) \doteq \prod_{(x:A')} B'(x)\ \mathrm{type}}\ \Pi\text{-eq}.$$

The Π-introduction rule

The introduction rule for dependent functions tells us how we may construct dependent functions of type $\prod_{(x:A)} B(x)$. The idea is that a dependent function $f : \prod_{(x:A)} B(x)$ is an operation that takes an $x : A$ to $f(x) : B(x)$. Hence the introduction rule of dependent functions postulates that, in order to construct a dependent function one has to construct a term $b(x) : B(x)$ indexed by $x : A$ in context Γ, i.e.:

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

This introduction rule for dependent functions is also called the **λ-abstraction rule**, and we also say that the λ-abstraction $\lambda x.\, b(x)$ **binds** the variable x in b. Just like ordinary mathematicians, we will sometimes write $x \mapsto b(x)$ for a function $\lambda x.\, b(x)$. The map $n \mapsto n^2$ is an example.

We will also require that λ-abstraction respects judgmental equality. Therefore we postulate the **congruence rule** for λ-abstraction, which asserts that

$$\frac{\Gamma, x : A \vdash b(x) \doteq b'(x) : B(x)}{\Gamma \vdash \lambda x.\, b(x) \doteq \lambda x.\, b'(x) : \prod_{(x:A)} B(x)}\ \lambda\text{-eq}.$$

The Π-elimination rule

The elimination rule for dependent function types provides us with a way to *use* dependent functions. The way to use a dependent function is to evaluate it at an argument of the domain type. The Π-elimination rule is therefore also called the **evaluation rule**:

$$\frac{\Gamma \vdash f : \prod_{(x:A)} B(x)}{\Gamma, x : A \vdash f(x) : B(x)}\ ev.$$

This rule asserts that given a dependent function $f : \prod_{(x:A)} B(x)$ in context Γ we obtain a term $f(x)$ of type $B(x)$ indexed by $x : A$ in context Γ. Again we require that evaluation respects judgmental equality:

$$\frac{\Gamma \vdash f \doteq f' : \prod_{(x:A)} B(x)}{\Gamma, x : A \vdash f(x) \doteq f'(x) : B(x)}\ ev\text{-eq}.$$

The Π-computation rules

We now postulate rules that specify the behavior of functions. First, we have a rule that asserts that a function of the form $\lambda x.\, b(x)$ behaves as expected: when we evaluate it at $x : A$, then we obtain the value $b(x) : B(x)$. This rule is called the **β-rule**

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

Second, we postulate a rule that asserts that all elements of a Π-type are (dependent) functions. This rule is known as the **η-rule**

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

In other words, the computation rules (β and η) for dependent function types postulate that λ-abstraction rule and the evaluation rule are mutual inverses. This completes the specification of dependent function types.

2.2 Ordinary function types

An important special case of Π-types arises when both A and B are types in context Γ. In this case, we can first weaken B by A and then apply the Π-formation rule to obtain the type $A \to B$ of *ordinary* functions from A to B, as in the following derivation:

$$\frac{\dfrac{\Gamma \vdash A\ \mathrm{type} \qquad \Gamma \vdash B\ \mathrm{type}}{\Gamma, x : A \vdash B\ \mathrm{type}}\ W}{\Gamma \vdash \prod_{(x:A)} B\ \mathrm{type}.}\ \Pi$$

A term $f : \prod_{(x:A)} B$ is a function that takes an argument $x : A$ and returns $f(x) : B$. In other words, terms of type $\prod_{(x:A)} B$ are indeed ordinary functions from A to B. Therefore, we define the type $A \to B$ of **(ordinary) functions** from A to B by

$$A \to B :\equiv \prod_{(x:A)} B.$$

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

If $f : A \to B$ is a function, then the type A is also called the **domain** of f, and the type B is also called the **codomain** of f.

Sometimes we will also write B^A for the type $A \to B$. Formally, we make such definitions by adding one more line to the above derivation:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma \vdash B\ \mathsf{type}}{\Gamma, x : A \vdash B\ \mathsf{type}}\, W}{\Gamma \vdash \prod_{(x:A)} B\ \mathsf{type}}\, \Pi}{\Gamma \vdash A \to B :\equiv \prod_{(x:A)} B\ \mathsf{type}.}$$

Remark 2.2.1 More generally, we can make definitions at the end of a derivation if the conclusion is a certain type in context, or if the conclusion is a certain term of a type in context. Suppose, for instance, that we have a derivation

$$\dfrac{\mathcal{D}}{\Gamma \vdash a : A,}$$

in which the derivation $\mathcal{D}$ makes use of the premises $\mathcal{H}_1, \ldots, \mathcal{H}_n$. If we wish to make a definition $\mathsf{c} :\equiv a$, then we can extend the derivation tree with

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash a : A}}{\Gamma \vdash \mathsf{c} :\equiv a : A.}$$

The effect of such a definition is that we have extended our type theory with a new constant c, for which the following inference rules are valid

$$\dfrac{\mathcal{H}_1 \quad \mathcal{H}_2 \quad \ldots \quad \mathcal{H}_n}{\Gamma \vdash \mathsf{c} : A} \qquad \dfrac{\mathcal{H}_1 \quad \mathcal{H}_2 \quad \ldots \quad \mathcal{H}_n}{\Gamma \vdash \mathsf{c} \doteq a : A.}$$

In our example of the definition of the ordinary function type $A \to B$, we therefore have by definition the following valid inference rules

$$\dfrac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma \vdash B\ \mathsf{type}}{\Gamma \vdash A \to B\ \mathsf{type}} \qquad \dfrac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma \vdash B\ \mathsf{type}}{\Gamma \vdash A \to B \doteq \prod_{(x:A)} B\ \mathsf{type}.}$$

There are of course many such definitions throughout the development of dependent type theory, the univalent foundations for mathematics, and synthetic homotopy theory. They are all included in the index at the end of this book.

Remark 2.2.2 By the term conversion rules of Exercise 1.1 we can now use the rules for λ-abstraction, evaluation, and so on, to obtain corresponding rules for the ordinary function type $A \to B$. We give a brief summary of these rules, omitting the congruence rules.

$$\dfrac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma \vdash B\ \mathsf{type}}{\Gamma \vdash A \to B\ \mathsf{type}}\, \to$$

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

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

Now we can use these rules to construct some familiar functions, such as the identity function $\mathsf{id} : A \to A$ on an arbitrary type A, and the composition $g \circ f : A \to C$ of any two functions $f : A \to B$ and $g : B \to C$.

Definition 2.2.3 For any type A in context Γ, we define the **identity function** $\mathsf{id}_A : A \to A$ using the generic term:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A\ \mathrm{type}}{\Gamma, x : A \vdash x : A}}{\Gamma \vdash \lambda x.\, x : A \to A}}{\Gamma \vdash \mathsf{id}_A :\equiv \lambda x.\, x : A \to A.}$$

The identity function therefore satisfies the following inference rules:

$$\frac{\Gamma \vdash A\ \mathrm{type}}{\Gamma \vdash \mathsf{id}_A : A \to A} \qquad \frac{\Gamma \vdash A\ \mathrm{type}}{\Gamma \vdash \mathsf{id}_A \doteq \lambda x.\, x : A \to A.}$$

Next, we define the composition of functions. We will introduce the composition operation itself as a function comp that takes two arguments: the first argument is a function $g : B \to C$, and the second argument is a function $f : A \to B$. The output is a function $\mathsf{comp}(g, f) : A \to C$, for which we often write $g \circ f$.

Remark 2.2.4 Since composition is a function that takes multiple arguments, we need to know how to represent such functions. Types of functions with multiple arguments can be formed by iterating the Π-formation rule or the $\to$-formation rule. For example, a function

$$f : A \to (B \to C)$$

takes two arguments: first it takes an argument $x : A$, and the output $f(x)$ has type $B \to C$. This is again a function type, so $f(x)$ is a function that takes an argument $y : B$, and its output $f(x)(y)$ has type C. We will usually write $f(x, y)$ for $f(x)(y)$.

Similarly, when $C(x, y)$ is a family of types indexed by $x : A$ and $y : B(x)$, then we can form the dependent function type $\prod_{(x:A)} \prod_{(y:B(x))} C(x, y)$. In the special case where $C(x, y)$ is a family of types indexed by two elements $x, y : A$ of the same type, then we often write

$$\prod_{(x,y:A)} C(x, y)$$

for the type $\prod_{(x:A)} \prod_{(y:A)} C(x, y)$.

With the idea of iterating function types, we see that type of the composition operation comp is

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

It is the type of functions, taking a function $g : B \to C$, to the type of functions $(A \to B) \to (A \to C)$. Thus, $\mathsf{comp}(g)$ is again a function, mapping a function $f : A \to B$ to a function of type $A \to C$.

Definition 2.2.5 For any three types A, B, and C in context Γ, there is a **composition** operation

$$\mathsf{comp} : (B \to C) \to ((A \to B) \to (A \to C)).$$

We will usually write $g \circ f$ for $\mathsf{comp}(g, f)$.

Construction The idea of the definition is to define $\mathsf{comp}(g, f)$ to be the function $\lambda x.\, g(f(x))$. The function comp is therefore defined as

$$\mathsf{comp} \coloneqq \lambda g.\, \lambda f.\, \lambda x.\, g(f(x)).$$

The derivation we use to construct comp is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma \vdash B\ \mathsf{type}}{\Gamma, f : B^A, x : A \vdash f(x) : B}\,(\mathrm{a})}{\Gamma, g : C^B, f : B^A, x : A \vdash f(x) : B} \quad \dfrac{\dfrac{\Gamma \vdash B\ \mathsf{type} \quad \Gamma \vdash C\ \mathsf{type}}{\Gamma, g : C^B, y : B \vdash g(y) : C}\,(\mathrm{b})}{\Gamma, g : C^B, f : B^A, y : B \vdash g(y) : C}}{\Gamma, g : C^B, f : B^A, x : A, y : B \vdash g(y) : C}}{\Gamma, g : C^B, f : B^A, x : A \vdash g(f(x)) : C}}{\Gamma, g : C^B, f : B^A \vdash \lambda x.\, g(f(x)) : C^A}}{\Gamma, g : B \to C \vdash \lambda f.\, \lambda x.\, g(f(x)) : B^A \to C^A}}{\Gamma \vdash \lambda g.\, \lambda f.\, \lambda x.\, g(f(x)) : C^B \to (B^A \to C^A)}}{\Gamma \vdash \mathsf{comp} \coloneqq \lambda g.\, \lambda f.\, \lambda x.\, g(f(x)) : C^B \to (B^A \to C^A).}$$

Note, however, that we haven't derived the rules (a) and (b) yet. These rules assert that the *generic functions* of $A \to B$ and $B \to C$ can also be evaluated. The formal derivation of this fact is as follows:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma \vdash B\ \mathsf{type}}{\Gamma \vdash A \to B\ \mathsf{type}}}{\Gamma, f : A \to B \vdash f : A \to B}}{\Gamma, f : A \to B, x : A \vdash f(x) : B.}$$

This completes the construction of comp. □

In the remainder of this section we will see how to use the given rules for function types to derive the laws of a category for functions. These are the laws that assert that function composition is associative and that the identity function satisfies the unit laws.

Lemma 2.2.6 *Composition of functions is associative, i.e., we can derive*

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash g : B \to C \qquad \Gamma \vdash h : C \to D}{\Gamma \vdash (h \circ g) \circ f \doteq h \circ (g \circ f) : A \to D.}$$

Proof The main idea of the proof is that both $((h \circ g) \circ f)(x)$ and $(h \circ (g \circ f))(x)$ evaluate to $h(g(f(x)))$, and therefore $(h \circ g) \circ f$ and $h \circ (g \circ f)$ must be judgmentally equal. This idea is made formal in the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash f : A \to B}{\Gamma, x : A \vdash f(x) : B} \quad \dfrac{\Gamma \vdash g : B \to C}{\Gamma, y : B \vdash g(y) : C}}{\Gamma, x : A \vdash g(f(x)) : C} \quad \dfrac{\Gamma \vdash h : C \to D}{\Gamma, z : C \vdash h(z) : D}}{\Gamma, x : A, z : C \vdash h(z) : D}}{\Gamma, x : A \vdash h(g(f(x))) : D}}{\Gamma, x : A \vdash h(g(f(x))) \doteq h(g(f(x))) : D}}{\Gamma, x : A \vdash (h \circ g)(f(x)) \doteq h((g \circ f)(x)) : D}}{\Gamma, x : A \vdash ((h \circ g) \circ f)(x) \doteq (h \circ (g \circ f))(x) : D}}{\Gamma \vdash (h \circ g) \circ f \doteq h \circ (g \circ f) : A \to D.}$$

□

Lemma 2.2.7 *Composition of functions satisfies the left and right unit laws, i.e., we can derive*

$$\frac{\Gamma \vdash f : A \to B}{\Gamma \vdash \mathsf{id}_B \circ f \doteq f : A \to B}$$

and

$$\frac{\Gamma \vdash f : A \to B}{\Gamma \vdash f \circ \mathsf{id}_A \doteq f : A \to B.}$$

Proof Note that it suffices to derive that $\mathsf{id}(f(x)) \doteq f(x)$ in context $\Gamma, x : A$, because once we derived this equality we can finish the derivation with

$$\frac{\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash \mathsf{id}(f(x)) \doteq f(x) : B}}{\Gamma \vdash \lambda x.\, \mathsf{id}(f(x)) \doteq \lambda x.\, f(x) : A \to B} \qquad \frac{\Gamma \vdash f : A \to B}{\Gamma \vdash \lambda x.\, f(x) \doteq f : A \to B}}{\Gamma \vdash \mathsf{id} \circ f \doteq f : A \to B.}$$

The derivation of the equality $\mathsf{id}(f(x)) \doteq f(x)$ in context $\Gamma, x : A$ is as follows:

$$\dfrac{\dfrac{\Gamma \vdash f : A \to B}{\Gamma, x : A \vdash f(x) : B} \qquad \dfrac{\Gamma \vdash A \text{ type} \qquad \dfrac{\Gamma \vdash B \text{ type}}{\Gamma, y : B \vdash \mathsf{id}(y) \doteq y : B}}{\Gamma, x : A, y : B \vdash \mathsf{id}(y) \doteq y : B}}{\Gamma, x : A \vdash \mathsf{id}(f(x)) \doteq f(x) : B.}$$

We leave the right unit law as Exercise 2.2. □

Exercises

2.1 The η-rule is often seen as a judgmental extensionality principle. Use the η-rule to show that if f and g take equal values, then they must be equal, i.e., give a derivation for the rule

$$\dfrac{\begin{array}{l}\Gamma \vdash f : \prod_{(x:A)} B(x) \\ \Gamma \vdash g : \prod_{(x:A)} B(x) \\ \Gamma, x : A \vdash f(x) \doteq g(x) : B(x)\end{array}}{\Gamma \vdash f \doteq g : \prod_{(x:A)} B(x).}$$

2.2 Give a derivation for the right unit law of Lemma 2.2.7.

2.3 (a) Construct the **constant function**

$$\dfrac{\Gamma \vdash A \text{ type}}{\Gamma, y : B \vdash \mathsf{const}_y : A \to B.}$$

(b) Show that

$$\dfrac{\Gamma \vdash f : A \to B}{\Gamma, z : C \vdash \mathsf{const}_z \circ f \doteq \mathsf{const}_z : A \to C.}$$

(c) Show that

$$\dfrac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash g : B \to C}{\Gamma, y : B \vdash g \circ \mathsf{const}_y \doteq \mathsf{const}_{g(y)} : A \to C.}$$

2.4 (a) Define the **swap function**

$$\dfrac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type} \qquad \Gamma, x : A, y : B \vdash C(x,y) \text{ type}}{\Gamma \vdash \sigma : \left(\prod_{(x:A)} \prod_{(y:B)} C(x,y)\right) \to \left(\prod_{(y:B)} \prod_{(x:A)} C(x,y)\right)}$$

that swaps the order of the arguments.

(b) Show that

$$\dfrac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type} \qquad \Gamma, x : A, y : B \vdash C(x,y) \text{ type}}{\Gamma \vdash \sigma \circ \sigma \doteq \mathsf{id} : \left(\prod_{(x:A)} \prod_{(y:B)} C(x,y)\right) \to \left(\prod_{(x:A)} \prod_{(y:B)} C(x,y)\right).}$$

3 The natural numbers

The set of natural numbers is the most important object in mathematics. We quote Bishop, from his Constructivist Manifesto, the first chapter in Foundations of Constructive Analysis [2], where he gives a colorful illustration of its importance to mathematics.

"The primary concern of mathematics is number, and this means the positive integers. We feel about number the way Kant felt about space. The positive integers and their arithmetic are presupposed by the very nature of our intelligence and, we are tempted to believe, by the very nature of intelligence in general. The development of the theory of the positive integers from the primitive concept of the unit, the concept of adjoining a unit, and the process of mathematical induction carries complete conviction. In the words of Kronecker, the positive integers were created by God. Kronecker would have expressed it even better if he had said that the positive integers were created by God for the benefit of man (and other finite beings). Mathematics belongs to man, not to God. We are not interested in properties of the positive integers that have no descriptive meaning for finite man. When a man proves a positive integer to exist, he should show how to find it. If God has mathematics of his own that needs to be done, let him do it himself."

A bit later in the same chapter, he continues:

"Building on the positive integers, weaving a web of ever more sets and ever more functions, we get the basic structures of mathematics: the rational number system, the real number system, the euclidean spaces, the complex number system, the algebraic number fields, Hilbert space, the classical groups, and so forth. Within the framework of these structures, most mathematics is done. Everything attaches itself to number, and every mathematical statement ultimately expresses the fact that if we perform certain computations within the set of positive integers, we shall get certain results."

3.1 The formal specification of the type of natural numbers

The type $\mathbb{N}$ of **natural numbers** is the archetypal example of an inductive type. The rules we postulate for the type of natural numbers come in four sets, just as the rules for Π-types:

(i) The formation rule, which asserts that the type $\mathbb{N}$ can be formed.
(ii) The introduction rules, which provide the zero element $0_{\mathbb{N}}$ and the successor function $\mathsf{succ}_{\mathbb{N}}$.
(iii) The elimination rule. This rule is the type theoretic version of the induction principle for $\mathbb{N}$.
(iv) The computation rules, which assert that any application of the elimination rule behaves as expected on the constructors $0_{\mathbb{N}}$ and $\mathsf{succ}_{\mathbb{N}}$ of $\mathbb{N}$.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The formation rule of $\mathbb{N}$

The type $\mathbb{N}$ is formed by the **$\mathbb{N}$-formation** rule

$$\frac{}{\vdash \mathbb{N}\ \mathrm{type}}\ \mathbb{N}\text{-form.}$$

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

The introduction rules of $\mathbb{N}$

Unlike the set of positive integers in Bishop's remarks, Peano's first axiom postulates that 0 is a natural number. The introduction rules for $\mathbb{N}$ equip it with the **zero element** and the **successor function**.

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

Remark 3.1.1 Every element in type theory always comes equipped with its type. Therefore it is possible in type theory that all elements have a *unique* type. In general, it is therefore good practice to make sure that every element is given a unique name, and in formalized mathematics in computer proof assistants this is even required. For example, the element $0_{\mathbb{N}}$ has type $\mathbb{N}$, and it is not also a type of $\mathbb{Z}$. This is why we annotate the terms $0_{\mathbb{N}}$ and $\mathsf{succ}_{\mathbb{N}}$ with their type in the subscript. The type $\mathbb{Z}$ of the integers will be introduced in the next section, which will come equipped with a zero element $0_{\mathbb{Z}}$ and a successor function $\mathsf{succ}_{\mathbb{Z}}$.

The induction principle of $\mathbb{N}$

The classical induction principle of the natural numbers tells us what we have to do in order to show that $\forall_{(n\in\mathbb{N})} P(n)$ holds, for a predicate P over $\mathbb{N}$. Recall that a predicate P on a set X is just a proposition $P(x)$ about an arbitrary $x \in X$. For example, the assertion that 'n is divisible by five' is a predicate on the natural numbers.

In dependent type theory we may think of a type family P over $\mathbb{N}$ as a predicate over $\mathbb{N}$. The type theoretical induction principle of $\mathbb{N}$ is therefore formulated using a type family P over $\mathbb{N}$:

$$\frac{\begin{array}{l}\Gamma, n : \mathbb{N} \vdash P(n)\ \mathrm{type} \\ \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \\ \Gamma \vdash p_S : \prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_{\mathbb{N}}(n))\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(p_0, p_S) : \prod_{(n:\mathbb{N})} P(n)}\ \mathbb{N}\text{-ind.}$$

In other words, the type theoretical induction principle of $\mathbb{N}$ tells us what we need to do in order to construct a dependent function $\prod_{(n:\mathbb{N})} P(n)$. Just as in the classical induction principle, there are two things to be constructed given a type family P over $\mathbb{N}$: in the **base case** we need to construct an element $p_0 : P(0_\mathbb{N})$, and for the **inductive step** we need to construct a function of type $P(n) \to P(\mathsf{succ}_\mathbb{N}(n))$ for all $n : \mathbb{N}$.

Remark 3.1.2 We might alternatively present the induction principle of $\mathbb{N}$ as the following inference rule

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \mathrm{type}}{\Gamma \vdash \mathsf{ind}_\mathbb{N} : P(0_\mathbb{N}) \to \left(\left(\prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_\mathbb{N}(n))\right) \to \prod_{(n:\mathbb{N})} P(n)\right).}$$

In other words, for any type family P over $\mathbb{N}$ there is a *function* $\mathsf{ind}_\mathbb{N}$ that takes two arguments, one for the base case and one for the inductive step, and returns a section of P. We claim that this rule is *interderivable* with the rule $\mathbb{N}$-ind above.

To see that indeed we get such a function from the rule $\mathbb{N}$-ind, we use generic elements. First, we let Γ' be the context

$$\Gamma,\ p_0 : P(0_\mathbb{N}),\ p_S : \prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_\mathbb{N}(n)).$$

By weakening we obtain that

$$\begin{aligned}
&\Gamma',\ n : \mathbb{N} \vdash P(n)\ \mathrm{type}\\
&\Gamma' \vdash p_0 : P(0_\mathbb{N})\\
&\Gamma' \vdash p_S : \prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_\mathbb{N}(n)).
\end{aligned}$$

Therefore, the induction principle of $\mathbb{N}$ provides us with a dependent function

$$\Gamma' \vdash \mathsf{ind}_\mathbb{N}(p_0, p_S) : \prod_{(n:\mathbb{N})} P(n).$$

Now we proceed by λ-abstraction twice to obtain a function

$$\mathsf{ind}_\mathbb{N} : P(0_\mathbb{N}) \to \left(\left(\prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_\mathbb{N}(n))\right) \to \prod_{(n:\mathbb{N})} P(n)\right)$$

in the original context Γ. This shows that we can define the function $\mathsf{ind}_\mathbb{N}$ from the rule $\mathbb{N}$-ind. Conversely, we can derive the rule $\mathbb{N}$-ind from the rule that presents $\mathsf{ind}_\mathbb{N}$ as a function. We conclude that the "official" rule $\mathbb{N}$-ind and the rule that presents $\mathsf{ind}_\mathbb{N}$ as a function are indeed interderivable.

The computation rules of $\mathbb{N}$

The computation rules for $\mathbb{N}$ postulate that the dependent function

$$\mathsf{ind}_{\mathbb{N}}(p_0, p_S) : \prod_{(n:\mathbb{N})} P(n)$$

behaves as expected when it is applied to $0_{\mathbb{N}}$ or a successor. There is one computation rule for each step in the induction principle, covering the base case and the inductive step.

The computation rule for the base case is

$$\frac{\begin{array}{l}\Gamma, n : \mathbb{N} \vdash P(n)\ \text{type} \\ \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \\ \Gamma \vdash p_S : \prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_{\mathbb{N}}(n))\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(p_0, p_S, 0_{\mathbb{N}}) \doteq p_0 : P(0_{\mathbb{N}}).}$$

The computation rule for the inductive step has the same premises as the computation rule for the base case:

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

This completes the formal specification of the type $\mathbb{N}$ of natural numbers.

3.2 Addition on the natural numbers

The type theoretic induction principle of $\mathbb{N}$ can be used to do all the usual constructions of operations on $\mathbb{N}$, and to derive all the familiar properties about natural numbers. Many of those properties, however, require a few more ingredients of Martin-Löf's dependent type theory. For example, the traditional inductive proof that the triangular numbers can be calculated by

$$1 + \cdots + n = \frac{n(n+1)}{2}$$

is analogous in type theory, but it requires the identity type to state this equation. We will introduce the identity type in Section 5. Until we have fully specified all the ways of forming types in Martin-Löf's dependent type theory, we are a bit limited in what we can do with the natural numbers, but at the present stage we can define some of the familiar operations on $\mathbb{N}$. We give in this section the type theoretical construction the **addition operation** by induction on $\mathbb{N}$, along with the complete derivation tree.

Definition 3.2.1 We define a function

$$\mathsf{add}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$$

satisfying the specification

$$\begin{aligned}\mathsf{add}_{\mathbb{N}}(m, 0_{\mathbb{N}}) &\doteq m\\ \mathsf{add}_{\mathbb{N}}(m, \mathsf{succ}_{\mathbb{N}}(n)) &\doteq \mathsf{succ}_{\mathbb{N}}(\mathsf{add}_{\mathbb{N}}(m, n)).\end{aligned}$$

Usually we will write $m + n$ for $\mathsf{add}_{\mathbb{N}}(m, n)$.

Construction. We will construct the binary operation $\mathsf{add}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ by induction on the second variable. In other words, we will construct an element

$$m : \mathbb{N} \vdash \mathsf{add}_{\mathbb{N}}(m) : \mathbb{N} \to \mathbb{N}.$$

The context Γ we work in is therefore $m : \mathbb{N}$. The induction principle of $\mathbb{N}$ is used with the family of types $P(n) :\equiv \mathbb{N}$ indexed by $n : \mathbb{N}$ in context $m : \mathbb{N}$. Therefore we need to construct

$$\begin{aligned}&m : \mathbb{N} \vdash \mathsf{add\text{-}zero}_{\mathbb{N}}(m) : \mathbb{N}\\ &m : \mathbb{N} \vdash \mathsf{add\text{-}succ}_{\mathbb{N}}(m) : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}),\end{aligned}$$

in order to obtain

$$m : \mathbb{N} \vdash \mathsf{add}_{\mathbb{N}}(m) :\equiv \mathsf{ind}_{\mathbb{N}}(\mathsf{add\text{-}zero}_{\mathbb{N}}(m), \mathsf{add\text{-}succ}_{\mathbb{N}}(m)) : \mathbb{N} \to \mathbb{N}.$$

The element $\mathsf{add\text{-}zero}_{\mathbb{N}}(m) : \mathbb{N}$ in context $m : \mathbb{N}$ is of course defined to be $m : \mathbb{N}$, i.e., by the generic element, because adding zero should just be the identity function. To see how the function $\mathsf{add\text{-}succ}_{\mathbb{N}}(m) : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ should be defined, we look at the specification of $\mathsf{add}_{\mathbb{N}}(m)$ when it is applied to a successor:

$$\mathsf{add}_{\mathbb{N}}(m, \mathsf{succ}_{\mathbb{N}}(n)) \doteq \mathsf{succ}_{\mathbb{N}}(\mathsf{add}_{\mathbb{N}}(m, n)).$$

This shows us that we should define

$$\mathsf{add\text{-}succ}_{\mathbb{N}}(m, n, x) \doteq \mathsf{succ}_{\mathbb{N}}(x),$$

because with this definition we will have

$$\begin{aligned}\mathsf{add}_{\mathbb{N}}(m, \mathsf{succ}_{\mathbb{N}}(n)) &\doteq \mathsf{ind}_{\mathbb{N}}(\mathsf{add\text{-}zero}_{\mathbb{N}}(m), \mathsf{add\text{-}succ}_{\mathbb{N}}(m), \mathsf{succ}_{\mathbb{N}}(n))\\ &\doteq \mathsf{add\text{-}succ}_{\mathbb{N}}(m, n, \mathsf{add}_{\mathbb{N}}(m, n))\\ &\doteq \mathsf{succ}_{\mathbb{N}}(\mathsf{add}_{\mathbb{N}}(m, n)).\end{aligned}$$

The formal derivation for the construction of $\mathsf{add\text{-}succ}_{\mathbb{N}}$ is as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathbb{N}\ \mathsf{type}} \quad \dfrac{}{\vdash \mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}}{n : \mathbb{N} \vdash \mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}}{m : \mathbb{N}, n : \mathbb{N} \vdash \mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}}{m : \mathbb{N} \vdash \lambda n.\, \mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}}{m : \mathbb{N} \vdash \mathsf{add\text{-}succ}_{\mathbb{N}}(m) :\equiv \lambda n.\, \mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}).}$$

We combine this derivation with the induction principle of $\mathbb{N}$ to complete the construction of addition:

$$\dfrac{\dfrac{\dfrac{\vdots}{m : \mathbb{N} \vdash \mathsf{add\text{-}zero}_{\mathbb{N}}(m) :\equiv m : \mathbb{N}} \quad \dfrac{\vdots}{m : \mathbb{N} \vdash \mathsf{add\text{-}succ}_{\mathbb{N}}(m) : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}}{m : \mathbb{N} \vdash \mathsf{ind}_{\mathbb{N}}(\mathsf{add\text{-}zero}_{\mathbb{N}}(m), \mathsf{add\text{-}succ}_{\mathbb{N}}(m)) : \mathbb{N} \to \mathbb{N}}}{m : \mathbb{N} \vdash \mathsf{add}_{\mathbb{N}}(m) :\equiv \mathsf{ind}_{\mathbb{N}}(\mathsf{add\text{-}zero}_{\mathbb{N}}(m), \mathsf{add\text{-}succ}_{\mathbb{N}}(m)) : \mathbb{N} \to \mathbb{N}.}$$

The asserted judgmental equalities then hold by the computation rules for $\mathbb{N}$. □

Remark 3.2.2 By the computation rules for $\mathbb{N}$ it follows that

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

A simple consequence of this definition is that $\mathsf{succ}_{\mathbb{N}}(n) \doteq n + 1$, as one would expect. However, the rules that we provided so far are not sufficient to also conclude that $0_{\mathbb{N}} + n \doteq n$ and $\mathsf{succ}_{\mathbb{N}}(m) + n \doteq \mathsf{succ}_{\mathbb{N}}(m + n)$. In fact, dependent type theory with its inductive types does not provide any means to prove such judgmental equalities.

Nevertheless, once we have introduced the *identity type* in Section 5 we will be able to *identify* $0_{\mathbb{N}} + n$ with n, and $\mathsf{succ}_{\mathbb{N}}(m) + n$ with $\mathsf{succ}_{\mathbb{N}}(m + n)$. See Propositions 5.6.1 and 5.6.2.

3.3 Pattern matching

Note that in definition Definition 3.2.1 we stated that $\mathsf{add}_{\mathbb{N}}$ is a function of type $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ satisfying the specification

$$\begin{aligned} \mathsf{add}_{\mathbb{N}}(m, 0_{\mathbb{N}}) &\doteq m \\ \mathsf{add}_{\mathbb{N}}(m, \mathsf{succ}_{\mathbb{N}}(n)) &\doteq \mathsf{succ}_{\mathbb{N}}(\mathsf{add}_{\mathbb{N}}(m, n)). \end{aligned}$$

Such a specification is enough to characterize the function $\mathsf{add}_{\mathbb{N}}(m)$ entirely, because it postulates the behaviour of $\mathsf{add}_{\mathbb{N}}(m)$ at the constructors of $\mathbb{N}$. It is therefore convenient to present the definition of $\mathsf{add}_{\mathbb{N}}$ recursively in the following way:

$$\begin{aligned} \mathsf{add}_{\mathbb{N}}(m, 0_{\mathbb{N}}) &:\equiv m \\ \mathsf{add}_{\mathbb{N}}(m, \mathsf{succ}_{\mathbb{N}}(n)) &:\equiv \mathsf{succ}_{\mathbb{N}}(\mathsf{add}_{\mathbb{N}}(m, n)). \end{aligned}$$

More generally, if we want to define a dependent function $f : \prod_{(n:\mathbb{N})} P(n)$ by induction on n, using

$$p_0 : P(0_{\mathbb{N}})$$

$$p_S : \prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_{\mathbb{N}}(n)),$$

we can present that definition by writing

$$\begin{aligned} f(0_{\mathbb{N}}) &:\equiv p_0 \\ f(\mathsf{succ}_{\mathbb{N}}(n)) &:\equiv p_S(n, f(n)). \end{aligned}$$

When the definition of f is presented in this way, we say that f is defined by **pattern matching** on the variable n. To see that f is fully specified when it is defined by pattern matching, we have to recover the dependent function

$$p_S : \prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_{\mathbb{N}}(n))$$

from the expression $p_S(n, f(n))$ that was used in the definition of f. This can of course be done by replacing all occurrences of the term $f(n)$ in the expression $p_S(n, f(n))$ with a fresh variable $x : P(n)$. In other words, when a subexpression of $p_S(n, f(n))$ *matches* $f(n)$, we replace that subexpression by x. This is where the name *pattern matching* comes from. Many computer proof assistants have the pattern matching mechanism built in, because it is a concise way of presenting a recursive definition. Another advantage of presenting definitions by pattern matching is that the judgmental equalities by which the object is defined are immediately displayed. Those judgmental equalities are all that is known about the defined object, and often proving things about it amounts to finding a way to apply those judgmental equalities.

Pattern matching can also be used in more complicated situations, such as defining a function by pattern matching on multiple variables, or by iterated pattern matching. For example, an alternative definition of addition on $\mathbb{N}$ could be given by pattern matching on both variables:

$$\begin{aligned} \mathsf{add}'_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) &:\equiv 0_{\mathbb{N}} \\ \mathsf{add}'_{\mathbb{N}}(0_{\mathbb{N}}, \mathsf{succ}_{\mathbb{N}}(n)) &:\equiv \mathsf{succ}_{\mathbb{N}}(n) \\ \mathsf{add}'_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m), 0_{\mathbb{N}}) &:\equiv \mathsf{succ}_{\mathbb{N}}(m) \\ \mathsf{add}'_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m), \mathsf{succ}_{\mathbb{N}}(n)) &:\equiv \mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(\mathsf{add}'_{\mathbb{N}}(m, n)). \end{aligned}$$

An example of a definition by iterated pattern matching is the **Fibonacci function** $F : \mathbb{N} \to \mathbb{N}$. This function is defined by

$$\begin{aligned} F(0_{\mathbb{N}}) &:\equiv 0_{\mathbb{N}} \\ F(1_{\mathbb{N}}) &:\equiv 1_{\mathbb{N}} \\ F(\mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n))) &:\equiv F(\mathsf{succ}_{\mathbb{N}}(n)) + F(n). \end{aligned}$$

However, since $F(\mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n)))$ is defined using both $F(\mathsf{succ}_{\mathbb{N}}(n))$ and $F(n)$, it is not immediately clear how to present F by the usual induction

principle of $\mathbb{N}$. It is a nice puzzle, which we leave as Exercise 3.5, to find a definition of the Fibonacci sequence with the usual induction principle of $\mathbb{N}$.

Exercises

3.1 (a) Define the **multiplication** operation

$$\mathsf{mul}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}).$$

(b) Define the **exponentiation function** $n, m \mapsto m^n$ of type $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$.

3.2 Define the binary **min** and **max** functions

$$\mathsf{min}_{\mathbb{N}}, \mathsf{max}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}).$$

3.3 (a) Define the **triangular numbers**

$$1 + \cdots + n.$$

(b) Define the **factorial** operation $n \mapsto n!$.

3.4 Define the **binomial coefficient** $\binom{n}{k}$ for any $n, k : \mathbb{N}$, making sure that $\binom{n}{k} \doteq 0$ when $n < k$.

3.5 Use the induction principle of $\mathbb{N}$ to define the **Fibonacci sequence** as a function $F : \mathbb{N} \to \mathbb{N}$ that satisfies the equations

$$F(0_{\mathbb{N}}) \doteq 0_{\mathbb{N}}$$
$$F(1_{\mathbb{N}}) \doteq 1_{\mathbb{N}}$$
$$F(\mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n))) \doteq F(\mathsf{succ}_{\mathbb{N}}(n)) + F(n).$$

3.6 Define division by two rounded down as a function $\mathbb{N} \to \mathbb{N}$ in two ways: first by pattern matching, and then directly by the induction principle of $\mathbb{N}$.

4 More inductive types

The induction principle for the natural numbers simply tells us what we have to do in order to construct a dependent function

$$\prod_{(n:\mathbb{N})} P(n).$$

Moreover, the induction principle completely determines the type of natural numbers. This idea can be applied much more broadly. Many types can be specified by stating: (1) how to form their elements, and (2) how to construct

dependent functions out of them. Thus we arrive at the idea of more general inductive types.

In this section we introduce the unit type, the empty type, coproducts, dependent pair types, and cartesian products. All of these types are introduced as inductive types, by stating what their constructors are and what their induction principles are.

From this section on, we will also start using a more informal style. The inductive types will be specified by a description of their constructors and induction principles in terms of operations on dependent function types, which is more tightly connected with how we will use them, but we will not display the formal rules. It is a good exercise for the reader to formally specify at least some of the inductive types of this section by stating their formal rules.

4.1 The idea of general inductive types

Just like the type of natural numbers, other inductive types are also specified by their *constructors*, an *induction principle*, and their *computation rules*:

(i) The constructors tell what structure the inductive type comes equipped with. There may be any finite number of constructors, even no constructors at all, in the specification of an inductive type.
(ii) The induction principle specifies the data that should be provided in order to construct a section of an arbitrary type family over the inductive type. The idea of the induction principle is always the same: in order to define a dependent function $f : \prod_{(x:A)} B(x)$, one has to specify the behaviour of f at the constructors of A.
(iii) The computation rules assert that the inductively defined section agrees on the constructors with the data that was used to define the section. Thus, there is a computation rule for every constructor.

Since any inductively defined function is entirely determined by its behavior on the constructors, we can again present such inductive definitions by pattern matching. Therefore, we will also specify for each inductive type how to give definitions by pattern matching.

4.2 The unit type

A straightforward example of an inductive type is the *unit type*, which has just one constructor. Its induction principle is analogous to just the base case of induction on the natural numbers.

Definition 4.2.1 We define the **unit type** to be a type $\mathbf{1}$ equipped with a term

$$\star : \mathbf{1},$$

satisfying the induction principle that for any family of types $P(x)$ indexed by $x : \mathbf{1}$, there is a function

$$\mathsf{ind}_{\mathbf{1}} : P(\star) \to \textstyle\prod_{(x:\mathbf{1})} P(x)$$

for which the computation rule

$$\mathsf{ind}_{\mathbf{1}}(p, \star) \doteq p$$

holds. Alternatively, a definition of a dependent function $f : \prod_{(x:\mathbf{1})} P(x)$ by induction using $p : P(\star)$ can be presented by pattern matching as

$$f(\star) :\equiv p.$$

A special case of the induction principle arises when P does not actually depend on $\mathbf{1}$. If we are given a type A, then we can first weaken it to obtain the constant family over $\mathbf{1}$, with value A. Then the induction principle of the unit type provides a function

$$\mathsf{ind}_{\mathbf{1}} : A \to (\mathbf{1} \to A).$$

In other words, by the induction principle for the unit type we obtain for every $x : A$ a function $\mathsf{pt}_x :\equiv \mathsf{ind}_{\mathbf{1}}(x) : \mathbf{1} \to A$.

4.3 The empty type

The empty type is a degenerate example of an inductive type. It does *not* come equipped with any constructors, and therefore there are also no computation rules. The induction principle merely asserts that any type family has a section. In other words: if we assume the empty type has a term, then we can prove anything.

Definition 4.3.1 We define the **empty type** to be a type $\emptyset$ satisfying the induction principle that for any family of types $P(x)$ indexed by $x : \emptyset$, there is a term

$$\mathsf{ind}_{\emptyset} : \textstyle\prod_{(x:\emptyset)} P(x).$$

It is again a special case of the induction principle that we have a function

$$\mathsf{ex\text{-}falso} :\equiv \mathsf{ind}_{\emptyset} : \emptyset \to A$$

for any type A. Indeed, to obtain this function one first weakens A to obtain the

constant family over $\emptyset$ with value A, and then the induction principle gives the desired function. The function ex-falso can be used to draw any conclusion after deriving a contradiction, because *ex falso quodlibet*.

We can also use the empty type to define the negation operation on types.

Definition 4.3.2 For any type A we define **negation** of A by

$$\neg A :\equiv A \to \emptyset.$$

We also say that a type A **is empty** if it comes equipped with an element of type $\neg A$. Therefore, we also define

$$\mathsf{is\text{-}empty}(A) :\equiv A \to \emptyset.$$

Remark 4.3.3 Since $\neg A$ is the type of functions from A to $\emptyset$, a proof of $\neg A$ is given by assuming that A holds, and then constructing an element of the empty type. In other words, we prove $\neg A$ by assuming A and deriving a contradiction. This proof technique is called **proof of negation**.

Proofs of negation shound not be confused with proofs by contradiction. Even though a proof of negation involves deriving a contradiction, in logic a **proof by contradiction** of a proposition P is an argument where we conclude that P holds after showing that $\neg P$ implies a contradiction. In other words, a proof by contradiction uses the logical step $\neg\neg P \Rightarrow P$, which is also called **double negation elimination**.

In type theory, however, note that the type $\neg\neg A$ is the type of functions

$$(A \to \emptyset) \to \emptyset.$$

This type is quite different from the type A itself, and with the given rules of type theory it is not possible to construct a function $\neg\neg A \to A$ unless more is known about the type A. In other words, before one can prove by contradiction that there is an element in A, one has to construct a function $\neg\neg A \to A$, and it depends on the specific type A whether this is possible at all. In Exercise 4.3 (d) we will see a situation where we can indeed construct a function $\neg\neg A \to A$. In practice, however, we will rarely use double negation elimination.

In the following proposition we illustrate how to work with the type theoretic definition of negation.

Proposition 4.3.4 *For any two types P and Q, there is a function*

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

Proof The desired function is defined by λ-abstraction, so we begin by assuming that we have a function $f : P \to Q$. Then we have to construct a

function $\neg Q \to \neg P$, which is again constructed by λ-abstraction. We assume that we have $\tilde{q} : \neg Q$. By our definition of $\neg Q$ we see that $\tilde{q}$ is a function $Q \to \emptyset$. Now we have to construct a term of type $\neg P$, which is the type of functions $P \to \emptyset$. We apply λ-abstraction once more, so we assume $p : P$. Now we have

$$\begin{aligned} f &: P \to Q \\ \tilde{q} &: Q \to \emptyset \\ p &: P, \end{aligned}$$

and our goal is to construct a term of the empty type.

Since we have $f : P \to Q$ and $p : P$, we obtain $f(p) : Q$. Moreover, we have $\tilde{q} : Q \to \emptyset$, so we obtain $\tilde{q}(f(p)) : \emptyset$. This completes the proof. The function we have constructed is

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

We leave it to the reader to construct the corresponding natural deduction tree, that formally constructs a function

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

4.4 Coproducts

Definition 4.4.1 Let A and B be types. We define the **coproduct** $A + B$ to be a type that comes equipped with

$$\begin{aligned} \mathsf{inl} &: A \to A + B \\ \mathsf{inr} &: B \to A + B, \end{aligned}$$

satisfying the induction principle that for any family of types $P(x)$ indexed by $x : A + B$, there is a term

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

for which the computation rules

$$\begin{aligned} \mathsf{ind}_+(f, g, \mathsf{inl}(x)) &\doteq f(x) \\ \mathsf{ind}_+(f, g, \mathsf{inr}(y)) &\doteq g(y) \end{aligned}$$

hold. Alternatively, a definition of a dependent function $h : \prod_{(x:A+B)} P(x)$ by induction using $f : \prod_{(x:A)} P(\mathsf{inl}(x))$ and $g : \prod_{(y:B)} P(\mathsf{inr}(y))$ can be presented by pattern matching as

$$h(\mathsf{inl}(x)) :\equiv f(x)$$

$$h(\mathsf{inr}(y)) :\equiv g(y).$$

Sometimes we write $[f, g]$ for the function $\mathsf{ind}_+(f, g)$. The coproduct of two types is sometimes also called the **disjoint sum**.

By the induction principle of coproducts we obtain a function

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

for any type X. Note that this special case of the induction principle of coproducts is very similar to the elimination rule of disjunction in first order logic: if P, P', and Q are propositions, then we have

$$(P \to Q) \to \big((P' \to Q) \to (P \vee P' \to Q)\big).$$

Indeed, we can think of *propositions as types* and of terms as their constructive proofs. Under this interpretation of type theory the coproduct is indeed the disjunction.

Remark 4.4.2 A simple application of the induction principle for coproducts gives us a map

$$f + g : A + B \to A' + B'$$

for every $f : A \to A'$ and $g : B \to B'$. Indeed, the map $f + g$ is defined by

$$(f + g)(\mathsf{inl}(x)) :\equiv \mathsf{inl}(f(x))$$
$$(f + g)(\mathsf{inr}(y)) :\equiv \mathsf{inr}(g(y)).$$

Proposition 4.4.3 *Consider two types A and B, and suppose that B is empty. Then there is a function*

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

Remark 4.4.4 In other words, there is a function

$$\mathsf{is\text{-}empty}(B) \to ((A + B) \to A),$$

for any two types A and B. Similarly, there is a function

$$\mathsf{is\text{-}empty}(A) \to ((A + B) \to B),$$

for any two types A and B.

Proof We will construct the function $(A+B) \to A$ with the induction principle of the coproduct $A + B$. Therefore, we must construct two functions:

$$f : A \to A$$
$$g : B \to A.$$

The function f is simply defined to be the identity function $\mathsf{id} : A \to A$. Recall that we have assumed that B is empty, so we have a function $\tilde{b} : B \to \emptyset$. Furthermore, we always have the function $\mathsf{ex\text{-}falso} : \emptyset \to A$. Therefore, we can define $g :\equiv \mathsf{ex\text{-}falso} \circ \tilde{b}$ to complete the proof. □

4.5 The type of integers

The set of integers is usually defined as a quotient of the set $\mathbb{N} \times \mathbb{N}$, by the equivalence relation

$$((n, m) \sim (n', m')) :\equiv (n + m' = n' + m).$$

We haven't introduced the identity type yet, in order to consider the type of identifications $n + m' = n' + m$, but more importantly there are no quotient types in Martin-Löf's dependent type theory. We will only discuss quotient types in Section 18 after we have assumed the univalence axiom and propositional truncations, because we will use the univalence axiom and propositional truncations to define them and derive their basic properties. Nevertheless, the type of integers is also definable in dependent type theory without set quotients, but we have to settle for a more pedestrian version of the integers that is defined using coproducts.

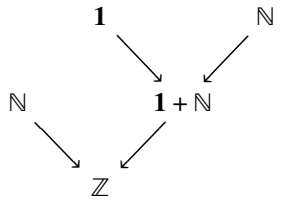
Definition 4.5.1 We define the **integers** to be the type $\mathbb{Z} :\equiv \mathbb{N} + (\mathbf{1} + \mathbb{N})$. The type of integers comes equipped with inclusion functions of the positive and negative integers

$$\begin{aligned} \mathsf{in\text{-}pos} &:\equiv \mathsf{inr} \circ \mathsf{inr} &&: \mathbb{N} \to \mathbb{Z} \\ \mathsf{in\text{-}neg} &:\equiv \mathsf{inl} &&: \mathbb{N} \to \mathbb{Z} \end{aligned}$$

and with the constants

$$\begin{aligned} -1_{\mathbb{Z}} &:\equiv \mathsf{in\text{-}neg}(0) \\ 0_{\mathbb{Z}} &:\equiv \mathsf{inr}(\mathsf{inl}(\star)) \\ 1_{\mathbb{Z}} &:\equiv \mathsf{in\text{-}pos}(0). \end{aligned}$$

The definition of the integers as the coproduct $\mathbb{N} + (\mathbf{1} + \mathbb{N})$ can be pictured as follows:



Remark 4.5.2 The type of integers is built entirely out of inductive types. Therefore it is possible to derive an induction principle especially tailored for the type $\mathbb{Z}$, which can be used to define the basic operations on $\mathbb{Z}$, such as the successor map, addition and multiplication. This induction principle asserts that for any type family P over $\mathbb{Z}$, we can define a dependent function $f : \prod_{(k:\mathbb{Z})} P(k)$ recursively by

$$\begin{aligned} f(-1_{\mathbb{Z}}) &\coloneqq p_{-1} \\ f(\mathsf{in\text{-}neg}(\mathsf{succ}_{\mathbb{N}}(n))) &\coloneqq p_{-S}(n, f(\mathsf{in\text{-}neg}(n))) \\ f(0_{\mathbb{Z}}) &\coloneqq p_0 \\ f(1_{\mathbb{Z}}) &\coloneqq p_1 \\ f(\mathsf{in\text{-}pos}(\mathsf{succ}_{\mathbb{N}}(n))) &\coloneqq p_S(n, f(\mathsf{in\text{-}pos}(n))), \end{aligned}$$

where the types of p_{-1}, p_{-S}, p_0, p_1, and p_S are

$$\begin{aligned} p_{-1} &: P(-1_{\mathbb{Z}}) \\ p_{-S} &: \textstyle\prod_{(n:\mathbb{N})} P(\mathsf{in\text{-}neg}(n)) \to P(\mathsf{in\text{-}neg}(\mathsf{succ}_{\mathbb{N}}(n))) \\ p_0 &: P(0_{\mathbb{Z}}) \\ p_1 &: P(1_{\mathbb{Z}}) \\ p_S &: \textstyle\prod_{(n:\mathbb{N})} P(\mathsf{in\text{-}pos}(n)) \to P(\mathsf{in\text{-}pos}(\mathsf{succ}_{\mathbb{N}}(n))). \end{aligned}$$

Definition 4.5.3 We define the **successor function** on the integers $\mathsf{succ}_{\mathbb{Z}} : \mathbb{Z} \to \mathbb{Z}$ using the induction principle of Remark 4.5.2, taking

$$\begin{aligned} \mathsf{succ}_{\mathbb{Z}}(-1_{\mathbb{Z}}) &\coloneqq 0_{\mathbb{Z}} \\ \mathsf{succ}_{\mathbb{Z}}(\mathsf{in\text{-}neg}(\mathsf{succ}_{\mathbb{N}}(n))) &\coloneqq \mathsf{in\text{-}neg}(n) \\ \mathsf{succ}_{\mathbb{Z}}(0_{\mathbb{Z}}) &\coloneqq 1_{\mathbb{Z}} \\ \mathsf{succ}_{\mathbb{Z}}(1_{\mathbb{Z}}) &\coloneqq \mathsf{in\text{-}pos}(1_{\mathbb{N}}) \\ \mathsf{succ}_{\mathbb{Z}}(\mathsf{in\text{-}pos}(\mathsf{succ}_{\mathbb{N}}(n))) &\coloneqq \mathsf{in\text{-}pos}(\mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n))). \end{aligned}$$

4.6 Dependent pair types

Given a type family B over A, we may consider pairs (a, b) of terms, where $a : A$ and $b : B(a)$. Note that the type of b depends on the first term in the pair. Therefore we call such a pair a **dependent pair**. The type of such dependent pairs is the inductive type that is generated by the dependent pairs.

Definition 4.6.1 Consider a type family B over A. The **dependent pair type**

(or **Σ-type**) is defined to be the inductive type $\sum_{(x:A)} B(x)$ equipped with a **pairing function**

$$\mathsf{pair} : \prod_{(x:A)} \Big(B(x) \to \sum_{(y:A)} B(y) \Big).$$

The induction principle for $\sum_{(x:A)} B(x)$ asserts that for any family of types $P(p)$ indexed by $p : \sum_{(x:A)} B(x)$, there is a function

$$\mathsf{ind}_\Sigma : \Big(\prod_{(x:A)} \prod_{(y:B(x))} P(\mathsf{pair}(x,y)) \Big) \to \Big(\prod_{(z:\sum_{(x:A)} B(x))} P(z) \Big).$$

satisfying the computation rule

$$\mathsf{ind}_\Sigma(g, \mathsf{pair}(x,y)) \doteq g(x,y).$$

Alternatively, a definition of a dependent function $f : \prod_{(z:\sum_{(x:A)} B(x))} P(z)$ by induction using a function $g : \prod_{(x:A)} \prod_{(y:B(x))} P((x,y))$ can be presented by pattern matching as

$$f(\mathsf{pair}(x,y)) :\equiv g(x,y).$$

We will usually write (x,y) for $\mathsf{pair}(x,y)$.

The induction principle of Σ-types can be used to define the projection functions.

Definition 4.6.2 Consider a type A and a type family B over A.

(i) The **first projection map**

$$\mathsf{pr}_1 : \Big(\sum_{(x:A)} B(x) \Big) \to A$$

is defined by induction as

$$\mathsf{pr}_1(x,y) :\equiv x.$$

(ii) The **second projection map** is a dependent function

$$\mathsf{pr}_2 : \prod_{(p:\sum_{(x:A)} B(x))} B(\mathsf{pr}_1(p)),$$

defined by induction as

$$\mathsf{pr}_2(x,y) :\equiv y.$$

Remark 4.6.3 If we want to construct a function

$$f : \prod_{(z:\sum_{(x:A)} B(x))} P(z)$$

by Σ-induction, then we get to assume a pair (x,y) consisting of $x : A$ and $y : B(x)$ and our goal will be to construct an element of type $P(x,y)$. The

induction principle of Σ-types is therefore converse to the **currying operation**, a familiar concept from the theory of programming languages, which is given by the function

$$\mathsf{ev\text{-}pair} : \left(\prod_{(z:\sum_{(x:A)} B(x))} P(z)\right) \to \left(\prod_{(x:A)} \prod_{(y:B(x))} P(x,y)\right)$$

given by $f \mapsto \lambda x.\, \lambda y.\, f(x,y)$. The induction principle ind_Σ is therefore also known as the **uncurrying operation**.

A common special case of the Σ-type occurs when the B is a constant family over A, i.e., when B is just a type weakened by A. In this case, the type $\sum_{(x:A)} B$ is the type of *ordinary* pairs (x, y) where $x : A$ and $y : B$, where the type of y does not depend on x. The type of ordinary pairs (x, y) consisting of $x : A$ and $y : B$ is of course the *product* of A and B, so we see that product types arise as a special case of Σ-types in a similar way to how function types were defined as a special case of Π-types.

Definition 4.6.4 Consider two types A and B. Then we define the **(cartesian) product** $A \times B$ of A and B by

$$A \times B :\equiv \sum_{(x:A)} B.$$

Remark 4.6.5 Since $A \times B$ is defined as a Σ-type, it follows that cartesian products also satisfy the induction principle of Σ-types. In this special case, the induction principle for $A \times B$ asserts that for any type family P over $A \times B$ there is a function

$$\mathsf{ind}_\times : \left(\prod_{(x:A)} \prod_{(y:B)} P(x,y)\right) \to \left(\prod_{(z:A\times B)} P(z)\right)$$

that satisfies the computation rule

$$\mathsf{ind}_\times(g, (x,y)) \doteq g(x,y).$$

The projection maps are defined similarly to the projection maps of Σ-types. When one thinks of types as propositions, then $A \times B$ is interpreted as the conjunction of A and B.

Exercises

4.1 (a) Define the predecessor function $\mathsf{pred}_\mathbb{Z} : \mathbb{Z} \to \mathbb{Z}$.
(b) Define the group operations

$$\mathsf{add}_\mathbb{Z} : \mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})$$
$$\mathsf{neg}_\mathbb{Z} : \mathbb{Z} \to \mathbb{Z}.$$

(c) Define the multiplication operation $\mathsf{mul}_{\mathbb{Z}} : \mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})$.

4.2 The type of **booleans** is defined to be an inductive type bool that comes equipped with

$$\mathsf{false} : \mathsf{bool} \qquad \text{and} \qquad \mathsf{true} : \mathsf{bool}.$$

The induction principle of the booleans asserts that for any family of types $P(x)$ indexed by $x : \mathsf{bool}$, there is a term

$$\mathsf{ind\text{-}bool} : P(\mathsf{false}) \to \left(P(\mathsf{true}) \to \textstyle\prod_{(x:\mathsf{bool})} P(x)\right)$$

for which the computation rules

$$\mathsf{ind\text{-}bool}(p_0, p_1, \mathsf{false}) \doteq p_0$$
$$\mathsf{ind\text{-}bool}(p_0, p_1, \mathsf{true}) \doteq p_1$$

hold.

(a) Construct the negation function $\mathsf{neg\text{-}bool} : \mathsf{bool} \to \mathsf{bool}$.

(b) Construct the boolean conjuntion operation $- \wedge - : \mathsf{bool} \to (\mathsf{bool} \to \mathsf{bool})$.

(c) Construct the boolean disjunction operation $- \vee - : \mathsf{bool} \to (\mathsf{bool} \to \mathsf{bool})$.

4.3 Let P and Q be types. We will write $P \leftrightarrow Q$ for the type $(P \to Q) \times (Q \to P)$. Use the fact that $\neg P$ is defined as the type $P \to \emptyset$ of functions from P to the empty type to give type theoretic proofs of the constructive tautologies in this exercise.

(a) Show that

(i) $\neg(P \times \neg P)$

(ii) $\neg(P \leftrightarrow \neg P)$.

(b) Construct the following maps in the structure of the **double negation monad**:

(i) $P \to \neg\neg P$

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

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

(c) Prove that the following double negations of classical laws hold:

(i) $\neg\neg(\neg\neg P \to P)$

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

(iii) $\neg\neg((P \to Q) + (Q \to P))$

(iv) $\neg\neg(P + \neg P)$.

(d) Show that

(i) $(P + \neg P) \to (\neg\neg P \to P)$
(ii) $\neg\neg(Q \to P) \leftrightarrow ((P + \neg P) \to (Q \to P))$.

(e) Prove the following tautologies, showing that $\neg P$, $P \to \neg\neg Q$, and $\neg\neg P \times \neg\neg Q$ are **double negation stable**:

(i) $\neg\neg\neg P \to \neg P$
(ii) $\neg\neg(P \to \neg\neg Q) \to (P \to \neg\neg Q)$
(iii) $\neg\neg((\neg\neg P) \times (\neg\neg Q)) \to (\neg\neg P) \times (\neg\neg Q)$.

(f) Show that

(i) $\neg\neg(P \times Q) \leftrightarrow (\neg\neg P) \times (\neg\neg Q)$
(ii) $\neg\neg(P + Q) \leftrightarrow \neg(\neg P \times \neg Q)$
(iii) $\neg\neg(P \to Q) \leftrightarrow (\neg\neg P \to \neg\neg Q)$.

4.4 For any type A we can define the type $\mathsf{list}(A)$ of **lists** of elements of A as the inductive type with constructors

$$\mathsf{nil} : \mathsf{list}(A)$$
$$\mathsf{cons} : A \to (\mathsf{list}(A) \to \mathsf{list}(A)).$$

(a) Write down the induction principle and the computation rules for $\mathsf{list}(A)$.
(b) Let A and B be types, suppose that $b : B$, and consider a binary operation $\mu : A \to (B \to B)$. Define a function

$$\mathsf{fold\text{-}list}(\mu) : \mathsf{list}(A) \to B$$

that iterates the operation μ, starting with $\mathsf{fold\text{-}list}(\mu, \mathsf{nil}) :\equiv b$.
(c) Define the operation

$$\mathsf{map\text{-}list} : (A \to B) \to (\mathsf{list}(A) \to \mathsf{list}(B))$$

for any two types A and B.
(d) Define a function $\mathsf{length\text{-}list} : \mathsf{list}(A) \to \mathbb{N}$.
(e) Define the functions

$$\mathsf{sum\text{-}list} : \mathsf{list}(\mathbb{N}) \to \mathbb{N}$$
$$\mathsf{product\text{-}list} : \mathsf{list}(\mathbb{N}) \to \mathbb{N},$$

where sum-list adds all the elements in a list of natural numbers, and product-list takes their product.
(f) Define a function

$$\mathsf{concat\text{-}list} : \mathsf{list}(A) \to (\mathsf{list}(A) \to \mathsf{list}(A))$$

that concatenates any two lists of elements in A.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

(g) Define a function

$$\mathsf{flatten\text{-}list} : \mathsf{list}(\mathsf{list}(A)) \to \mathsf{list}(A)$$

that concatenates all the lists in a lists of lists in A.

(h) Define a function $\mathsf{reverse\text{-}list} : \mathsf{list}(A) \to \mathsf{list}(A)$ that reverses the order of the elements in any list.

5 Identity types

From the perspective of types as proof-relevant propositions, how should we think of *equality* in type theory? Given a type A, and two elements $x, y : A$, the equality $x = y$ should again be a type. Indeed, we want to *use* type theory to prove equalities. *Dependent* type theory provides us with a convenient setting for this: the equality type $x = y$ is dependent on $x, y : A$.

Then, if $x = y$ is to be a type, how should we think of the elements of $x = y$. An element $p : x = y$ witnesses that x and y are equal elements of type A. In other words $p : x = y$ is an *identification* of x and y. In a proof-relevant world, there might be many elements of type $x = y$. I.e., there might be many identifications of x and y. And, since $x = y$ is itself a type, we can form the type $p = q$ for any two identifications $p, q : x = y$. That is, since $x = y$ is a type, we may also use the type theory to prove things *about* identifications (for instance, that two given such identifications can themselves be identified), and we may use the type theory to perform constructions with them. As we will see shortly, we can give every type a groupoidal structure.

Clearly, the equality type should not just be any type dependent on $x, y : A$. Then how do we form the equality type, and what ways are there to use identifications in constructions in type theory? The answer to both these questions is that we will form the identity type as an *inductive* type, generated by just a reflexivity identification providing an identification of x to itself. The induction principle then provides us with a way of performing constructions with identifications, such as concatenating them, inverting them, and so on. Thus, the identity type is equipped with a reflexivity element, and further possesses the structure that are generated by its induction principle and by the type theory. This inductive construction of the identity type is elegant, beautifully simple, but far from trivial!

The situation where two elements can be identified in possibly more than one way is analogous to the situation in *homotopy theory*, where two points of a space can be connected by possibly more than one *path*. Indeed, for any two points x, y in a space, there is a *space of paths* from x to y. Moreover, between

Table I.1 *The homotopy interpretation*

Type theory	*Homotopy theory*
Types	Spaces
Dependent types	Fibrations
Elements	Points
Dependent pair type	Total space
Identity type	Path fibration

any two paths from x to y there is a space of *homotopies* between them, and so on. This leads to the homotopy interpretation of type theory, outlined in Table I.1. The connection between homotopy theory and type theory has been made precise by the construction of homotopical models of type theory, and it has led to the fruitful research area of *synthetic homotopy theory*, the topic of Chapter III.

5.1 The inductive definition of identity types

Definition 5.1.1 Consider a type A and let $a : A$. Then we define the **identity type** of A at a as an inductive family of types $a =_A x$ indexed by $x : A$, of which the constructor is

$$\mathsf{refl}_a : a =_A a.$$

The induction principle of the identity type postulates that for any family of types $P(x, p)$ indexed by $x : A$ and $p : a =_A x$, there is a function

$$\mathsf{ind\text{-}eq}_a : P(a, \mathsf{refl}_a) \to \textstyle\prod_{(x:A)} \prod_{(p:a=_A x)} P(x, p)$$

that satisfies $\mathsf{ind\text{-}eq}_a(p, a, \mathsf{refl}_a) \doteq p$.

An element of type $a =_A x$ is also called an **identification** of a with x, and sometimes it is called a **path** from a to x. The induction principle for identity types is sometimes called **identification elimination** or **path induction**. We also write Id_A for the identity type on A, and often we write $a = x$ for the type of identifications of a with x, omitting reference to the ambient type A.

Remark 5.1.2 We see that the identity type is not just an inductive type, like the inductive types $\mathbb{N}$, $\emptyset$, and $\mathbf{1}$ for example, but it is an inductive *family* of types. Even though we have a type $a =_A x$ for any $x : A$, the constructor only provides an element $\mathsf{refl}_a : a =_A a$, identifying a with itself. The induction principle then asserts that in order to prove something about all identifications of a with some $x : A$, it suffices to prove this assertion about refl_a only. We will see in the next

sections that this induction principle is strong enough to derive many familiar facts about equality, namely that it is a symmetric and transitive relation, and that all functions preserve equality.

Remark 5.1.3 Since the identity types require getting used to, we provide the formal rules for identity types. The identity type is formed by the formation rule:

$$\frac{\Gamma \vdash a : A}{\Gamma, x : A \vdash a =_A x \text{ type}}$$

The constructor of the identity type is then given by the introduction rule:

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_a : a =_A a}$$

The induction principle is now given by the elimination rule:

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A, p : a =_A x \vdash P(x, p) \text{ type}}{\Gamma \vdash \mathsf{ind\text{-}eq}_a : P(a, \mathsf{refl}_a) \to \prod_{(x:A)} \prod_{(p:a=_A x)} P(x, p)}$$

And finally the computation rule is:

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A, p : a =_A x \vdash P(x, p) \text{ type}}{\Gamma, u : P(a, \mathsf{refl}_A) \vdash \mathsf{ind\text{-}eq}_a(u, a, \mathsf{refl}_a) \doteq u : P(a, \mathsf{refl}_a)}$$

Remark 5.1.4 One might wonder whether it is also possible to form the identity type at a *variable* of type A, rather than at an element. This is certainly possible: since we can form the identity type in *any* context, we can form the identity type at a variable $x : A$ as follows:

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

In this way we obtain the 'binary' identity type. Its constructor is then also indexed by $x : A$. We have the following introduction rule

$$\frac{\Gamma, x : A \vdash x : A}{\Gamma, x : A \vdash \mathsf{refl}_x : x =_A x}$$

and similarly we have elimination and computation rules.

5.2 The groupoidal structure of types

We show that identifications can be *concatenated* and *inverted*, which corresponds to the transitivity and symmetry of the identity type.

Definition 5.2.1 Let A be a type. We define the **concatenation** operation

$$\mathsf{concat} : \prod_{(x,y,z:A)} (x = y) \to ((y = z) \to (x = z)).$$

We will write $p \cdot q$ for $\mathsf{concat}(p, q)$.

Construction We first construct a function

$$f(x) : \prod_{(y:A)} (x = y) \to \prod_{(z:A)} (y = z) \to (x = z)$$

for any $x : A$. By the induction principle for identity types, it suffices to construct

$$f(x, x, \mathsf{refl}_x) : \prod_{(z:A)} (x = z) \to (x = z).$$

Here we have the function $\lambda z.\, \mathsf{id}_{(x=z)}$. The function $f(x)$ we obtain via identity elimination is explicitly thus defined as

$$f(x) :\equiv \mathsf{ind\text{-}eq}_x(\lambda z.\, \mathsf{id}) : \prod_{(y:A)} (x = y) \to \prod_{(z:A)} (y = z) \to (x = z).$$

To finish the construction of concat, we use Exercise 2.4 to swap the order of the third and fourth variable of f, i.e., we define

$$\mathsf{concat}_{x,y,z}(p, q) :\equiv f(x, y, p, z, q).$$ □

Definition 5.2.2 Let A be a type. We define the **inverse operation**

$$\mathsf{inv} : \prod_{(x,y:A)} (x = y) \to (y = x).$$

Most of the time we will write p^{-1} for $\mathsf{inv}(p)$.

Construction By the induction principle for identity types, it suffices to construct

$$\mathsf{inv}(\mathsf{refl}_x) : x = x,$$

for any $x : A$. Here we take $\mathsf{inv}(\mathsf{refl}_x) :\equiv \mathsf{refl}_x$. □

The next question is whether the concatenation and inverting operations on identifications behave as expected. More concretely: is concatenation of identifications associative, does it satisfy the unit laws, and is the inverse of an identification indeed a two-sided inverse?

For example, in the case of associativity we are asking to compare the identifications

$$(p \cdot q) \cdot r \qquad \text{and} \qquad p \cdot (q \cdot r)$$

for any $p : x = y$, $q : y = z$, and $r : z = w$ in a type A. The computation rules of the identity type are not strong enough to conclude that $(p \cdot q) \cdot r$ and $p \cdot (q \cdot r)$ are judgmentally equal. However, both $(p \cdot q) \cdot r$ and $p \cdot (q \cdot r)$ are elements of the same type: they are identifications of type $x = w$. Since the identity type is a type like any other, we can ask whether there is an *identification*

$$(p \cdot q) \cdot r = p \cdot (q \cdot r).$$

This is a very useful idea: while it is often impossible to show that two elements of the same type are judgmentally equal, it may be the case that those two elements can be *identified*. Indeed, we identify two elements by constructing an element of the identity type, and we can use all the type theory at our disposal in order to construct such an element. In this way we can show, for example, that addition on the natural numbers or on the integers is associative and satisfies the unit laws. And indeed, here we will show that concatenation of identifications is associative and satisfies the unit laws.

Definition 5.2.3 Let A be a type and consider three consecutive identifications

$$x \overset{p}{=\!=} y \overset{q}{=\!=} z \overset{r}{=\!=} w$$

in A. We define the **associator**

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

Construction By the induction principle for identity types it suffices to show that

$$\textstyle\prod_{(z:A)} \prod_{(q:x=z)} \prod_{(w:A)} \prod_{(r:z=w)} (\mathsf{refl}_x \cdot q) \cdot r = \mathsf{refl}_x \cdot (q \cdot r).$$

Let $q : x = z$ and $r : z = w$. Note that by the computation rule of identity types we have a judgmental equality $\mathsf{refl}_x \cdot q \doteq q$. Therefore we conclude that

$$(\mathsf{refl}_x \cdot q) \cdot r \doteq q \cdot r.$$

Similarly we have a judgmental equality $\mathsf{refl}_x \cdot (q \cdot r) \doteq q \cdot r$. Thus we see that the left-hand side and the right-hand side in

$$(\mathsf{refl}_x \cdot q) \cdot r = \mathsf{refl}_x \cdot (q \cdot r)$$

are judgmentally equal, so we can simply define $\mathsf{assoc}(\mathsf{refl}_x, q, r) :\equiv \mathsf{refl}_{q \cdot r}$. □

Definition 5.2.4 Let A be a type. We define the left and right **unit law operations**, which assigns to each $p : x = y$ the identifications

$$\mathsf{left\text{-}unit}(p) : \mathsf{refl}_x \cdot p = p$$
$$\mathsf{right\text{-}unit}(p) : p \cdot \mathsf{refl}_y = p,$$

respectively.

Construction By identification elimination it suffices to construct

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

In both cases we take $\mathsf{refl}_{\mathsf{refl}_x}$. □

Definition 5.2.5 Let A be a type. We define left and right **inverse law operations**

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

Construction By identification elimination it suffices to construct

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

Using the computation rules we see that

$$\mathsf{refl}_x^{-1} \cdot \mathsf{refl}_x \doteq \mathsf{refl}_x \cdot \mathsf{refl}_x \doteq \mathsf{refl}_x,$$

so we define $\mathsf{left\text{-}inv}(\mathsf{refl}_x) :\equiv \mathsf{refl}_{\mathsf{refl}_x}$. Similarly it follows from the computation rules that

$$\mathsf{refl}_x \cdot \mathsf{refl}_x^{-1} \doteq \mathsf{refl}_x^{-1} \doteq \mathsf{refl}_x$$

so we again define $\mathsf{right\text{-}inv}(\mathsf{refl}_x) :\equiv \mathsf{refl}_{\mathsf{refl}_x}$. □

Remark 5.2.6 We have seen that the associator, the unit laws, and the inverse laws, are all proven by constructing an identification of identifications. And indeed, there is nothing that would stop us from considering identifications of those identifications of identifications. We can go up as far as we like in the *tower of identity types*, which is obtained by iteratively taking identity types.

The iterated identity types give types in homotopy type theory a very intricate structure. One important way of studying this structure is via the homotopy groups of types, a subject that we will gradually be working towards.

5.3 The action on identifications of functions

Using the induction principle of the identity type we can show that every function preserves identifications. In other words, every function sends identified elements to identified elements. Note that this is a form of continuity for functions in type

theory: if there is an identification that identifies two points x and y of a type A, then there also is an identification that identifies the values $f(x)$ and $f(y)$ in the codomain of f.

Definition 5.3.1 Let $f : A \to B$ be a map. We define the **action on paths** of f as an operation

$$\mathsf{ap}_f : \prod_{(x,y:A)} (x = y) \to (f(x) = f(y)).$$

Moreover, there are operations

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

Construction First we define ap_f by the induction principle of identity types, taking

$$\mathsf{ap}_f(\mathsf{refl}_x) :\equiv \mathsf{refl}_{f(x)}.$$

Next, we construct $\mathsf{ap\text{-}id}_A$ by the induction principle of identity types, taking

$$\mathsf{ap\text{-}id}_A(\mathsf{refl}_x) :\equiv \mathsf{refl}_{\mathsf{refl}_x}.$$

Finally, we construct $\mathsf{ap\text{-}comp}(f,g)$ by the induction principle of identity types, taking

$$\mathsf{ap\text{-}comp}(f,g,\mathsf{refl}_x) :\equiv \mathsf{refl}_{\mathsf{refl}_{g(f(x))}}.$$ □

Definition 5.3.2 Let $f : A \to B$ be a map. Then there are identifications

$$\mathsf{ap\text{-}refl}(f,x) : \mathsf{ap}_f(\mathsf{refl}_x) = \mathsf{refl}_{f(x)}$$
$$\mathsf{ap\text{-}inv}(f,p) : \mathsf{ap}_f(p^{-1}) = \mathsf{ap}_f(p)^{-1}$$
$$\mathsf{ap\text{-}concat}(f,p,q) : \mathsf{ap}_f(p \cdot q) = \mathsf{ap}_f(p) \cdot \mathsf{ap}_f(q)$$

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

Construction To construct $\mathsf{ap\text{-}refl}(f,x)$ we simply observe that $\mathsf{ap}_f(\mathsf{refl}_x) \doteq \mathsf{refl}_{f(x)}$, so we take

$$\mathsf{ap\text{-}refl}(f,x) :\equiv \mathsf{refl}_{\mathsf{refl}_{f(x)}}.$$

We construct $\mathsf{ap\text{-}inv}(f,p)$ by identification elimination on p, taking

$$\mathsf{ap\text{-}inv}(f,\mathsf{refl}_x) :\equiv \mathsf{refl}_{\mathsf{ap}_f(\mathsf{refl}_x)}.$$

Finally we construct $\mathsf{ap\text{-}concat}(f,p,q)$ by identification elimination on p, taking

$$\mathsf{ap\text{-}concat}(f,\mathsf{refl}_x,q) :\equiv \mathsf{refl}_{\mathsf{ap}_f(q)}.$$ □

5.4 Transport

Dependent types also come with an action on identifications: the *transport* functions. Given an identification $p : x = y$ in the base type A, we can transport any element $b : B(x)$ to the fiber $B(y)$.

Definition 5.4.1 Let A be a type, and let B be a type family over A. We will construct a **transport** operation

$$\mathsf{tr}_B : \textstyle\prod_{(x,y:A)} (x = y) \to (B(x) \to B(y)).$$

Construction We construct $\mathsf{tr}_B(p)$ by induction on $p : x =_A y$, taking

$$\mathsf{tr}_B(\mathsf{refl}_x) :\equiv \mathsf{id}_{B(x)}. \qquad \square$$

Thus we see that type theory cannot distinguish between identified elements x and y, because for any type family B over A one obtains an element of $B(y)$ from the elements of $B(x)$.

As an application of the transport function we construct the *dependent* action on paths of a dependent function $f : \prod_{(x:A)} B(x)$. Note that for such a dependent function f, and an identification $p : x =_A y$, it does not make sense to directly compare $f(x)$ and $f(y)$, since the type of $f(x)$ is $B(x)$ whereas the type of $f(y)$ is $B(y)$, which might not be exactly the same type. However, we can first *transport* $f(x)$ along p, so that we obtain the element $\mathsf{tr}_B(p, f(x))$ which is of type $B(y)$. Now we can ask whether it is the case that $\mathsf{tr}_B(p, f(x)) = f(y)$. The dependent action on paths of f establishes this identification.

Definition 5.4.2 Given a dependent function $f : \prod_{(a:A)} B(a)$ and an identification $p : x = y$ in A, we construct an identification

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

Construction The identification $\mathsf{apd}_f(p)$ is constructed by the induction principle for identity types. Thus, it suffices to construct an identification

$$\mathsf{apd}_f(\mathsf{refl}_x) : \mathsf{tr}_B(\mathsf{refl}_x, f(x)) = f(x).$$

Since transporting along refl_x is the identity function on $B(x)$, we simply take $\mathsf{apd}_f(\mathsf{refl}_x) :\equiv \mathsf{refl}_{f(x)}$. $\square$

5.5 The uniqueness of refl

The identity type is an inductive *family* of types. This has some subtle, but important implications. For instance, while the type $a = x$ indexed by $x : A$ is inductively generated by refl_a, the type $a = a$ is *not* inductively generated by

refl_a. Hence we cannot use the induction principle of identity types to show that $p = \mathsf{refl}_a$ for any $p : a = a$. The obstacle, which prevents us from applying the induction principle of identity types in this case, is that the endpoint of $p : a = a$ is not free.

Nevertheless, the identity type $a = x$ is generated by a single element $\mathsf{refl}_a : a = a$, so it is natural to wonder in what sense the reflexivity identification is unique. An identification with an element a is specified by first giving the endpoint x with which we seek to identify a, and then giving the identification $p : a = x$. It is therefore only the pair (a, refl_a) which is unique in the type of all pairs

$$(x, p) : \textstyle\sum_{(x:A)} a = x.$$

We prove this fact in the following proposition.

Proposition 5.5.1 *Consider an element $a : A$. Then there is an identification*

$$(a, \mathsf{refl}_a) = y$$

in the type $\sum_{(x:A)} a = x$, for any $y : \sum_{(x:A)} a = x$.

Proof By Σ-induction it suffices to show that there is an identification

$$(a, \mathsf{refl}_a) = (x, p)$$

for any $x : A$ and $p : a = x$. We proceed by the induction principle of identity types. Therefore it suffices to show that

$$(a, \mathsf{refl}_a) = (a, \mathsf{refl}_a).$$

We obtain such an identification by reflexivity. □

Proposition 5.5.1 shows that there is, up to identification, only one element in Σ-type of the identity type. Such types are called contractible, and they are the subject of Section 10.

5.6 The laws of addition on $\mathbb{N}$

Now that we have introduced the identity type, we can start proving equations. We will prove here that there are identifications

$$\begin{aligned} 0 + n &= n & m + 0 &= m \\ \mathsf{succ}_{\mathbb{N}}(m) + n &= \mathsf{succ}_{\mathbb{N}}(m + n) & m + \mathsf{succ}_{\mathbb{N}}(n) &= \mathsf{succ}_{\mathbb{N}}(m + n) \\ (m + n) + k &= m + (n + k) & m + n &= n + m. \end{aligned}$$

The unit laws, associativity, and commutativity of addition are of course familiar. The successor laws will be useful to prove commutativity. In Exercise 5.5 you will be asked to prove the laws of multiplication on $\mathbb{N}$. There will again be *successor laws* as part of this exercise, because they are useful intermediate steps in the more complicated laws.

Recall that addition on the natural numbers is defined in such a way that

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

These two judgmental equalities are all we currently know about the function $m, n \mapsto m + n$ on $\mathbb{N}$. Consequently, we will have to find ways to apply these two judgmental equalities in our proofs of the laws of addition. Of course, the judgmental equalities coincide with two of the six laws. For the remaining four laws, we will have to proceed by induction on $\mathbb{N}$.

Proposition 5.6.1 *For any natural number n, there are identifications*

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

Proof We can define

$$\mathsf{right\text{-}unit\text{-}law\text{-}add}_{\mathbb{N}}(n) :\equiv \mathsf{refl}_n,$$

because the computation rule for addition gives us that $n + 0 \doteq n$.

It remains to define the left unit law. We proceed by induction on n. In the base case we have to show that $0 + 0 = 0$, which holds by reflexivity. For the inductive step, assume that we have an identification $p : 0 + n = n$. Our goal is to show that $0 + \mathsf{succ}_{\mathbb{N}}(n) = \mathsf{succ}_{\mathbb{N}}(n)$. However, it suffices to construct an identification

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

because by the computation rule for addition we have that $0 + \mathsf{succ}_{\mathbb{N}}(n) \doteq \mathsf{succ}_{\mathbb{N}}(0 + n)$. Now we use the action on paths of $\mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ to obtain

$$\mathsf{ap}_{\mathsf{succ}_{\mathbb{N}}}(p) : \mathsf{succ}_{\mathbb{N}}(0 + n) = \mathsf{succ}_{\mathbb{N}}(n).$$

The left unit law is therefore defined by

$$\mathsf{left\text{-}unit\text{-}law\text{-}add}_{\mathbb{N}}(n) :\equiv \mathsf{ind}_{\mathbb{N}}(\mathsf{refl}_0, \lambda p.\, \mathsf{ap}_{\mathsf{succ}_{\mathbb{N}}}(p)). \qquad \square$$

Proposition 5.6.2 *For any natural numbers m and n, there are identifications*

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

Proof We can define

$$\mathsf{right\text{-}successor\text{-}law\text{-}add}_{\mathbb{N}}(m,n) :\equiv \mathsf{refl}_{\mathsf{succ}_{\mathbb{N}}(m+n)}$$

because we have a judgmental equality $m + \mathsf{succ}_{\mathbb{N}}(n) \doteq \mathsf{succ}_{\mathbb{N}}(m+n)$ by the computation rules for $\mathsf{add}_{\mathbb{N}}$.

The left successor law is constructed by induction on n. In the base case we have to construct an identification $\mathsf{succ}_{\mathbb{N}}(m) + 0 = \mathsf{succ}_{\mathbb{N}}(m+0)$, which is obtained by reflexivity. For the inductive step, assume that we have an identification $p : \mathsf{succ}_{\mathbb{N}}(m) + n = \mathsf{succ}_{\mathbb{N}}(m+n)$. Our goal is to show that

$$\mathsf{succ}_{\mathbb{N}}(m) + \mathsf{succ}_{\mathbb{N}}(n) = \mathsf{succ}_{\mathbb{N}}(m + \mathsf{succ}_{\mathbb{N}}(n)).$$

Note that we have the judgmental equalities

$$\begin{aligned}\mathsf{succ}_{\mathbb{N}}(m) + \mathsf{succ}_{\mathbb{N}}(n) &\doteq \mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m) + n)\\ \mathsf{succ}_{\mathbb{N}}(m + \mathsf{succ}_{\mathbb{N}}(n)) &\doteq \mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m+n))\end{aligned}$$

Therefore it suffices to construct an identification

$$\mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m) + n) = \mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m+n)).$$

Such an identification is given by $\mathsf{ap}_{\mathsf{succ}_{\mathbb{N}}}(p)$. □

Proposition 5.6.3 *Addition on the natural numbers is associative, i.e., for any three natural numbers m, n, and k, there is an identification*

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

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

$$(m+n)+0 \doteq m+n \doteq m+(n+0).$$

Therefore we define $\mathsf{associative\text{-}add}_{\mathbb{N}}(m,n,0) :\equiv \mathsf{refl}_{m+n}$.

For the inductive step, let $p : (m+n)+k = m+(n+k)$. Our goal is to show that

$$(m+n) + \mathsf{succ}_{\mathbb{N}}(k) = m + (n + \mathsf{succ}_{\mathbb{N}}(k)).$$

Note that we have the judgmental equalities

$$\begin{aligned}(m+n) + \mathsf{succ}_{\mathbb{N}}(k) &\doteq \mathsf{succ}_{\mathbb{N}}((m+n)+k)\\ m + (n + \mathsf{succ}_{\mathbb{N}}(k)) &\doteq m + (\mathsf{succ}_{\mathbb{N}}(n+k))\\ &\doteq \mathsf{succ}_{\mathbb{N}}(m+(n+k))\end{aligned}$$

Therefore it suffices to construct an identification

$$\mathsf{succ}_{\mathbb{N}}((m+n)+k) = \mathsf{succ}_{\mathbb{N}}(m+(n+k)),$$

which we have by $\mathsf{ap}_{\mathsf{succ}_\mathbb{N}}(p)$. □

Proposition 5.6.4 *Addition on the natural numbers is commutative, i.e., for any two natural numbers m and n there is an identification*

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

Proof We construct $\mathsf{commutative\text{-}add}_\mathbb{N}(m,n)$ by induction on m. In the base case we have to show that $0 + n = n + 0$, which holds by the unit laws for n, proven in Proposition 5.6.1.

For the inductive step, let $p : m + n = n + m$. Our goal is to construct an identification $\mathsf{succ}_\mathbb{N}(m) + n = n + \mathsf{succ}_\mathbb{N}(m)$. Now it is clear why we first proved the successor laws: we compute

$$\begin{aligned}\mathsf{succ}_\mathbb{N}(m) + n &= \mathsf{succ}_\mathbb{N}(m + n)\\ &= \mathsf{succ}_\mathbb{N}(n + m)\\ &\doteq n + \mathsf{succ}_\mathbb{N}(m).\end{aligned}$$

The first identification is obtained by Proposition 5.6.2, and the second identification is the identification $\mathsf{ap}_{\mathsf{succ}_\mathbb{N}}(p)$. □

Exercises

5.1 Show that the operation inverting identifications distributes over the concatenation operation, i.e., construct an identification

$$\mathsf{distributive\text{-}inv\text{-}concat}(p,q) : (p \cdot q)^{-1} = q^{-1} \cdot p^{-1}.$$

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

5.2 For any $p : x = y$, $q : y = z$, and $r : x = z$, construct maps

$$\begin{aligned}\mathsf{inv\text{-}con}(p,q,r) &: (p \cdot q = r) \to (q = p^{-1} \cdot r)\\ \mathsf{con\text{-}inv}(p,q,r) &: (p \cdot q = r) \to (p = r \cdot q^{-1}).\end{aligned}$$

5.3 Let B be a type family over A, and consider an identification $p : a = x$ in A. Construct for any $b : B(a)$ an identification

$$\mathsf{lift}_B(p,b) : (a,b) = (x, \mathsf{tr}_B(p,b)).$$

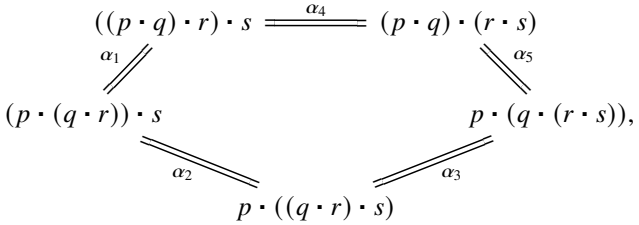
In other words, an identification $p : x = y$ in the *base type A lifts* to an identification in $\sum_{(x:A)} B(x)$ for every element in $B(x)$, analogous to the path lifting property for fibrations in homotopy theory.

5.4 Consider four consecutive identifications

$$a \overset{p}{=\!=} b \overset{q}{=\!=} c \overset{r}{=\!=} d \overset{s}{=\!=} e$$

in a type A. In this exercise we will show that the **Mac Lane pentagon** for identifications commutes.

(a) Construct the five identifications $\alpha_1, \ldots, \alpha_5$ in the pentagon



where α_1, α_2, and α_3 run counter-clockwise, and α_4 and α_5 run clockwise.

(b) Show that

$$(\alpha_1 \cdot \alpha_2) \cdot \alpha_3 = \alpha_4 \cdot \alpha_5.$$

5.5 In this exercise we show that the operations of addition and multiplication on the natural numbers satisfy the laws of a commutative **semi-ring**.

(a) Show that multiplication satisfies the following laws:

$$\begin{array}{lll} m \cdot 0 = 0 & m \cdot 1 = m & m \cdot \mathsf{succ}_{\mathbb{N}}(n) = m + m \cdot n \\ 0 \cdot m = 0 & 1 \cdot m = m & \mathsf{succ}_{\mathbb{N}}(m) \cdot n = m \cdot n + n. \end{array}$$

(b) Show that multiplication on $\mathbb{N}$ is commutative:

$$m \cdot n = n \cdot m.$$

(c) Show that multiplication on $\mathbb{N}$ distributes over addition from the left and from the right, i.e., show that we have identifications

$$\begin{aligned} m \cdot (n + k) &= m \cdot n + m \cdot k \\ (m + n) \cdot k &= m \cdot k + n \cdot k. \end{aligned}$$

(d) Show that multiplication on $\mathbb{N}$ is associative:

$$(m \cdot n) \cdot k = m \cdot (n \cdot k).$$

5.6 Show that

$$\mathsf{succ}_{\mathbb{Z}}(\mathsf{pred}_{\mathbb{Z}}(k)) = k \qquad \text{and} \qquad \mathsf{pred}_{\mathbb{Z}}(\mathsf{succ}_{\mathbb{Z}}(k)) = k$$

for any $k : \mathbb{Z}$, where $\mathsf{pred}_{\mathbb{Z}}$ is the predecessor function on the integers, defined in Exercise 4.1 (a).

5.7 In this exercise we will show that the laws for abelian groups hold for addition on the integers, using the group operations on $\mathbb{Z}$ defined in Exercise 4.1 (b).

(a) Show that addition satisfies the left and right unit laws, i.e., show that

$$0 + x = x$$
$$x + 0 = x.$$

(b) Show that the following successor and predecessor laws hold for addition on $\mathbb{Z}$.

$$\mathsf{pred}_{\mathbb{Z}}(x) + y = \mathsf{pred}_{\mathbb{Z}}(x + y) \qquad \mathsf{succ}_{\mathbb{Z}}(x) + y = \mathsf{succ}_{\mathbb{Z}}(x + y)$$
$$x + \mathsf{pred}_{\mathbb{Z}}(y) = \mathsf{pred}_{\mathbb{Z}}(x + y) \qquad x + \mathsf{succ}_{\mathbb{Z}}(y) = \mathsf{succ}_{\mathbb{Z}}(x + y).$$

(c) Use part (b) to show that addition on the integers is associative and commutative, show that

$$(x + y) + z = x + (y + z)$$
$$x + y = y + x.$$

(d) Show that addition satisfies the left and right inverse laws:

$$(-x) + x = 0$$
$$x + (-x) = 0.$$

5.8 In this exercise we will show that $\mathbb{Z}$ satisfies the axioms of a **ring**, using the multiplication operation defined in Exercise 4.1 (c).

(a) Show that multiplication on $\mathbb{Z}$ satisfies the following laws for 0 and 1:

$$0 \cdot x = 0 \qquad 1 \cdot x = x$$
$$x \cdot 0 = 0 \qquad x \cdot 1 = x.$$

(b) Show that multiplication on $\mathbb{Z}$ satisfies the predecessor and successor laws:

$$\mathsf{pred}_{\mathbb{Z}}(x) \cdot y = x \cdot y - y \qquad \mathsf{succ}_{\mathbb{Z}}(x) \cdot y = x \cdot y + y$$
$$x \cdot \mathsf{pred}_{\mathbb{Z}}(y) = x \cdot y - x \qquad y \cdot \mathsf{succ}_{\mathbb{Z}}(y) = x \cdot y + x.$$

(c) Show that multiplication on $\mathbb{Z}$ distributes over addition, both from the left and from the right:

$$x \cdot (y + z) = x \cdot y + x \cdot z$$
$$(x + y) \cdot z = x \cdot z + y \cdot z.$$

There are 306 exercises.

Ljubljana, November 10, 2021

(d) Show that multiplication on Z is associative and commutative:

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$
$$x \cdot y = y \cdot x.$$

6 Universes

To complete our specification of dependent type theory, we introduce type theoretic *universes*. Universes can be thought of as types that consist of types. In reality, however, a universe consists of a type $\mathcal{U}$ equipped with a type family $\mathcal{T}$ over $\mathcal{U}$. For any $X : \mathcal{U}$ we think of X as an *encoding* of the type $\mathcal{T}(X)$. The type family $\mathcal{T}$ is called a *universal type family*.

One of the aspects that make universes useful is that they are postulated to be closed under all the type constructors. For example, if we are given $X : \mathcal{U}$ and $P : \mathcal{T}(X) \to \mathcal{U}$, then the universe is equipped with an element $\check{\Sigma}(X, P) : \mathcal{U}$ satisfying the judgmental equality

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

This judgmental equality asserts that the element $\check{\Sigma}(X, P)$ of the universe $\mathcal{U}$ *represents* the Σ-type $\sum_{(x:\mathcal{T}(X))} \mathcal{T}(P(x))$.

We will similarly assume that any universe is closed under Π-types and the other ways of forming types. However, there is an important restriction: it would be inconsistent to assume that the universe is contained in itself. One way of thinking about this is that universes are types of *small* types, and it cannot be the case that the universe is small with respect to itself. We address this problem by assuming that there are many universes: enough universes so that any type family can be obtained by substituting into the universal type family of some universe.

There are several reasons to equip type theory with universes. One important reason is that it enables us to define new type families over inductive types, using their induction principle. We use this way of defining type families to define many familiar relations over $\mathbb{N}$, such as the ordering relations $\leq$ and $<$. We also introduce a relation $\mathsf{Eq}_{\mathbb{N}}$ called the *observational equality* on $\mathbb{N}$. This equivalence relation can be used to show that $0_{\mathbb{N}} \neq 1_{\mathbb{N}}$.

The idea of introducing an observational equality relation for a particular type is that it should help us thinking about the identity type. The identity type has been introduced in a very generic and uniform way. In specific cases, however, we have a clear idea of what the equality relation *should be*. In the case of the natural numbers, for instance, we will use the observational equality $\mathsf{Eq}_{\mathbb{N}}$ to

characterize the identity type of $\mathbb{N}$. Characterizing identity types is one of the main themes in homotopy type theory.

A second reason to introduce universes is that it allows us to define many types of types equipped with structure. One of the most important examples is the type of groups, which is the type of types equipped with the group operations satisfying the group laws, and for which the underlying type is a set. We won't discuss the condition for a type to be a set until Section 12, so the definition of groups in type theory will be given much later.

6.1 Specification of type theoretic universes

Definition 6.1.1 A **universe** in type theory is a type $\mathcal{U}$ in the empty context, equipped with a type family $\mathcal{T}$ over $\mathcal{U}$ called a **universal family**, that is closed under the type forming operations in the sense that it comes equipped with the following structure:

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

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

for which the judgmental equality

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

holds, for every $X : \mathcal{U}$ and $P : \mathcal{T}(X) \to \mathcal{U}$.

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

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

for which the judgmental equality

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

holds, for every $X : \mathcal{U}$ and $P : \mathcal{T}(X) \to \mathcal{U}$.

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

$$\check{\mathrm{I}} : \prod_{(X:\mathcal{U})} \mathcal{T}(X) \to (\mathcal{T}(X) \to \mathcal{U})$$

for which the judgmental equality

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

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

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

$$\check{+} : \mathcal{U} \to (\mathcal{U} \to \mathcal{U})$$

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

(v) $\mathcal{U}$ contains elements $\check{\emptyset}, \check{\mathbf{1}}, \check{\mathbb{N}} : \mathcal{U}$ that satisfy the judgmental equalities

$$\mathcal{T}(\check{\emptyset}) \doteq \emptyset$$
$$\mathcal{T}(\check{\mathbf{1}}) \doteq \mathbf{1}$$
$$\mathcal{T}(\check{\mathbb{N}}) \doteq \mathbb{N}.$$

Consider a universe $\mathcal{U}$ and a type A. We say that A is a type in $\mathcal{U}$, or that $\mathcal{U}$ **contains** A, if $\mathcal{U}$ comes equipped with an element $\check{A} : \mathcal{U}$ in context Γ, for which the judgment

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

holds. If A is a type in $\mathcal{U}$, we usually write simply A for $\check{A}$ and also A for $\mathcal{T}(\check{A})$.

Remark 6.1.2 Since ordinary function types are defined as a special case of dependent function types, we don't have to assume separately that universes are closed under ordinary function types. Similarly, it follows from the assumption that universes are closed under dependent pair types that universes are closed under cartesian product types.

6.2 Assuming enough universes

Most of the time we will get by with assuming one universe $\mathcal{U}$, and indeed we recommend on a first reading of this text to simply assume that there is one universe $\mathcal{U}$. However, sometimes we might want to consider the universe $\mathcal{U}$ itself to be a type in some universe. In such situations we cannot get by with a single universe, because the assumption that $\mathcal{U}$ is a element of itself would lead to inconsistencies like the Russell's paradox.

Russell's paradox is the famous argument that there cannot be a set of all sets. If there were such a set S, then we could consider Russell's subset

$$R := \{x \in S \mid x \notin x\}.$$

Russell then observed that $R \in R$ if and only if $R \notin R$, so we reach a contradiction. A variant of this argument reaches a similar contradiction when we assume that $\mathcal{U}$ is a universe that contains an element $\check{\mathcal{U}} : \mathcal{U}$ such that $\mathcal{T}\big(\check{\mathcal{U}}\big) \doteq \mathcal{U}$. In order to avoid such paradoxes, Russell and Whitehead formulated the *ramified theory*

of types in their book *Principia Mathematica*. The ramified theory of types is a precursor of Martin Löf's type theory that we are studying in this book.

Even though the universe is not an element of itself, it is still convenient if every type, including any universe, is in *some* universe. Therefore we will assume that there are sufficiently many universes:

Postulate 6.2.1 We assume that there are **enough universes**, i.e., that for every finite list of types in context

$$\Gamma_1 \vdash A_1 \text{ type} \qquad \cdots \qquad \Gamma_n \vdash A_n \text{ type},$$

there is a universe $\mathcal{U}$ that contains each A_i in the sense that $\mathcal{U}$ comes equipped with

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

for which the judgment

$$\Gamma_i \vdash \mathcal{T}\big(\check{A}_i\big) \doteq A_i \text{ type}$$

holds.

With this assumption it will rarely be necessary to work with more than one universe at the same time. Using the assumption that for any finite list of types in context there is a universe that contains those types, we obtain many specific universes.

Definition 6.2.2 The **base universe** $\mathcal{U}_0$ is the universe that we obtain using Postulate 6.2.1 with the empty list of types in context.

In other words, the base universe is a universe that is closed under all the ways of forming types, but it isn't specified to contain any further types.

Definition 6.2.3 The **successor universe** of a universe $\mathcal{U}$ is the universe $\mathcal{U}^+$ obtained using Postulate 6.2.1 with the finite list

$$\vdash \mathcal{U} \text{ type}$$
$$X : \mathcal{U} \vdash \mathcal{T}(X) \text{ type}.$$

Remark 6.2.4 The successor universe $\mathcal{U}^+$ of $\mathcal{U}$ therefore contains the type $\mathcal{U}$ as well as every type in $\mathcal{U}$, in the following sense

$$\vdash \check{\mathcal{U}} : \mathcal{U}^+ \qquad \vdash \mathcal{T}^+(\check{\mathcal{U}}) \doteq \mathcal{U} \text{ type}$$
$$X : \mathcal{U} \vdash \check{\mathcal{T}}(X) : \mathcal{U}^+ \qquad X : \mathcal{U} \vdash \mathcal{T}^+(\check{\mathcal{T}}(X)) \doteq \mathcal{T}(X) \text{ type}.$$

In particular, we obtain a function $i : \mathcal{U} \to \mathcal{U}^+$ that includes the types in $\mathcal{U}$ into $\mathcal{U}^+$, given by

$$i :\equiv \lambda X.\, \check{\mathcal{T}}(X).$$

Using successor universes we can create an infinite tower

$$\mathcal{U},\ \mathcal{U}^+,\ \mathcal{U}^{++},\ \ldots$$

of universes, starting at any universe $\mathcal{U}$, in which each universe is contained in the next. However, such towers of universes need not be exhaustive in the sense that it might not be the case that every type is contained in a universe in this tower.

Definition 6.2.5 The **join** of two universes $\mathcal{U}$ and $\mathcal{V}$ is the universe $\mathcal{U} \sqcup \mathcal{V}$ that we obtain using Postulate 6.2.1 with the two types

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

Remark 6.2.6 Since the join $\mathcal{U} \sqcup \mathcal{V}$ contains all the types in $\mathcal{U}$ and $\mathcal{V}$, there are maps

$$i : \mathcal{U} \to \mathcal{U} \sqcup \mathcal{V}$$
$$j : \mathcal{V} \to \mathcal{U} \sqcup \mathcal{V}$$

Note that we don't postulate any relations between the universes. In general it will therefore be the case that the universes $(\mathcal{U} \sqcup \mathcal{V}) \sqcup \mathcal{W}$ and $\mathcal{U} \sqcup (\mathcal{V} \sqcup \mathcal{W})$ will be unrelated.

6.3 Observational equality of the natural numbers

Using universes, we can define many relations on the natural numbers. We give here the example of *observational equality* of $\mathbb{N}$. The idea of observational equality is that, if we want to prove that m and n are observationally equal, we may do so by looking at m and n:

(i) If both m and n are $0_{\mathbb{N}}$, then they are observationally equal.
(ii) If one of them is $0_{\mathbb{N}}$ and the other is a successor, then they are not observationally equal.
(iii) If both m and n are successors, say $m \doteq \mathsf{succ}_{\mathbb{N}}(m')$ and $n \doteq \mathsf{succ}_{\mathbb{N}}(n')$, then m and n are observationally equal if and only if their predecessors m' and n' are observationally equal.

Thus, observational equality is an inductively defined relation, which gives us an algorithm for checking equality on $\mathbb{N}$. Indeed, it can be used to show that equality of natural numbers is *decidable*, i.e., there is a program that decides for any two natural numbers m and n whether they are equal or not.

Definition 6.3.1 We define the **observational equality** of $\mathbb{N}$ as binary relation $\mathsf{Eq}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathcal{U}_0)$ satisfying

$$\mathsf{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) \doteq \mathbf{1} \qquad \mathsf{Eq}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n), 0_{\mathbb{N}}) \doteq \emptyset$$
$$\mathsf{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, \mathsf{succ}_{\mathbb{N}}(n)) \doteq \emptyset \qquad \mathsf{Eq}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n), \mathsf{succ}_{\mathbb{N}}(m)) \doteq \mathsf{Eq}_{\mathbb{N}}(n, m).$$

Construction We define $\mathsf{Eq}_{\mathbb{N}}$ by double induction on $\mathbb{N}$. By the first application of induction it suffices to provide

$$E_0 : \mathbb{N} \to \mathcal{U}_0$$
$$E_S : \mathbb{N} \to ((\mathbb{N} \to \mathcal{U}_0) \to (\mathbb{N} \to \mathcal{U}_0))$$

We define E_0 by induction, taking $E_{00} :\equiv \mathbf{1}$ and $E_{0S}(n, X, m) :\equiv \emptyset$. The resulting family E_0 satisfies

$$E_0(0_{\mathbb{N}}) \doteq \mathbf{1}$$
$$E_0(\mathsf{succ}_{\mathbb{N}}(n)) \doteq \emptyset.$$

We define E_S by induction, taking $E_{S0} :\equiv \emptyset$ and $E_{SS}(n, X, m) :\equiv X(m)$. The resulting family E_S satisfies

$$E_S(n, X, 0_{\mathbb{N}}) \doteq \emptyset$$
$$E_S(n, X, \mathsf{succ}_{\mathbb{N}}(m)) \doteq X(m)$$

Therefore we have by the computation rule for the first induction that the judgmental equality

$$\mathsf{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, m) \doteq E_0(m)$$
$$\mathsf{Eq}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n), m) \doteq E_S(n, \mathsf{Eq}_{\mathbb{N}}(n), m)$$

holds, from which the judgmental equalities in the statement of the definition follow. □

The observational equality of the natural numbers is important because it can be used to prove equalities and negations of equalities. Proposition 6.3.3 enables us to do so.

Lemma 6.3.2 *Observational equality of $\mathbb{N}$ is a reflexive relation, i.e., we have*

$$\mathsf{refl}\text{-}\mathsf{Eq}_{\mathbb{N}} : \textstyle\prod_{(n:\mathbb{N})} \mathsf{Eq}_{\mathbb{N}}(n, n).$$

Proof The function $\mathsf{refl\text{-}Eq}_{\mathbb{N}}$ is defined by induction on n, taking

$$\mathsf{refl\text{-}Eq}_{\mathbb{N}}(0_{\mathbb{N}}) :\equiv \star$$
$$\mathsf{refl\text{-}Eq}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n)) :\equiv \mathsf{refl\text{-}Eq}_{\mathbb{N}}(n).$$

□

Proposition 6.3.3 *For any two natural numbers m and n, we have*

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

Proof The function $(m = n) \to \mathsf{Eq}_{\mathbb{N}}(m, n)$ is defined by the induction principle of identity types, using the reflexivity of $\mathsf{Eq}_{\mathbb{N}}$.

The converse $\mathsf{Eq}_{\mathbb{N}}(m, n) \to (m = n)$ is defined by induction on m and n. If both m and n are zero, we have $\mathsf{refl}_{0_{\mathbb{N}}} : 0_{\mathbb{N}} = 0_{\mathbb{N}}$. If one of m and n is zero and the other is a successor, then $\mathsf{Eq}_{\mathbb{N}}(m, n)$ is empty and we have a function $\emptyset \to (m = n)$ by the induction principle of the empty type. In the inductive step, suppose we have a function $f : \mathsf{Eq}_{\mathbb{N}}(m, n) \to (m = n)$. Then we can define a function

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

as the composite

$$\begin{array}{ccc} \mathsf{Eq}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m), \mathsf{succ}_{\mathbb{N}}(n)) & \dashrightarrow & (\mathsf{succ}_{\mathbb{N}}(m) = \mathsf{succ}_{\mathbb{N}}(n)). \\ {\scriptstyle \mathsf{id}}\downarrow & & \uparrow{\scriptstyle \mathsf{ap}_{\mathsf{succ}_{\mathbb{N}}}} \\ \mathsf{Eq}_{\mathbb{N}}(m, n) & \xrightarrow[f]{} & (m = n) \end{array}$$

Note that the map on the left is the identity function, because we have the judgmental equality $\mathsf{Eq}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m), \mathsf{succ}_{\mathbb{N}}(n)) \doteq \mathsf{Eq}_{\mathbb{N}}(m, n)$ by definition of $\mathsf{Eq}_{\mathbb{N}}$. □

6.4 Peano's seventh and eighth axioms

Using the observational equality of $\mathbb{N}$, we can prove Peano's seventh and eighth axioms. In his *Arithmetices Principia* [4], the natural numbers are based at 1, but today it is customary to have the natural numbers based at 0. Adapting for this, the seventh and eighth axioms assert that

(P7) For any two natural numbers m and n, we have

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

(P8) For any natural number n, we have $0_{\mathbb{N}} \neq \mathsf{succ}_{\mathbb{N}}(n)$.

Theorem 6.4.1 *For any two natural numbers m and n, we have*

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

Proof The forward implication is given by the action on paths of the successor function

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

The direction of interest is the converse, which asserts that the successor function is injective.

Here we use Proposition 6.3.3, which asserts that $(m = n) \leftrightarrow \mathsf{Eq}_\mathbb{N}(m, n)$ for all $m, n : \mathbb{N}$. Furthermore, we have $\mathsf{Eq}_\mathbb{N}(\mathsf{succ}_\mathbb{N}(m), \mathsf{succ}_\mathbb{N}(n)) \doteq \mathsf{Eq}_\mathbb{N}(m, n)$. Therefore, we obtain

$$\begin{array}{ccc} (\mathsf{succ}_\mathbb{N}(m) = \mathsf{succ}_\mathbb{N}(n)) & \dashrightarrow & (m = n) \\ \downarrow & & \uparrow \\ \mathsf{Eq}_\mathbb{N}(\mathsf{succ}_\mathbb{N}(m), \mathsf{succ}_\mathbb{N}(n)) & \xrightarrow[\mathsf{id}]{} & \mathsf{Eq}_\mathbb{N}(m, n), \end{array}$$

and we define the function $(\mathsf{succ}_\mathbb{N}(m) = \mathsf{succ}_\mathbb{N}(n)) \to (m = n)$ as the composite of the maps going down, then right, and then up. □

Theorem 6.4.2 *For any natural number n, we have* $0_\mathbb{N} \neq \mathsf{succ}_\mathbb{N}(n)$.

Proof By Proposition 6.3.3 it follows that there is a family of maps

$$(0_\mathbb{N} = n) \to \mathsf{Eq}_\mathbb{N}(0_\mathbb{N}, n).$$

indexed by $n : \mathbb{N}$. Since $\mathsf{Eq}_\mathbb{N}(0_\mathbb{N}, \mathsf{succ}_\mathbb{N}(n)) \doteq \emptyset$ it follows that

$$(0_\mathbb{N} = \mathsf{succ}_\mathbb{N}(n)) \to \emptyset,$$

which is precisely the claim. □

Exercises

6.1 (a) Show that

$$(m = n) \leftrightarrow (m + k = n + k)$$
$$(m = n) \leftrightarrow (m \cdot (k + 1) = n \cdot (k + 1))$$

for all $m, n, k : \mathbb{N}$. In other words, adding k and multiplying by $k + 1$ are injective functions.

(b) Show that

$$(m + n = 0) \leftrightarrow (m = 0) \times (n = 0)$$
$$(mn = 0) \leftrightarrow (m = 0) + (n = 0)$$
$$(mn = 1) \leftrightarrow (m = 1) \times (n = 1)$$

for all $m, n : \mathbb{N}$.

(c) Show that

$$m \neq m + (n + 1)$$
$$m + 1 \neq (m + 1)(n + 2)$$

for all $m, n : \mathbb{N}$.

6.2 (a) Define observational equality $\mathsf{Eq\text{-}bool}$ by induction on the booleans.

(b) Show that

$$(x = y) \leftrightarrow \mathsf{Eq\text{-}bool}(x, y)$$

for any $x, y : \mathsf{bool}$.

(c) Show that $b \neq \mathsf{neg\text{-}bool}(b)$ for any $b : \mathsf{bool}$. Conclude that $\mathsf{false} \neq \mathsf{true}$.

6.3 The ordering relation $\leq$ on $\mathbb{N}$ is defined recursively by

$$(0_{\mathbb{N}} \leq 0_{\mathbb{N}}) :\equiv \mathbf{1} \qquad (0_{\mathbb{N}} \leq n + 1) :\equiv \mathbf{1}$$
$$(m + 1 \leq 0_{\mathbb{N}}) :\equiv \emptyset \qquad (m + 1 \leq n + 1) :\equiv (m \leq n).$$

(a) Show that $\leq$ satisfies the axioms of a *poset*, i.e., show that $\leq$ is

(i) reflexive,
(ii) antisymmetric, and
(iii) transitive.

(b) Show that

$$(m \leq n) + (n \leq m)$$

for any $m, n : \mathbb{N}$.

(c) Show that

$$(m \leq n) \leftrightarrow (m + k \leq n + k)$$

holds for any $m, n, k : \mathbb{N}$.

(d) Show that

$$(m \leq n) \leftrightarrow (m \cdot (k + 1) \leq n \cdot (k + 1))$$

holds for any $m, n, k : \mathbb{N}$.

(e) Show that $k \leq \mathsf{min}_{\mathbb{N}}(m, n)$ holds if and only if both $k \leq m$ and $k \leq n$ hold, and show that $\mathsf{max}_{\mathbb{N}}(m, n) \leq k$ holds if and only if both $m \leq k$ and $n \leq k$ hold.

6.4 The strict ordering relation $<$ on $\mathbb{N}$ is defined recursively by

$$(0_{\mathbb{N}} < 0_{\mathbb{N}}) :\equiv \emptyset \qquad (0_{\mathbb{N}} < n + 1) :\equiv \mathbf{1}$$
$$(m + 1 < 0_{\mathbb{N}}) :\equiv \emptyset \qquad (m + 1 < n + 1) :\equiv (m < n).$$

(a) Show that the strict ordering relation is

(i) antireflexive,
(ii) antisymmetric, and
(iii) transitive.

(b) Show that $n < n + 1$ and

$$(m < n) \to (m < n + 1)$$

for any $m, n : \mathbb{N}$.

(c) Show that

$$(m < n) \leftrightarrow (m + 1 \leq n)$$
$$(m < n) \leftrightarrow (n \not\leq m)$$

for any $m, n : \mathbb{N}$.

6.5 The **distance function**

$$\mathsf{dist}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$$

is defined recursively by

$$\mathsf{dist}_{\mathbb{N}}(0, 0) :\equiv 0 \qquad \mathsf{dist}_{\mathbb{N}}(0, n + 1) :\equiv n + 1$$
$$\mathsf{dist}_{\mathbb{N}}(m + 1, 0) :\equiv m + 1 \qquad \mathsf{dist}_{\mathbb{N}}(m + 1, n + 1) :\equiv \mathsf{dist}_{\mathbb{N}}(m, n).$$

In other words, the distance between two natural numbers is the *symmetric difference* between them.

(a) Show that $\mathsf{dist}_{\mathbb{N}}$ satisfies the axioms of a metric:

(i) $(m = n) \leftrightarrow (\mathsf{dist}_{\mathbb{N}}(m, n) = 0)$,
(ii) $\mathsf{dist}_{\mathbb{N}}(m, n) = \mathsf{dist}_{\mathbb{N}}(n, m)$,
(iii) $\mathsf{dist}_{\mathbb{N}}(m, n) \leq \mathsf{dist}_{\mathbb{N}}(m, k) + \mathsf{dist}_{\mathbb{N}}(k, n)$.

(b) Show that $\mathsf{dist}_{\mathbb{N}}(m, n) = \mathsf{dist}_{\mathbb{N}}(m, k) + \mathsf{dist}_{\mathbb{N}}(k, n)$ if and only if either both $m \leq k$ and $k \leq n$ hold or both $n \leq k$ and $k \leq m$ hold.

(c) Show that $\mathsf{dist}_{\mathbb{N}}$ is translation invariant and linear:

$$\mathsf{dist}_{\mathbb{N}}(a + m, a + n) = \mathsf{dist}_{\mathbb{N}}(m, n),$$
$$\mathsf{dist}_{\mathbb{N}}(k \cdot m, k \cdot n) = k \cdot \mathsf{dist}_{\mathbb{N}}(m, n).$$

(d) Show that $x + \mathsf{dist}_{\mathbb{N}}(x, y) = y$ for any $x \leq y$.

6.6 Construct the absolute value function

$$|-| : \mathbb{Z} \to \mathbb{N}$$

and show that it satisfies the following three properties:

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

(i) $(x = 0) \leftrightarrow (|x| = 0)$,
(ii) $|x + y| \leq |x| + |y|$,
(iii) $|xy| = |x||y|$.

7 Modular arithmetic via the Curry-Howard interpretation

We have now fully described Martin-Löf's dependent type theory. It is now up to us to start developing some mathematics in it, and Martin-Löf's dependent type theory is great for elementary mathematics, such as basic number theory, some algebra, and combinatorics. The fundamental idea that is used to develop basic mathematics in type theory is the Curry-Howard interpretation. This is a translation of logic into type theory, which we will use to express concepts of mathematics such as divisibility, the congruence relations, and so on.

We will also introduce the family Fin of the standard finite types, indexed by $\mathbb{N}$, and show how each Fin_{k+1} can be equipped with the group structure of integers modulo $k + 1$. Our goal here is to demonstrate how to do those things in type theory, so we will aim for a high degree of accuracy.

7.1 The Curry-Howard interpretation

The *Curry-Howard interpretation* is an interpretation of logic into type theory. Recall that 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. These two aspects of the foundations of mathematics are unified in type theory. The idea of the Curry-Howard interpretation is therefore to express propositions as types, and to think of the elements of those types as their proofs. We illustrate this idea with an example.

Example 7.1.1 A natural number d is said to divide a natural number n if there exists a natural number k such that $d \cdot k = n$. To represent the divisibility predicate in type theory, we need to define a *type*

$$d \mid n,$$

of which the elements are witnesses that d divides n. In other words, $d \mid n$ should be the type that consists of natural numbers k equipped with an identification $d \cdot k = n$. In general, the type of $x : A$ equipped with $y : B(x)$ is represented as the type $\sum_{(x:A)} B(x)$. The interpretation of the existential quantification $(\exists)$ into type theory via the Curry-Howard interpretation is therefore using Σ-types.

Definition 7.1.2 Consider two natural numbers d and n. We say that d **divides** n if there is a element of type

$$d \mid n :\equiv \sum_{(k:\mathbb{N})} d \cdot k = n.$$

Remark 7.1.3 This type-theoretical definition of the divisibility relation using Σ-types has two important consequences:

(i) The principal way to show that $d \mid n$ holds is to construct a pair (k, p) consisting of a natural number k and an identification $p : d \cdot k = n$.
(ii) The principal way to use a hypothesis $H : d \mid n$ in a proof is to proceed by Σ-induction on the variable H. We then get to assume a natural number k and an identification $p : d \cdot k = n$, in order to proceed with the proof.

Example 7.1.4 Just as existential quantification ($\exists$) is translated via the Curry-Howard interpretation to Σ-types, the translation of the universal quantification ($\forall$) in type theory via the Curry-Howard interpretation is to Π-types. For example, the assertion that every natural number is divisible by 1 is expressed in type theory as

$$\prod_{(x:\mathbb{N})} 1 \mid x.$$

In other words, in order to show that every number $x : \mathbb{N}$ is divisible by 1 we need to construct a dependent function

$$\lambda x.\, p(x) : \prod_{(x:\mathbb{N})} 1 \mid x.$$

We do this by constructing an element

$$p(x) : \sum_{(k:\mathbb{N})} 1 \cdot k = x$$

indexed by $x : \mathbb{N}$. Such an element $p(x)$ is constructed as the pair $(x, q(x))$, where the identification $q(x) : 1 \cdot x = x$ is obtained from the left unit law of multiplication on $\mathbb{N}$, which was constructed in Exercise 5.5.

Similarly, the type theoretic proof that every natural number k divides 0, i.e., that $k \mid 0$, is the pair $(0, p)$ consisting of the natural number 0 and the identification $p : k \cdot 0 = 0$ obtained from the right annihilation law of multiplication on $\mathbb{N}$. This identification was also constructed in Exercise 5.5.

In the following proposition we will see examples of how a hypothesis of type $d \mid x$ can be used.

Proposition 7.1.5 *Consider three natural numbers d, x and y. If d divides any two of the three numbers x, y, and $x + y$, then it also divides the third.*

Proof We will only show that if d divides x and y, then it divides $x + y$. The remaining two claims, that if d divides y and $x + y$ then it divides x, and that if d divides x and $x + y$ then it divides y, are left as Exercise 7.1.

Suppose that d divides both x and y. By assumption we have elements

$$H : \sum_{(k:\mathbb{N})} d \cdot k = x, \qquad \text{and} \qquad K : \sum_{(k:\mathbb{N})} d \cdot k = y.$$

Since the types of the variables H and K are Σ-types, we proceed by Σ-induction on H and K. Therefore we get to assume a natural number $k : \mathbb{N}$ equipped with an identification $p : d \cdot k = x$, and a natural number $l : \mathbb{N}$ equipped with an identification $q : d \cdot l = y$. Our goal is now to construct an identification

$$d \cdot (k + l) = x + y.$$

We construct such an identification as a concatenation $\alpha \cdot (\beta \cdot \gamma)$, where the types of the identifications α, β, and γ are as follows:

$$d \cdot (k + l) \overset{\alpha}{=\!=} d \cdot k + d \cdot l \overset{\beta}{=\!=} x + d \cdot l \overset{\gamma}{=\!=} x + y.$$

The identification α is obtained from the fact that multiplication on $\mathbb{N}$ distributes over addition, which was shown in Exercise 5.5 (c). The identifications β and γ are constructed using the action on paths of a function:

$$\beta \coloneqq \mathsf{ap}_{(\lambda t.\, t + d \cdot l)}(p), \qquad \text{and} \qquad \gamma \coloneqq \mathsf{ap}_{(\lambda t.\, x + t)}(q)$$

To conclude the proof that $d \mid x + y$, note that we have constructed the pair

$$(k + l, \alpha \cdot (\beta \cdot \gamma)) : \sum_{(k:\mathbb{N})} d \cdot k = x + y. \qquad \square$$

The full Curry-Howard interpretation of logic into type theory also involves interpretations of disjunction, conjunction, implication, and equality.

The introduction and elimination rules for disjunction are, for instance,

$$\frac{P}{P \vee Q} \qquad \frac{Q}{P \vee Q} \qquad \text{and} \qquad \frac{P \Rightarrow R \quad Q \Rightarrow R}{P \vee Q \Rightarrow R}$$

The two introduction rules assert that $P \vee Q$ holds provided that P holds, and that $P \vee Q$ holds provided that Q holds. These rules are analogous to the introduction rules for coproduct, which assert that there are functions $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$. Furthermore, the non-dependent elimination principle for coproducts gives a function

$$(A \to C) \to ((B \to C) \to (A + B \to C))$$

for any type C, which is again analogous to the elimination rule of disjunction. The Curry-Howard interpretation of disjunction into type theory is therefore as coproducts.

The Curry-Howard interpretation	
Propositions	Types
Proofs	Elements
Predicates	Type families
$\top$	$\mathbf{1}$
$\bot$	$\emptyset$
$P \vee Q$	$A + B$
$P \wedge Q$	$A \times B$
$P \Rightarrow Q$	$A \to B$
$\neg P$	$A \to \emptyset$
$\exists_x P(x)$	$\sum_{(x:A)} B(x)$
$\forall_x P(x)$	$\prod_{(x:A)} B(x)$
$x = y$	$x = y$

Table I.2 *The Curry-Howard interpretation of logic into type theory.*

To interpret conjunction into type theory we observe that the introduction rule and elimination rules for conjunction are

$$\frac{P \quad Q}{P \wedge Q} \qquad \text{and} \qquad \frac{P \wedge Q}{P} \qquad \frac{P \wedge Q}{Q}$$

Product types possess such structure, where we have the pairing operation $\mathsf{pair} : A \to (B \to A \times B)$ and the projections $\mathsf{pr}_1 : A \times B \to A$ and $\mathsf{pr}_2 : A \times B \to B$ give interpretations of the introduction and elimination rules for conjunction. The Curry-Howard interpretation of conjunction into type theory is therefore by products. We summarize the full Curry-Howard interpretation in Table I.2.

Remark 7.1.6 We should note, however, that despite the similarities between logic and type theory that are highlighted in the Curry-Howard interpretation, there are also some differences. One important difference is that types may contain many elements, whereas in logic, propositions are usually considered to be *proof irrelevant*. This means that to establish the truth of a proposition it only matters *whether* it can be proven, not in how many different ways it can be proven. To address this dissimilarity between general types and logic, we will introduce in Chapter II a more refined way of interpreting logic into type theory. In Section 12 we will define the type $\mathsf{is\text{-}prop}(A)$, which expresses the property that the type A is a proposition. Furthermore, we will introduce the *propositional truncation* operation in Section 14, which we will use to interpret logic into type theory in such a way that all logical assertions are interpreted as types that satisfy the condition of being a proposition.

7.2 The congruence relations on $\mathbb{N}$

Relations in the Curry-Howard interpretation of logic into type theory are also type valued. More specifically, a binary relation on a type A is a family of types $R(x, y)$ indexed by $x, y : A$. Such relations are sometimes called *typal*.

Definition 7.2.1 Consider a type A. A **(typal) binary relation** on A is defined to be a family of types $R(x, y)$ indexed by $x, y : A$. Given a binary relation R on A, we say that R is **reflexive** if it comes equipped with

$$\rho : \prod_{(x:A)} R(x, x),$$

we say that R is **symmetric** if it comes equipped with

$$\sigma : \prod_{(x,y:A)} R(x, y) \to R(y, x),$$

and we say that R is **transitive** if it comes equipped with

$$\tau : \prod_{(x,y,z:A)} R(x, y) \to (R(y, z) \to R(x, z)).$$

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

To define the congruence relation modulo k in type theory using the Curry-Howard interpretation, we will define for any three natural numbers x, y, and k, a *type*

$$x \equiv y \mod k$$

consisting of the proofs that x is congruent to y modulo k. We will define this type by directly interpreting Gauss' definition of the congruence relations in his *Disquisitiones Arithmeticae* [3]: two numbers x and y are congruent modulo k if k divides the symmetric difference $\mathsf{dist}_{\mathbb{N}}(x, y)$ between x and y. Recall that $\mathsf{dist}_{\mathbb{N}}(x, y)$ was defined in Exercise 6.5 recursively by

$$\begin{aligned} \mathsf{dist}_{\mathbb{N}}(0, 0) &:\equiv 0 & \mathsf{dist}_{\mathbb{N}}(0, y + 1) &:\equiv y + 1 \\ \mathsf{dist}_{\mathbb{N}}(x + 1, 0) &:\equiv x + 1 & \mathsf{dist}_{\mathbb{N}}(x + 1, y + 1) &:\equiv \mathsf{dist}_{\mathbb{N}}(x, y). \end{aligned}$$

Definition 7.2.2 Consider three natural numbers $k, x, y : \mathbb{N}$. We say that x is **congruent to** y **modulo** k if it comes equipped with an element of type

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

Example 7.2.3 For example, $k \equiv 0 \mod k$. To see this, we have to show that $k \mid \mathsf{dist}_{\mathbb{N}}(k, 0)$. Since $\mathsf{dist}_{\mathbb{N}}(k, 0) = k$ it suffices to show that $k \mid k$. That is, we have to construct a natural number l equipped with an identification $p : kl = k$. Of course, we choose $l :\equiv 1$, and the equation $k1 = k$ holds by the right unit law for multiplication on $\mathbb{N}$, which was shown in Exercise 5.5.

Proposition 7.2.4 *For each $k : \mathbb{N}$, the congruence relation modulo k is an equivalence relation.*

Proof Reflexivity follows from the fact that $\mathsf{dist}_{\mathbb{N}}(x, x) = 0$, and any number divides 0. Symmetry follows from the fact that $\mathsf{dist}_{\mathbb{N}}(x, y) = \mathsf{dist}_{\mathbb{N}}(y, x)$ for any two natural numbers x and y.

The non-trivial part of the claim is therefore transitivity. Here we use the fact that for any three natural numbers x, y, and z, at least one of the equalities

$$\begin{aligned}\mathsf{dist}_{\mathbb{N}}(x, y) + \mathsf{dist}_{\mathbb{N}}(y, z) &= \mathsf{dist}_{\mathbb{N}}(x, z)\\ \mathsf{dist}_{\mathbb{N}}(y, z) + \mathsf{dist}_{\mathbb{N}}(x, z) &= \mathsf{dist}_{\mathbb{N}}(x, y)\\ \mathsf{dist}_{\mathbb{N}}(x, z) + \mathsf{dist}_{\mathbb{N}}(x, y) &= \mathsf{dist}_{\mathbb{N}}(y, z)\end{aligned}$$

holds. A formal proof of this fact is given by case analysis on the six possible ways in which x, y, and z can be ordered:

$$\begin{aligned} &x \leq y \text{ and } y \leq z, && x \leq z \text{ and } z \leq y,\\ &y \leq z \text{ and } z \leq x, && y \leq x \text{ and } x \leq z,\\ &z \leq x \text{ and } x \leq y, && z \leq y \text{ and } y \leq x.\end{aligned}$$

Therefore it follows by Exercise 6.5 (b) and Proposition 7.1.5 that $k \mid \mathsf{dist}_{\mathbb{N}}(x, z)$ if $k \mid \mathsf{dist}_{\mathbb{N}}(x, y)$ and $k \mid \mathsf{dist}_{\mathbb{N}}(y, z)$. □

7.3 The standard finite types

The standard finite sets are classically defined as the sets $\{x \in \mathbb{N} \mid x < k\}$. This leads to the question of how to interpret a subset $\{x \in A \mid P(x)\}$ in type theory.

Since type theory is set up in such a way that elements come equipped with their types, subsets aren't formed the same way as in set theory, where the comprehension axiom is used to form the set $\{x \in A \mid P(x)\}$ for any predicate P over A. The Curry-Howard interpretation dictates that predicates are interpreted as dependent types. Therefore, a set of elements $x \in A$ such that $P(x)$ holds is interpreted in type theory as the type of terms $x : A$ equipped with an element (a proof) $p : P(x)$. In other words, we interpret a subset $\{x \in A \mid P(x)\}$ as the type $\sum_{(x:A)} P(x)$.

Remark 7.3.1 The alert reader may now have observed that the interpretation of a subset $\{x \in A \mid P(x)\}$ in type theory is the same as the interpretation of the proposition $\exists_{(x \in A)} P(x)$, while indeed the subset $\{x \in A \mid P(x)\}$ has a substantially different role in mathematics than the proposition $\exists_{(x \in A)} P(x)$. This points at a slight problem of the Curry-Howard interpretation of the existential quantifier. While the Curry-Howard interpretation of the existential quantifier is

nevertheless useful and important, we will reinterpret the existential quantifier in type theory in Section 17.4.

Since subsets are interpreted as Σ-types, the 'classical' definition of the standard finite types is

$$\mathsf{classical\text{-}Fin}_k :\equiv \sum_{(x:\mathbb{N})} x < k.$$

This is a perfectly fine definition of the standard finite types. However, the usual definition of the standard finite types in Martin-Löf's dependent type theory is a more direct, recursive definition, which takes full advantage of the inductive constructions of dependent type theory.

Definition 7.3.2 We define the type family Fin of the **standard finite types** over $\mathbb{N}$ recursively by

$$\begin{aligned}\mathsf{Fin}_0 &:\equiv \emptyset\\ \mathsf{Fin}_{k+1} &:\equiv \mathsf{Fin}_k + \mathbf{1}.\end{aligned}$$

We will write i for the inclusion $\mathsf{inl} : \mathsf{Fin}_k \to \mathsf{Fin}_{k+1}$ and we will write $\star$ for the point $\mathsf{inr}(\star)$.

In Exercise 7.7 you will be asked to show that the types $\mathsf{classical\text{-}Fin}_k$ and Fin_k are isomorphic.

Remark 7.3.3 The type family Fin over $\mathbb{N}$ can be given its own induction principle, which is, at least for the time being, the principal way to make constructions on Fin_k for arbitrary $k : \mathbb{N}$ and to prove properties about those constructions. The induction principle of the standard finite types tells us that the family of standard finite types is inductively generated by

$$\begin{aligned}i &: \mathsf{Fin}_k \to \mathsf{Fin}_{k+1}\\ \star &: \mathsf{Fin}_{k+1}.\end{aligned}$$

In other words, we can define a dependent function $f : \prod_{(k:\mathbb{N})} \prod_{(x:\mathsf{Fin}_k)} P_k(x)$ by defining

$$\begin{aligned}g_k &: \prod_{(x:\mathsf{Fin}_k)} P_k(x) \to P_{k+1}(i(x))\\ p_k &: P_{k+1}(\star)\end{aligned}$$

for each $k : \mathbb{N}$. The function f defined in this way then satisfies the judgmental equalities

$$\begin{aligned}f_{k+1}(i(x)) &\doteq g_k(x, f_k(x))\\ f_{k+1}(\star) &\doteq p_k.\end{aligned}$$

These judgmental equalities completely determine the function f, and therefore we may also present such inductive definitions by pattern matching:

$$f_{k+1}(i(x)) :\equiv g_k(x, f_k(x))$$
$$f_{k+1}(\star) :\equiv p_k.$$

We will often use definitions by pattern matching for two reasons: (i) such definitions are concise, and (ii) they display the judgmental equalities that hold for the defined object. Those judgmental equalities are the only thing we know about that object, and proving a claim about it often amounts to finding a way to apply these judgmental equalities.

To illustrate this way of working with the standard finite types, we define the inclusion functions $\mathsf{Fin}_k \to \mathbb{N}$, and show that these are injective. In order to show that ι_k is injective, we will also show that ι_k is bounded.

Definition 7.3.4 We define the inclusion $\iota_k : \mathsf{Fin}_k \to \mathbb{N}$ inductively by

$$\iota_{k+1}(i(x)) :\equiv \iota_k(x)$$
$$\iota_{k+1}(\star) :\equiv k.$$

Lemma 7.3.5 *The function $\iota : \mathsf{Fin}_k \to \mathbb{N}$ is bounded, in the sense that $\iota(x) < k$ for each $x : \mathsf{Fin}_k$.*

Proof The proof is by induction. In the base case there is nothing to show. In the inductive step, we have the inequalities $\iota_{k+1}(i(x)) \doteq \iota_k(x) < k < k+1$, where the first inequality holds by the inductive hypothesis, and we also have

$$\iota_{k+1}(\star) \doteq k < k+1. \qquad \square$$

Proposition 7.3.6 *The inclusion function $\iota_k : \mathsf{Fin}_k \to \mathbb{N}$ is injective, for each $k : \mathbb{N}$.*

Proof We define a function $\alpha_k(x, y) : (\iota_k(x) = \iota_k(y)) \to (x = y)$ recursively by

$$\alpha_{k+1}(i(x), i(y), p) :\equiv \mathsf{ap}_i(\alpha_k(x, y, p)) \qquad \alpha_{k+1}(i(x), \star, p) :\equiv \mathsf{ex\text{-}falso}(f(p))$$
$$\alpha_{k+1}(\star, i(y), p) :\equiv \mathsf{ex\text{-}falso}(g(p)) \qquad \alpha_{k+1}(\star, \star, p) :\equiv \mathsf{refl},$$

where $f : (\iota_{k+1}(i(x)) = \iota_{k+1}(\star)) \to \emptyset$ and $g : (\iota_{k+1}(\star) = \iota_{k+1}(i(y))) \to \emptyset$ are obtained from the fact that $\iota_{k+1}(i(z)) \doteq \iota_k(z) < k$ for any $z : \mathsf{Fin}_k$, and the fact that $\iota_{k+1}(\star) \doteq k$. $\square$

7.4 The natural numbers modulo $k+1$

Given an equivalence relation $\sim$ on a set A in classical mathematics, the quotient $A/\sim$ comes equipped with a quotient map $q : A \to A/\sim$ that satisfies two important properties: (1) The map q satisfies the condition

$$q(x) = q(y) \leftrightarrow x \sim y,$$

and (2) the map q is surjective. The first condition is called the **effectiveness** of the quotient map.

In classical mathematics, a map $f : A \to B$ is said to be surjective if for every $b \in B$ there exists an element $a \in A$ such that $f(a) = b$. Following the Curry-Howard interpretation, a map $f : A \to B$ is therefore surjective if it comes equipped with a dependent function

$$\prod_{(b:B)} \sum_{(a:A)} f(a) = b.$$

However, there is a subtle issue with this interpretation of surjectivity. It is somewhat stronger than the classical notion of surjectivity, because a dependent function $\prod_{(b:B)} \sum_{(a:A)} f(a) = b$ provides for every element $b : B$ an *explicit* element $a : A$ equipped with an explicit identification $p : f(a) = b$, whereas in the classical notion of surjectivity such an element $a \in A$ is merely asserted to exist. To emphasize that the Curry-Howard interpretation of surjectivity is stronger than intended we make the following definition, and we will properly introduce surjective maps in Section 15.2.

Definition 7.4.1 Consider a function $f : A \to B$. We say that f is **split surjective** if it comes equipped with an element of type

$$\mathsf{is\text{-}split\text{-}surjective}(f) := \prod_{(b:B)} \sum_{(a:A)} f(a) = b.$$

Martin-Löf's dependent type theory doesn't have a general way of forming quotients of types. However, in the specific case of the congruence relations on $\mathbb{N}$ we can define the type of natural numbers modulo $k+1$ as the standard finite type Fin_{k+1}. We will show that Fin_{k+1} comes equipped with a map

$$[-]_{k+1} : \mathbb{N} \to \mathsf{Fin}_{k+1}$$

for each $k : \mathbb{N}$, and we will show in Theorems 7.4.7 and 7.4.8 that this map satisfies conditions (1) and (2) in the split surjective sense.

To prepare for the definition of the quotient map $[-]_{k+1}$, we will first define a zero element of Fin_{k+1} and successor function on each Fin_k. We will also define an auxiliary function $\mathsf{skip\text{-}zero}_k : \mathsf{Fin}_k \to \mathsf{Fin}_{k+1}$, which is used in the definition of the successor function. The map $[-]_{k+1}$ is then defined by iterating the successor function.

Definition 7.4.2

(i) We define the **zero element** $\mathsf{zero}_k : \mathsf{Fin}_{k+1}$ recursively by

$$\begin{aligned}\mathsf{zero}_0 &:\equiv \star\\ \mathsf{zero}_{k+1} &:\equiv i(\mathsf{zero}_k).\end{aligned}$$

Since there is a mismatch between the index of zero_k and the index of its type, we will often simply write zero or 0 for the zero element of Fin_{k+1}.

(ii) We define the function $\mathsf{skip\text{-}zero}_k : \mathsf{Fin}_k \to \mathsf{Fin}_{k+1}$ recursively by

$$\begin{aligned}\mathsf{skip\text{-}zero}_{k+1}(i(x)) &:\equiv i(\mathsf{skip\text{-}zero}_k(x))\\ \mathsf{skip\text{-}zero}_{k+1}(\star) &:\equiv \star.\end{aligned}$$

(iii) We define the **successor function** $\mathsf{succ}_k : \mathsf{Fin}_k \to \mathsf{Fin}_k$ recursively by

$$\begin{aligned}\mathsf{succ}_{k+1}(i(x)) &:\equiv \mathsf{skip\text{-}zero}_k(x)\\ \mathsf{succ}_{k+1}(\star) &:\equiv \mathsf{zero}_k.\end{aligned}$$

Definition 7.4.3 For any $k : \mathbb{N}$, we define the map $[-]_{k+1} : \mathbb{N} \to \mathsf{Fin}_{k+1}$ recursively on x by

$$\begin{aligned}[0]_{k+1} &:\equiv 0\\ [x+1]_{k+1} &:\equiv \mathsf{succ}_{k+1}[x]_{k+1}.\end{aligned}$$

Our next intermediate goal is to show that $x \equiv \iota[x]_{k+1} \mod k+1$ for any natural number x. This fact is a consequence of the following simple lemma, that will help us compute with the maps $\iota : \mathsf{Fin}_k \to \mathbb{N}$.

Lemma 7.4.4 *We make three claims:*

(i) *For any $k : \mathbb{N}$ there is an identification*

$$\iota(\mathsf{zero}_k) = 0$$

(ii) *For any $k : \mathbb{N}$ and any $x : \mathsf{Fin}_k$, we have*

$$\iota(\mathsf{skip\text{-}zero}_k(x)) = \iota(x) + 1.$$

(iii) *For any $k : \mathbb{N}$ and any $x : \mathsf{Fin}_k$, we have*

$$\iota(\mathsf{succ}_k(x)) \equiv \iota(x) + 1 \mod k.$$

Proof For the first claim, we define an identification $\alpha_k : \iota(\mathsf{zero}_k) = 0$ recursively by

$$\begin{aligned}\alpha_0 &:\equiv \mathsf{refl}\\ \alpha_{k+1} &:\equiv \alpha_k.\end{aligned}$$

For the second claim, we define an identification $\beta_k(x) : \iota(\mathsf{skip\text{-}zero}_k(x)) = \iota(x) + 1$ recursively by

$$\beta_{k+1}(i(x)) :\equiv \beta_k(x)$$
$$\beta_{k+1}(\star) :\equiv \mathsf{refl}.$$

For the third claim, we again define an element $\gamma_k(x) : \iota(\mathsf{succ}_k(x)) \equiv \iota(x) + 1 \mod k$ recursively. To obtain

$$\gamma_{k+1}(i(x)) : \iota(\mathsf{succ}_{k+1}(i(x))) \equiv \iota(i(x)) + 1 \mod k+1,$$

we calculate

$$\begin{aligned} \iota(\mathsf{succ}_{k+1}(i(x))) &\doteq \iota(\mathsf{skip\text{-}zero}(x)) && \text{by definition of succ} \\ &= \iota(x) + 1 && \text{by claim (ii).} \end{aligned}$$

Since the congruence relation modulo $k+1$ is reflexive, we obtain $\gamma_{k+1}(i(x))$ from the identification of the above calculation. To obtain

$$\gamma_{k+1}(\star) : \iota(\mathsf{succ}_{k+1}(\star)) \equiv \iota(\star) + 1 \mod k+1,$$

we calculate

$$\begin{aligned} \iota(\mathsf{succ}_{k+1}(\star)) &\doteq \iota(0) && \text{by definition of succ} \\ &= 0 && \text{by claim (i)} \\ &\equiv k+1 && \text{by Example 7.1.4} \\ &\doteq \iota(\star) + 1 && \text{by definition of } \iota. \end{aligned}$$ □

Proposition 7.4.5 *For any $x : \mathbb{N}$ we have*

$$\iota[x]_{k+1} \equiv x \mod k+1.$$

Proof The proof by induction on x. The fact that

$$\iota[0]_{k+1} \equiv 0 \mod k+1$$

is immediate from the fact that $\iota[0]_{k+1} \doteq \iota(0) = 0$, which was shown in Lemma 7.4.4. In the inductive step, we have to show that

$$\iota[x+1]_{k+1} \equiv x+1 \mod k+1.$$

This follows from the following computation

$$\begin{aligned} \iota[x+1]_{k+1} &\doteq \iota(\mathsf{succ}_{k+1}[x]_{k+1}) && \text{by definition of } [-]_{k+1} \\ &\equiv \iota[x]_{k+1} + 1 && \text{by Lemma 7.4.4} \\ &\equiv x+1 && \text{by the inductive hypothesis.} \end{aligned}$$ □

We need one more fact before we can prove Theorems 7.4.7 and 7.4.8.

Proposition 7.4.6 *For any natural number $x < d$ we have*

$$d \mid x \leftrightarrow x = 0.$$

Consequently, for any two natural numbers x and y such that $\mathsf{dist}_{\mathbb{N}}(x, y) < k$, we have

$$x \equiv y \mod k \leftrightarrow x = y.$$

Proof Note that the implication $x = 0 \to d \mid x$ is trivial, so it suffices to prove the forward implication

$$d \mid x \to x = 0.$$

This implication clearly holds if $x \doteq 0$. Therefore we only have to show that $d \mid x + 1$ implies $x + 1 = 0$, if we assume that $x + 1 < d$. In other words, we will derive a contradiction from the hypotheses that $x + 1 < d$ and $d \mid x + 1$. To reach a contradiction we use Exercise 6.4 (c), by which it suffices to show that $d \leq x + 1$.

We proceed by Σ-induction on the (unnamed) variable of type $d \mid x+1$, so we get to assume a natural number k equipped with an identification $p : dk = x + 1$. In the case where $k \doteq 0$ we reach an immediate contradiction via Theorem 6.4.2, because we obtain that $0 = d \cdot 0 = x + 1$. In the case where $k \doteq \mathsf{succ}_{\mathbb{N}}(k')$ it follows that

$$d \leq dk' + d \doteq dk = x + 1. \qquad \square$$

Theorem 7.4.7 *Consider a natural number k. Then we have*

$$[x]_{k+1} = [y]_{k+1} \leftrightarrow x \equiv y \mod k + 1,$$

for any $x, y : \mathbb{N}$.

Proof First note that, since ι is injective by Proposition 7.3.6, we have

$$[x]_{k+1} = [y]_{k+1} \leftrightarrow \iota[x]_{k+1} = \iota[y]_{k+1}.$$

Since the inequalities $\iota[x]_{k+1} < k + 1$ and $\iota[y]_{k+1} < k + 1$ hold by Lemma 7.3.5, it follows by Proposition 7.4.6 that

$$\iota[x]_{k+1} = \iota[y]_{k+1} \leftrightarrow \iota[x]_{k+1} \equiv \iota[y]_{k+1} \mod k + 1.$$

The latter condition is by Proposition 7.4.5 equivalent to the condition that $x \equiv y \mod k + 1$. $\square$

Theorem 7.4.8 *For any $x : \mathsf{Fin}_{k+1}$ there is an identification*

$$[\iota(x)]_{k+1} = x.$$

In other words, the map $[-]_{k+1} : \mathbb{N} \to \mathsf{Fin}_{k+1}$ is split surjective.

Proof Since $\iota : \mathsf{Fin}_{k+1} \to \mathbb{N}$ is injective by Proposition 7.3.6, it suffices to show that

$$\iota[\iota(x)]_{k+1} = \iota(x).$$

Now observe that $\iota[\iota(x)]_{k+1} < k+1$ and $\iota(x) < k+1$. By Proposition 7.4.6 it therefore suffices to show that

$$\iota[\iota(x)]_{k+1} \equiv \iota(x) \mod k+1.$$

This fact is an instance of Proposition 7.4.5. □

7.5 The cyclic group structure on the standard finite types

We now define the group operations on Fin_{k+1}, and using Theorems 7.4.7 and 7.4.8 we will show that these group operations satisfy the laws of an abelian group. This abelian group is the familiar group $\mathbb{Z}_k$ of integers modulo k, and therefore we define $\mathbb{Z}_k :\equiv \mathsf{Fin}_k$.

Definition 7.5.1 We define the **addition** operation on $\mathbb{Z}_{k+1}$ by

$$x + y :\equiv [\iota(x) + \iota(x)]_{k+1},$$

and we define the **additive inverse** operation on $\mathbb{Z}_{k+1}$ by

$$-x :\equiv [\mathsf{dist}_{\mathbb{N}}(\iota(x), k+1)]_{k+1}.$$

Remark 7.5.2 The following congruences modulo $k+1$ follow immediately from Proposition 7.4.5:

$$\begin{aligned} \iota(0) &\equiv 0 \\ \iota(x+y) &\equiv \iota(x) + \iota(y) \\ \iota(-x) &\equiv \mathsf{dist}_{\mathbb{N}}(\iota(x), k+1). \end{aligned}$$

Before we show that addition on $\mathbb{Z}_k$ satisfies the group laws, we have to show that addition on $\mathbb{N}$ preserves the congruence relation.

Proposition 7.5.3 *Consider $x, y, x', y' : \mathbb{N}$. If any two of the following three properties hold, then so does the third:*

(i) $x \equiv x' \mod k$,
(ii) $y \equiv y' \mod k$,
(iii) $x + y \equiv x' + y' \mod k$.

Proof Recall that the distance function $\mathsf{dist}_{\mathbb{N}}$ is translation invariant by Exercise 6.5 (c). Therefore it follows that

$$a \equiv b \mod k \leftrightarrow a + c \equiv b + c \mod k. \tag{$*$}$$

We will use this observation to prove the claim.

First, suppose that $x \equiv x'$ and $y \equiv y'$ modulo k. Then it follows by Eq. ($*$) that

$$x + y \equiv x' + y \equiv x' + y'.$$

This shows that (i) and (ii) together imply (iii).

Next, suppose that $x \equiv x'$ and $x + y \equiv x' + y'$ modulo k. Then it follows that

$$x + y \equiv x' + y' \equiv x + y'.$$

Applying Eq. ($*$) once more in the reverse direction, we obtain that $y \equiv y'$ modulo k. This shows that (i) and (iii) together imply (ii).

The remaining claim, that (ii) and (iii) together imply (i), follows by commutativity of addition from the fact that (i) and (iii) together imply (ii). □

Theorem 7.5.4 *The addition operation on $\mathbb{Z}_{k+1}$ satisfies the laws of an abelian group:*

$$\begin{aligned} 0 + x &= x & x + 0 &= x \\ (-x) + x &= 0 & x + (-x) &= 0 \\ (x + y) + z &= x + (y + z) & x + y &= y + x. \end{aligned}$$

Proof We first note that by commutativity of addition on $\mathbb{N}$, it follows immediately that addition on $\mathbb{Z}_{k+1}$ is commutative.

To prove associativity, note that by Theorem 7.4.7 it suffices to show that

$$\iota(x + y) + \iota(z) \equiv \iota(x) + \iota(y + z) \mod k + 1.$$

Since addition on $\mathbb{Z}_{k+1}$ maps preserves the congruence relation, and since we have the congruences

$$\begin{aligned} \iota(x + y) &\equiv \iota(x) + \iota(y) \mod k + 1 \\ \iota(y + z) &\equiv \iota(y) + \iota(z) \mod k + 1, \end{aligned}$$

it suffices to show that

$$(\iota(x) + \iota(y)) + \iota(z) \equiv \iota(x) + (\iota(y) + \iota(z)) \mod k + 1.$$

This follows immediately by associativity of addition on $\mathbb{N}$.

To show that addition on $\mathbb{Z}_{k+1}$ satisfies the right unit law, we first observe that it suffices to show that

$$[\iota(x) + \iota(0)]_{k+1} = [\iota(x)]_{k+1}$$

because there is an identification $[\iota(x)]_{k+1} = x$ by Theorem 7.4.8. By Theorem 7.4.7 it now suffices tho show that

$$\iota(x) + \iota(0) \equiv \iota(x) \mod k + 1.$$

This follows immediately from the fact that $\iota(0) = 0$. The left unit law now follows from the right unit law by commutativity. We leave the inverse laws as an exercise. □

Exercises

7.1 Complete the proof of Proposition 7.1.5.

7.2 Show that the divisibility relation satisfies the axioms of a poset, i.e., that it is reflexive, antisymmetric, and transitive.

7.3 Construct a dependent function

$$\prod_{(x:\mathbb{N})} (x \neq 0) \to ((x \leq n) \to (x \mid n!))$$

for every $n : \mathbb{N}$.

7.4 Define $1 \coloneqq [1]_{k+1} : \mathsf{Fin}_{k+1}$. Show that

$$\mathsf{succ}_{k+1}(x) = x + 1$$

for any $x : \mathsf{Fin}_{k+1}$.

7.5 The observational equality on Fin_k is a binary relation

$$\mathsf{Eq}_k : \mathsf{Fin}_k \to (\mathsf{Fin}_k \to \mathcal{U}_0)$$

defined recursively by

$$\mathsf{Eq}_{k+1}(i(x), i(y)) \coloneqq \mathsf{Eq}_k(x, y) \qquad \mathsf{Eq}_{k+1}(i(x), \star) \coloneqq \emptyset$$
$$\mathsf{Eq}_{k+1}(\star, i(y)) \coloneqq \emptyset \qquad \mathsf{Eq}_{k+1}(\star, \star) \coloneqq \mathbf{1}.$$

(a) Show that

$$(x = y) \leftrightarrow \mathsf{Eq}_k(x, y)$$

for any two elements $x, y : \mathsf{Fin}_k$.

(b) Show that the function $i : \mathsf{Fin}_k \to \mathsf{Fin}_{k+1}$ is injective, for each $k : \mathbb{N}$.

(c) Show that

$$\mathsf{succ}_{k+1}(i(x)) \neq 0$$

for any $x : \mathsf{Fin}_k$.

(d) Show that function $\mathsf{succ}_k : \mathsf{Fin}_k \to \mathsf{Fin}_k$ is injective, for each $k : \mathbb{N}$.

7.6 The predecessor function $\mathsf{pred}_k : \mathsf{Fin}_k \to \mathsf{Fin}_k$ is defined in three steps, just as in the definition of the successor function on Fin_k.

(i) We define the element $\mathsf{neg\text{-}two}_k : \mathsf{Fin}_{k+1}$ by

$$\begin{aligned}\mathsf{neg\text{-}two}_0 &:\equiv \star\\ \mathsf{neg\text{-}two}_{k+1} &:\equiv i(\star).\end{aligned}$$

(ii) We define the function $\mathsf{skip\text{-}neg\text{-}two}_k : \mathsf{Fin}_k \to \mathsf{Fin}_{k+1}$ recursively by

$$\begin{aligned}\mathsf{skip\text{-}neg\text{-}two}_{k+1}(i(x)) &:\equiv i(i(x))\\ \mathsf{skip\text{-}neg\text{-}two}_{k+1}(\star) &:\equiv \star.\end{aligned}$$

(iii) Finally, we define the **predecessor function** $\mathsf{pred}_k : \mathsf{Fin}_k \to \mathsf{Fin}_k$ recursively by

$$\begin{aligned}\mathsf{pred}_{k+1}(i(x)) &:\equiv \mathsf{skip\text{-}neg\text{-}two}_k(\mathsf{pred}_k(x))\\ \mathsf{pred}_{k+1}(\star) &:\equiv \mathsf{neg\text{-}two}_k.\end{aligned}$$

Show that pred_k is an inverse to succ_k, i.e., construct identifications

$$\mathsf{succ}_k(\mathsf{pred}_k(x)) = x, \qquad \text{and} \qquad \mathsf{pred}_k(\mathsf{succ}_k(x)) = x$$

for each $x : \mathsf{Fin}_k$.

7.7 Recall that

$$\mathsf{classical\text{-}Fin}_k :\equiv \textstyle\sum_{(x:\mathbb{N})} x < k.$$

(a) Show that

$$(x = y) \leftrightarrow (\mathsf{pr}_1(x) = \mathsf{pr}_1(y))$$

for each $x, y : \mathsf{classical\text{-}Fin}_k$.

(b) By Lemma 7.3.5 it follows that the map $\iota : \mathsf{Fin}_k \to \mathbb{N}$ induces a map $\iota : \mathsf{Fin}_k \to \mathsf{classical\text{-}Fin}_k$. Construct a map

$$\alpha_k : \mathsf{classical\text{-}Fin}_k \to \mathsf{Fin}_k$$

for each $k : \mathbb{N}$, and show that

$$\alpha_k(\iota(x)) = x \qquad \text{and} \qquad \iota(\alpha_k(y)) = y$$

for each $x : \mathsf{Fin}_k$ and each $y : \mathsf{classical\text{-}Fin}_k$.

7.8 The multiplication operation $x, y \mapsto xy$ on $\mathbb{Z}_{k+1}$ is defined by

$$xy :\equiv [\iota(x)\iota(y)]_{k+1}.$$

There are 306 exercises.

Ljubljana, November 10, 2021

(a) Show that $\iota(xy) \equiv \iota(x)\iota(y) \mod k+1$ for each $x, y : \mathbb{Z}_{k+1}$.
(b) Show that

$$xy \equiv x'y' \mod k$$

for any $x, y, x', y' : \mathbb{N}$ such that $x \equiv x' \mod k$ and $y \equiv y' \mod k$.
(c) Show that multiplication on $\mathbb{Z}_{k+1}$ satisfies the laws of a commutative ring:

$$(xy)z = x(yz) \qquad xy = yx$$
$$1x = x \qquad x1 = x$$
$$x(y+z) = xy + yz \qquad (x+y)z = xz + yz.$$

7.9 (Euclidean division) Consider two natural numbers a and b.

(a) Construct two natural numbers q and r such that $(b \neq 0) \to (r < b)$, along with an identification

$$a = qb + r.$$

(b) Show that for any four natural numbers q, q' and r, r' such that the implications $(b \neq 0) \to (r < b)$ and $(b \neq 0) \to (r' < b)$ hold, and for which there are identifications

$$a = qb + r \qquad \text{and} \qquad a = q'b + r',$$

we have $q = q'$ and $r = r'$.

7.10 The type $\mathbb{N}_k$ of **k-ary natural numbers** is an inductive type with the following constructors:

$$\mathsf{constant}_{\mathbb{N}_k} : \mathsf{Fin}_k \to \mathbb{N}_k$$
$$\mathsf{unary\text{-}op}_{\mathbb{N}_k} : \mathsf{Fin}_k \to (\mathbb{N}_k \to \mathbb{N}_k).$$

A k-ary natural number can be converted back into an ordinary natural number via the function $f_k : \mathbb{N}_k \to \mathbb{N}$, which is defined recursively by

$$f_k(\mathsf{constant}_{\mathbb{N}_k}(x)) :\equiv \iota(x)$$
$$f_k(\mathsf{unary\text{-}op}_{\mathbb{N}_k}(x, n)) :\equiv k(f_k(n) + 1) + \iota(x).$$

(a) Show that the type $\mathbb{N}_0$ is empty.
(b) Show that the function $f_k : \mathbb{N}_k \to \mathbb{N}$ is injective.
(c) Show that the function $f_{k+1} : \mathbb{N}_{k+1} \to \mathbb{N}$ has an inverse, i.e. construct a function

$$g_k : \mathbb{N} \to \mathbb{N}_{k+1}$$

equipped with identifications

$$f_{k+1}(g_k(n)) = n$$
$$g_k(f_{k+1}(x)) = x$$

for each $n : \mathbb{N}$ and each $x : \mathbb{N}_{k+1}$.

8 Decidability in elementary number theory

Martin-Löf's dependent type theory is a foundation for constructive mathematics, but in constructive mathematics there is no way to show that $P \vee \neg P$ holds for an arbitrary proposition P. Likewise, in type theory there is no way to construct an element of type $A + \neg A$ for an arbitrary type A. Consequently, if we want to reason by case analysis over whether A is empty or nonempty, we first have to *show* that $A + \neg A$ holds.

A type A that comes equipped with an element of type $A + \neg A$ is said to be *decidable*. Even though we cannot show that all types are decidable, many types are indeed decidable. Examples include the empty type and any type that comes equipped with a point, such as the type of natural numbers.

Decidability is an important concept with many applications in number theory and finite mathematics, and in this section we will explore the applications of decidability to elementary number theory. For example, the natural numbers satisfy a well-ordering principle with respect to decidable type families over the natural numbers; decidability can be used to construct the greatest common divisor of any two natural numbers; and it can also be used to show that there are infinitely many prime numbers.

8.1 Decidability and decidable equality

Definition 8.1.1 A type A is said to be **decidable** if it comes equipped with an element of type

$$\mathsf{is\text{-}decidable}(A) :\equiv A + \neg A.$$

A family P over a type A is said to be **decidable** if $P(x)$ is decidable for every $x : A$.

Example 8.1.2 The principal way to show that a type A is decidable is to either construct an element $a : A$, or to construct a function $A \to \emptyset$. For example, the types $\mathbf{1}$ and $\emptyset$ are decidable. Indeed, we have

$$\mathsf{inl}(\star) : \mathsf{is\text{-}decidable}(\mathbf{1})$$

$$\mathsf{inr}(\mathsf{id}) : \mathsf{is\text{-}decidable}(\emptyset).$$

Furthermore, any type A equipped with a point $a : A$ is decidable because we have $\mathsf{inl}(a) : \mathsf{is\text{-}decidable}(A)$ for such A.

Example 8.1.3 The principal way to use a hypothesis that A is decidable is to proceed by the induction principle of coproducts, i.e., to proceed by case analysis.

For example, if A and B are decidable types, then the types $A + B$, $A \times B$, and $A \to B$ are also decidable. This is straightforward to prove directly by pattern-matching on the variables of type $\mathsf{is\text{-}decidable}(A)$ and $\mathsf{is\text{-}decidable}(B)$. When we go through these proofs, the familiar truth table emerges:

is-decidable				
A	B	$A + B$	$A \times B$	$A \to B$
$\mathsf{inl}(a)$	$\mathsf{inl}(b)$	$\mathsf{inl}(\mathsf{inl}(a))$	$\mathsf{inl}(a, b)$	$\mathsf{inl}(\lambda x.\, b)$
$\mathsf{inl}(a)$	$\mathsf{inr}(g)$	$\mathsf{inl}(\mathsf{inl}(a))$	$\mathsf{inr}(g \circ \mathsf{pr}_2)$	$\mathsf{inr}(\lambda h.\, g(h(a)))$
$\mathsf{inr}(f)$	$\mathsf{inl}(b)$	$\mathsf{inl}(\mathsf{inr}(b))$	$\mathsf{inr}(f \circ \mathsf{pr}_1)$	$\mathsf{inl}(\mathsf{ex\text{-}falso} \circ f)$
$\mathsf{inr}(f)$	$\mathsf{inr}(g)$	$\mathsf{inr}[f, g]$	$\mathsf{inr}(f \circ \mathsf{pr}_1)$	$\mathsf{inl}(\mathsf{ex\text{-}falso} \circ f)$

Since $A \to B$ is decidable whenever both A and B are decidable, it also follows that the negation $\neg A$ of any decidable type A is decidable.

Example 8.1.4 Since the empty type and the unit type are both decidable types, it also follows that the types $\mathsf{Eq}_{\mathbb{N}}(m, n)$, $m \leq n$ and $m < n$ are decidable for each $m, n : \mathbb{N}$. The proofs in each of the three cases is by induction on m and n.

For instance, to show that $\mathsf{Eq}_{\mathbb{N}}(m, n)$ is decidable for each $m, n : \mathbb{N}$, we simply note that the types

$$\begin{aligned} \mathsf{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) &\doteq \mathbf{1} \\ \mathsf{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, \mathsf{succ}_{\mathbb{N}}(n)) &\doteq \emptyset \\ \mathsf{Eq}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m), 0_{\mathbb{N}}) &\doteq \emptyset \end{aligned}$$

are all decidable, and that the type $\mathsf{Eq}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m), \mathsf{succ}_{\mathbb{N}}(n)) \doteq \mathsf{Eq}_{\mathbb{N}}(m, n)$ is decidable by the inductive hypothesis.

The fact that $\mathbb{N}$ has decidable observational equality also implies that equality itself is decidable on $\mathbb{N}$. This leads to the general concept of decidable equality, which is important in many results about decidability.

Definition 8.1.5 We say that a type A has **decidable equality** if the identity type $x = y$ is decidable for every $x, y : A$. We will write

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

Before we show that $\mathbb{N}$ has decidable equality, let us show that if $A \leftrightarrow B$ and A is decidable, then B must be decidable.

Lemma 8.1.6 *Consider two types A and B, and suppose that $A \leftrightarrow B$. Then A is decidable if and only if B is decidable.*

Proof Since we have functions $f : A \to B$ and $g : B \to A$ by assumption, we obtain by Proposition 4.3.4 the functions

$$\tilde{f} : \neg B \to \neg A$$
$$\tilde{g} : \neg A \to \neg B.$$

By Remark 4.4.2 we have therefore the functions

$$f + \tilde{g} : (A + \neg A) \to (B + \neg B)$$
$$g + \tilde{f} : (B + \neg B) \to (A + \neg A).$$ □

Proposition 8.1.7 *Equality on the natural numbers is decidable.*

Proof Recall from Proposition 6.3.3 that we have

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

The claime therefore follows by Lemma 8.1.6, since we have observed in Example 8.1.4 that $\mathsf{Eq}_{\mathbb{N}}(m, n)$ is decidable for every $m, n : \mathbb{N}$. □

It is certainly not provable with the given rules of type theory that every type has decidable equality. In fact, we will show in Theorem 12.3.5 that if a type has decidable equality, then it is a *set*. However, it is also not provable that every set has decidable equality unless one assumes the *law of excluded middle*. We will discuss this principle in Section 17.4. For now, it is important to remember that in order to use decidability, we must first *prove that it holds*, and many familiar types do indeed have decidable equality.

Proposition 8.1.8 *The type Fin_k has decidable equality for each $k : \mathbb{N}$.*

Proof Recall from Exercise 7.5 that we constructed an observational equality relation Eq_k on Fin_k for each $k : \mathbb{N}$, which satisfies

$$(x = y) \leftrightarrow \mathsf{Eq}_k(x, y).$$

The type $\mathsf{Eq}_k(x, y)$ is decidable, since it is recursively defined using the decidable types $\emptyset$ and $\mathbf{1}$. □

We can use the fact that the finite types Fin_k have decidable equality to show that the divisibility relation on $\mathbb{N}$ is decidable.

Theorem 8.1.9 *For any $d, x : \mathbb{N}$, the type $d \mid x$ is decidable.*

Proof Note that $0 \mid x$ is decidable because $0 \mid x$ if and only if $x = 0$, which is decidable by Proposition 8.1.7. Therefore it suffices to show that $d + 1 \mid x$ is decidable.

By Theorem 7.4.7 it follows that $d + 1 \mid x$ holds if and only if we have an identification $[x]_{d+1} = 0$ in Fin_{d+1}. Therefore the claim follows from the fact that Fin_{d+1} has decidable equality. □

8.2 Constructions by case analysis

A common way to construct functions and to prove properties about them is by case analysis. For example, a famous function of Collatz is specified by case analysis on whether n is even or odd:

$$\mathsf{collatz}(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3n+1 & \text{if } n \text{ is odd.} \end{cases}$$

The Collatz function is of course uniquely determined by this specification, but it is important to note that there is a bit of work to be done in order to define the Collatz function according to the rules of dependent type theory. First we note that, since the Collatz function is specified by case analysis on whether n is even or odd, we will have to use a dependent function witnessing the fact that every number is either even or odd. In other words, we will make use of the dependent function

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

which we have by Theorem 8.1.9. The type $\mathsf{is\text{-}decidable}(2 \mid n)$ is the coproduct $(2 \mid n) + (2 \nmid n)$, so the idea is to proceed by case analysis on whether $d(n)$ is of the form $\mathsf{inl}(x)$ or $\mathsf{inr}(x)$, i.e., by the induction principle of coproducts. However, $d(n)$ is not a free variable of type $\mathsf{is\text{-}decidable}(2 \mid n)$. Before we can proceed by induction, we must therefore first *generalize* the element $d(n)$ to a free variable $y : \mathsf{is\text{-}decidable}(2 \mid n)$. In other words, we will first define a function

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

by the induction principle of coproducts, and then we obtain the Collatz function by substituting $d(n)$ for y in $h(n, y)$. Putting these ideas together, we obtain the following type theoretical definition of the Collatz function.

Definition 8.2.1 Write $d : \prod_{(n:\mathbb{N})} \mathsf{is\text{-}decidable}(2 \mid n)$ for the function deciding $2 \mid n$, given in Theorem 8.1.9.

(i) We define a function $h : \prod_{(n:\mathbb{N})} (\mathsf{is\text{-}decidable}(2 \mid n) \to \mathbb{N})$ by

$$h(n, \mathsf{inl}(m, p)) :\equiv m$$
$$h(n, \mathsf{inr}(f)) :\equiv 3n + 1.$$

(ii) We define the function $\mathsf{collatz} : \mathbb{N} \to \mathbb{N}$ by

$$\mathsf{collatz}(n) :\equiv h(n, d(n)).$$

Remark 8.2.2 The general ideas behind the formal construction of the Collatz function lead to the type theoretic concept of *with-abstraction*. With-abstraction is a type-theoretically precise generalization of case analysis.

In full generality, if our goal is to define a dependent function $f : \prod_{(x:A)} C(x)$, and we already have a function $g : \prod_{(x:A)} B(x)$, then it suffices to define a dependent function

$$h : \prod_{(x:A)} B(x) \to C(x).$$

Indeed, given g and h as above, we can define $f :\equiv \lambda x.\, h(x, g(x))$. In other words, to define $f(x)$ using $g(x) : B(x)$, we generalize $g(x)$ to an arbitrary element $y : B(x)$ and proceed to define an element $h(x, y) : C(x)$.

With-abstraction is a concise way to present such a definition. In a definition by with-abstraction, we may write

$$f(x) \textit{ with } [g(x)/y] :\equiv h(x, y),$$

to define a function $f : \prod_{(x:A)} C(x)$ that satisfies the judgmental equality $f \doteq \lambda x.\, h(x, g(x))$. In other words, $f(x)$ is defined to be $h(x, y)$ with $g(x)$ for y.

The definition of the Collatz function can therefore be given by with-abstraction as

$$\mathsf{collatz}(n) \textit{ with } [d(n)/y] :\equiv h(x, y).$$

However, recall that the function h was defined by pattern matching on y. We can combine with-abstraction and pattern matching to obtain a *direct* definition of the Collatz function that doesn't explicitly mention the function h anymore. This gives us the following concise way to define the Collatz function:

$$\mathsf{collatz}(n) \textit{ with } [d(n)/\mathsf{inl}(m, p)] :\equiv m$$
$$\mathsf{collatz}(n) \textit{ with } [d(n)/\mathsf{inr}(f)] :\equiv 3n + 1.$$

Notice that in addition to the information in the specification of the Collatz

function, the definition by with-abstraction also tells us which decision procedure was used to decide whether n is even or not. The combination of with-abstraction and pattern matching, which allows us to skip the explicit definition of the function h, is what makes with-abstraction so useful.

Using with-abstraction we can find a slight improvement of the decidability results of $A \to B$ and $A \times B$ in Example 8.1.3, and we will use these improved claims in the construction of the greatest common divisor.

Proposition 8.2.3 *Consider a decidable type A, and let B be a type equipped with a function*

$$A \to \mathsf{is\text{-}decidable}(B).$$

Then the types $A \times B$ and $A \to B$ are also decidable.

Proof We only prove the claim about the decidability of $A \to B$, since the claim about the decidability of $A \times B$ is proven similarly. Since A is assumed to be decidable, we proceed by case analysis on $A + \neg A$. In the case where we have $f : \neg A$, we have the functions

$$A \xrightarrow{f} \emptyset \xrightarrow{\mathsf{ex\text{-}falso}} B.$$

Therefore we obtain the element $\mathsf{inl}(\mathsf{ex\text{-}falso} \circ f) : \mathsf{is\text{-}decidable}(A \to B)$. In the case where we have an element $a : A$, we have to construct a function

$$d : (A \to \mathsf{is\text{-}decidable}(B)) \to \mathsf{is\text{-}decidable}(A \to B)$$

Given $H : A \to \mathsf{is\text{-}decidable}(B)$, we can use with-abstraction to proceed by case analysis on $H(a) : B + \neg B$. The function d is therefore defined as

$$\begin{aligned} d(H) \ \textit{with}\ [H(a)/\mathsf{inl}(b)] &:\equiv \mathsf{inl}(\lambda x.\, b) \\ d(H) \ \textit{with}\ [H(a)/\mathsf{inr}(g)] &:\equiv \mathsf{inr}(\lambda h.\, g(h(a))). \end{aligned}$$

□

For a general family of decidable types P over $\mathbb{N}$, we cannot prove that the type

$$\textstyle\prod_{(x:\mathbb{N})} P(x)$$

is decidable. However, if we know in advance that $P(x)$ holds for any $m \leq x$, then we can decide $\prod_{(x:\mathbb{N})} P(x)$ by checking the decidability of each $P(x)$ until m.

Proposition 8.2.4 *Consider a decidable type family P over $\mathbb{N}$ equipped with a natural number m such that the type*

$$\textstyle\prod_{(x:\mathbb{N})} (m \leq x) \to P(x)$$

is decidable. Then the type $\prod_{(x:\mathbb{N})} P(x)$ *is decidable.*

Proof Our proof is by induction on m, but we will first make sure that the inductive hypothesis will be strong enough by quantifying over all decidable type families over $\mathbb{N}$. Of course, we cannot do this directly. However, by the assumption that there are enough universes (Postulate 6.2.1), there is a universe $\mathcal{U}$ that contains P. We fix this universe, and we will prove by induction on m that for every decidable type family $Q : \mathbb{N} \to \mathcal{U}$ for which the type

$$\prod_{(x:\mathbb{N})} (m \leq x) \to Q(x),$$

is decidable, the type $\prod_{(x:\mathbb{N})} Q(x)$ is again decidable.

In the base case, it follows by assumption that the type $\prod_{(x:A)} Q(x)$ is decidable. For the inductive step, let $Q : \mathbb{N} \to \mathcal{U}$ be a decidable type family for which the type

$$\prod_{(x:\mathbb{N})} (m + 1 \leq x) \to Q(x)$$

is decidable. Since Q is assumed to be decidable, we can proceed by case analysis on $Q(0) + \neg Q(0)$. In the case of $\neg Q(0)$, it follows that $\neg \prod_{(x:\mathbb{N})} Q(x)$. In the case where we have $q : Q(0)$, consider the type family $Q' : \mathbb{N} \to \mathcal{U}$ given by

$$Q'(x) :\equiv Q(x + 1).$$

Then Q' is decidable since Q is decidable, and moreover it follows that the type $\prod_{(x:\mathbb{N})} (m \leq x) \to Q'(x)$ is decidable. The inductive hypothesis implies therefore that the type $\prod_{(x:\mathbb{N})} Q'(x)$ is decidable. In the case where $\neg \prod_{(x:\mathbb{N})} Q'(x)$, it follows that $\neg \prod_{(x:\mathbb{N})} Q(x)$, and in the case where we have a function $g : \prod_{(x:\mathbb{N})} Q'(x)$, we can construct a function $f : \prod_{(x:\mathbb{N})} Q(x)$ by

$$\begin{aligned} f(0) &:\equiv q \\ f(x + 1) &:\equiv g(x). \end{aligned}$$

□

Corollary 8.2.5 *Consider two decidable families P and Q over $\mathbb{N}$, and suppose that P comes equipped with an upper bound m. Then the type*

$$\prod_{(n:\mathbb{N})} P(n) \to Q(n)$$

is decidable.

Proof Since m is assumed to be an upper bound for P, it follows $P(n) \to Q(n)$ for any $m \leq n$. With this observation we apply Proposition 8.2.4. □

8.3 The well-ordering principle of $\mathbb{N}$

The well-ordering principle of the natural numbers in classical mathematics asserts that any nonempty subset of $\mathbb{N}$ has a least element. To formulate the well-ordering principle in type theory, we will use type families over $\mathbb{N}$ instead of subsets of $\mathbb{N}$. Moreover, the classical well-ordering principle tacitly assumes that subsets are decidable. The type theoretic well-ordering principle of $\mathbb{N}$ is therefore formulated using *decidable* families over $\mathbb{N}$.

Definition 8.3.1 Let P be a family over $\mathbb{N}$, not necessarily decidable.

(i) We say that a natural number n is a **lower bound** for P if it comes equipped with an element of type

$$\mathsf{is\text{-}lower\text{-}bound}_P(n) :\equiv \prod_{(x:\mathbb{N})} P(x) \to (n \leq x).$$

(ii) We say that a natural number n is an **upper bound** for P if it comes equipped with an element of type

$$\mathsf{is\text{-}upper\text{-}bound}_P(n) :\equiv \prod_{(x:\mathbb{N})} P(x) \to (x \leq n).$$

A minimal element of P is therefore a natural number n for which $P(n)$ holds, and which is also a lower bound for P. The well-ordering principle of $\mathbb{N}$ asserts that such an element exists for any decidable family P, as soon as $P(n)$ holds for some n.

Theorem 8.3.2 (Well-ordering principle of $\mathbb{N}$) *Let P be a decidable family over $\mathbb{N}$, where d witnesses that P is decidable. Then there is a function*

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

Proof By the assumption that there are enough universes (Postulate 6.2.1), there is a universe $\mathcal{U}$ that contains P. Instead of proving the claim for the given type family P, we will show by induction on $n : \mathbb{N}$ that there is a function

$$Q(n) \to \left(\sum_{(m:\mathbb{N})} Q(m) \times \mathsf{is\text{-}lower\text{-}bound}_Q(m)\right) \qquad (*)$$

for every decidable family $Q : \mathbb{N} \to \mathcal{U}$. Note that we are now also quantifying over the decidable families $Q : \mathbb{N} \to \mathcal{U}$. This slightly strengthens the inductive hypothesis, which we will be able to exploit.

The base case is trivial, since $0_{\mathbb{N}}$ is a lower bound of every type family over $\mathbb{N}$. For the inductive step, assume that Eq. $(*)$ holds for every decidable type family $Q : \mathbb{N} \to \mathcal{U}$. Furthermore, let $Q : \mathbb{N} \to \mathcal{U}$ be a decidable type family equipped with an element $q : Q(\mathsf{succ}_{\mathbb{N}}(n))$. Our goal is to construct an element of type

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

Since $Q(0_\mathbb{N})$ is assumed to be decidable, it suffices to construct a function

$$(Q(0_\mathbb{N}) + \neg Q(0_\mathbb{N})) \to \textstyle\sum_{(m:\mathbb{N})} Q(m) \times \mathsf{is\text{-}lower\text{-}bound}_Q(m).$$

Therefore we can proceed by case analysis on $Q(0_\mathbb{N}) + \neg Q(0_\mathbb{N})$. In the case where we have an element of type $Q(0_\mathbb{N})$, it follows immediately that $0_\mathbb{N}$ must be minimal. In the case where $\neg Q(0_\mathbb{N})$, we consider the decidable subset Q' of $\mathbb{N}$ given by

$$Q'(n) :\equiv Q(\mathsf{succ}_\mathbb{N}(n)).$$

Since we have $q : Q'(n)$, we obtain a minimal element in Q' by the inductive hypothesis. Of course, by the assumption that $Q(0_\mathbb{N})$ doesn't hold, the minimal element of Q' is also the minimal element of Q. □

8.4 The greatest common divisor

The greatest common divisor of two natural numbers a and b is a natural number $\mathsf{gcd}(a, b)$ that satisfies the property that

$$x \mid a \text{ and } x \mid b \qquad \text{if and only if} \qquad x \mid \mathsf{gcd}(a, b)$$

for any $x : \mathbb{N}$. In other words, any number $x : \mathbb{N}$ that divides both a and b also divides the greatest common divisor. Moreover, since $\mathsf{gcd}(a, b)$ divides itself, it follows from the reverse implication that $\mathsf{gcd}(a, b)$ divides both a and b.

This property can also be seen as the *specification* of what it means to be a greatest common divisor of a and b. In formal developments of mathematics, when you're about to construct an object that satisfies a certain specification, it can be useful to start out with that specification. For example, there is more than one way to define the greatest common divisor—we will define it here in Definition 8.4.6 using the well-ordering principle and again in **??** using Euclid's algorithm—and both definitions satisfy the specification that uniquely characterizes it. Hence we make the following definition.

Definition 8.4.1 Consider three natural numbers a, b, and d. We say that d is a **greatest common divisor** of a and b if it comes equipped with an element of type

$$\mathsf{is\text{-}gcd}_{a,b}(d) :\equiv \textstyle\prod_{(x:\mathbb{N})} (x \mid a) \times (x \mid b) \leftrightarrow (x \mid d).$$

The property of being a greatest common divisor uniquely characterizes the greatest common divisor, in the following sense.

Proposition 8.4.2 *Suppose d and d' are both a greatest common divisor of a and b. Then $d = d'$.*

Proof If both d and d' are a greatest common divisor of a and b, then both d and d' divide both a and b, and hence it follows that $d \mid d'$ and $d' \mid d$. Since the divisibility relation was shown to be a partial order in Exercise 7.2, it follows by antisymmetry that $d = d'$. □

Note that for any two natural numbers a and b, the type

$$\textstyle\sum_{(n:\mathbb{N})} \prod_{(x:\mathbb{N})} (x \mid a) \times (x \mid b) \to (x \mid n) \tag{$*$}$$

consists of all the multiples of the common divisors of a and b, including 0. On the other hand, the type

$$\textstyle\sum_{(n:\mathbb{N})} \prod_{(x:\mathbb{N})} (x \mid n) \to (x \mid a) \times (x \mid b) \tag{$**$}$$

consists of all the common divisors of a and b except in the case where $a = 0$ and $b = 0$. In this case, the type in Eq. (∗∗) consists of all natural numbers.

Eqs. (∗) and (∗∗) provide us with two ways to define the greatest common divisor. We can either define the greatest common divisor of a and b as the greatest natural number in the type in Eq. (∗∗) or we can define it as the least *nonzero* natural number of the type in Eq. (∗), provided that we make an exception in the case where both $a = 0$ and $b = 0$. Since we already have established the well-ordering principle of $\mathbb{N}$, we will opt for the second approach. In Exercise 8.10 you will be asked to show that any *bounded* decidable family over $\mathbb{N}$ has a maximum as soon as it contains some natural number.

In order to correctly define the greatest common divisor using well-ordering principle of $\mathbb{N}$, we need a slight modification of the type family in Eq. (∗). We define this family as follows:

Definition 8.4.3 Given $a, b : \mathbb{N}$, we define the type family $M(a, b)$ over $\mathbb{N}$ by

$$M(a, b, n) :\equiv (a + b \neq 0) \to (n \neq 0) \times \left(\textstyle\prod_{(x:\mathbb{N})} (x \mid a) \times (x \mid b) \to (x \mid n)\right).$$

In other words, if $a + b = 0$ then the type $\sum_{(n:\mathbb{N})} M(a, b, n)$ consist of all the natural numbers. On the other hand, if $a + b \neq 0$ it consists of the nonzero natural numbers n with the property that any common divisor of a and b also divides n. These are exactly the nonzero multiples of the greatest common divisor of a and b.

Since we intend to apply the well-ordering principle, we must show that the family $M(a, b)$ is decidable. This is a step that one can skip in classical mathematics, because all the subsets of $\mathbb{N}$ are decidable there. However, in our current setting we have no choice but to prove it.

Proposition 8.4.4 *The type family $M(a, b)$ is decidable for each $a, b : \mathbb{N}$.*

Proof The type $a + b \neq 0$ is decidable because it is the negation of the type $a + b = 0$, which is decidable by Proposition 8.1.7. Therefore it suffices to show that the type

$$(n \neq 0) \times \prod_{(x:\mathbb{N})} (x \mid a) \times (x \mid b) \to (x \mid n)$$

is decidable, and by Proposition 8.2.3 we also get to assume that $a + b \neq 0$. The type $n \neq 0$ is again decidable by Proposition 8.1.7, so it suffices to show that the type

$$\prod_{(x:\mathbb{N})} (x \mid a) \times (x \mid b) \to (x \mid n)$$

is decidable. The types $(x \mid a) \times (x \mid b)$ and $(x \mid n)$ are decidable by Theorem 8.1.9, so by Corollary 8.2.5 it suffices to check that the family of types $(x \mid a) \times (x \mid b)$ indexed by $x : \mathbb{N}$ has an upper bound. If x is a common divisor of a and b, then it follows that x divides $a + b$. Furthermore, since we have assumed that $a + b \neq 0$, it follows that $x \leq a + b$. This provides the upper bound. □

We are almost in position to apply the well-ordering principle of $\mathbb{N}$ to define the greatest common divisor. It just remains to show that there is some $n : \mathbb{N}$ for which $M(a, b, n)$ holds. We prove this in the following lemma.

Lemma 8.4.5 *There is an element of type* $M(a, b, a + b)$.

Proof To construct an element of type $M(a, b, a + b)$, assume that $a + b \neq 0$. Then we have tautologically that $a + b \neq 0$, and any common divisor of a and b is also a divisor of $a + b$. □

Definition 8.4.6 We define the **greatest common divisor** $\mathsf{gcd} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ by the well-ordering principle of $\mathbb{N}$ (Theorem 8.3.2) as the least natural number n for which $M(a, b, n)$ holds, using the fact that $M(a, b)$ is a decidable type family (Proposition 8.4.4) and that $M(a, b, a + b)$ always holds (Lemma 8.4.5).

Lemma 8.4.7 *For any two natural numbers a and b, we have* $\mathsf{gcd}(a, b) = 0$ *if and only if* $a + b = 0$.

Proof To prove the forward direction, assume that $\mathsf{gcd}(a, b) = 0$. By definition of $\mathsf{gcd}(a, b)$ we have that $M(a, b, \mathsf{gcd}(a, b))$ holds. More explicitly, the implication

$$(a + b \neq 0) \to (\mathsf{gcd}(a, b) \neq 0) \times \prod_{(x:\mathbb{N})} (x \mid a) \times (x \mid b) \to (x \mid \mathsf{gcd}(a, b))$$

holds. However, we have assumed that $\mathsf{gcd}(a, b) = 0$, so it follows from the above implication that $\neg(a + b \neq 0)$. In other words, we have $\neg\neg(a + b = 0)$.

The fact that equality on $\mathbb{N}$ is decidable implies via Exercise 4.3 (d) that $\neg\neg(a + b = 0) \to (a + b = 0)$, so we conclude that $a + b = 0$.

For the converse direction, recall that the inequality $\gcd(a, b) \leq a + b$ holds by minimality, since $M(a, b, a + b)$ holds by Lemma 8.4.5. If $a + b = 0$, it therefore follows that $\gcd(a, b) \leq 0$, which implies that $\gcd(a, b) = 0$. □

Theorem 8.4.8 *For any two natural numbers a and b, the number* $\gcd(a, b)$ *is a greatest common divisor of a and b in the sense of Definition 8.4.1.*

Proof We give the proof by case analysis on whether $a + b = 0$. If we assume that $a + b = 0$, then it follows that both $a = 0$ and $b = 0$, and by Lemma 8.4.7 it also follows that $\gcd(a, b) = 0$. Since any number divides 0, the claim follows immediately.

In the case where $a+b \neq 0$, it follows from Lemma 8.4.7 that also $\gcd(a, b) \neq 0$. From the fact that $M(a, b, \gcd(a, b))$ we therefore immediately obtain that

$$\prod_{(x:\mathbb{N})} (x \mid a) \times (x \mid b) \to (x \mid \gcd(a, b)).$$

Therefore it remains to show that if x divides $\gcd(a, b)$, then x divides both a and b. By transitivity of the divisibility relation it suffices to show that $\gcd(a, b)$ divides both a and b. We will show only that $\gcd(a, b)$ divides a, the proof that $\gcd(a, b)$ divides b is similar.

Since $\gcd(a, b)$ is nonzero, it follows by Euclidean divison (Exercise 7.9) that there are numbers q and $r < \gcd(a, b)$ such that

$$a = q \cdot \gcd(a, b) + r.$$

From this equation and Proposition 7.1.5 it follows that any number x which divides both a and b also divides r, because we have already noted that any such x divides $\gcd(a, b)$. This observation implies that $r = 0$, because we have $r < \gcd(a, b)$ by construction and $\gcd(a, b)$ is minimal. Therefore we conclude that $\gcd(a, b)$ divides a. □

8.5 The infinitude of primes

When the natural numbers are ordered by the divisibility relation, the number 1 is at the bottom. Directly above 1 are the prime numbers. Above the prime numbers are the multiples of two primes, then the multiples of three primes, and so on. At the top of this ordering we find 0. For any natural number n, the numbers strictly below n are the proper divisors of n. A prime number is therefore a number of which has exactly one proper divisor.

Definition 8.5.1

(i) Consider two natural numbers d and n. Then d is said to be a **proper divisor** of n if it comes equipped with an element of type

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

(ii) A natural number n is said to be **prime** if it comes equipped with an element of type

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

Proposition 8.5.2 *For any $n : \mathbb{N}$, the type $\mathsf{is\text{-}prime}(n)$ is decidable.*

Proof We will first show that $\mathsf{is\text{-}prime}(n) \leftrightarrow \mathsf{is\text{-}prime}'(n)$, where

$$\mathsf{is\text{-}prime}'(n) :\equiv (n \neq 1) \times \prod_{(x:\mathbb{N})} \mathsf{is\text{-}proper\text{-}divisor}(n,x) \to (x = 1).$$

For the forward direction, simply note that 1 is not a proper divisor of itself, and therefore 1 is not a prime. For the converse direction, suppose that $n \neq 1$ and that any proper divisor of n is 1. Then it follows that 1 is a proper divisor of n, which implies that n is prime.

Now we proceed by showing that the type $\mathsf{is\text{-}prime}'(n)$ is decidable for every $n : \mathbb{N}$. The proof is by case analysis on whether $n = 0$ or $n \neq 0$. In the case where $n = 0$, note that any nonzero number is a proper divisor of 0, and therefore $\mathsf{is\text{-}prime}'(0)$ doesn't hold. In particular, $\mathsf{is\text{-}prime}'(0)$ is decidable.

Now suppose that $n \neq 0$. In order to show that the type $\mathsf{is\text{-}prime}'(n)$ is decidable, note that the type $n \neq 1$ is decidable since it is the negation of the decidable type $n = 1$. Therefore it suffices ot show that the type

$$\prod_{(x:\mathbb{N})} \mathsf{is\text{-}proper\text{-}divisor}(n,x) \to (x = 1)$$

is decidable. Since the types $(x \neq n) \times (x \mid n)$ and $x = 1$ are decidable, it follows from Corollary 8.2.5 that it suffices to check that

$$((x \neq n) \times (x \mid n)) \to (x \leq n)$$

for any $x : \mathbb{N}$. This follows from the implication $(x \mid n) \to (x \leq n)$, which holds because we have assumed that $n \neq 0$. □

The proof that there are infinitely many primes proceeds by constructing a prime number larger than n, for any $n : \mathbb{N}$. The number $n! + 1$ is relatively prime with any number $x \leq n$. Therefore there is a least number $n < m$ that is relatively prime with any number $x \leq n$, and it follows that this number m must be prime.

Definition 8.5.3 For any two natural numbers n and m, we define the type

$$R(n,m) :\equiv (n < m) \times \prod_{(x:\mathbb{N})} (x \leq n) \to ((x \mid m) \to (x = 1)).$$

Lemma 8.5.4 *The type $R(n,m)$ is decidable for each $n,m:\mathbb{N}$.*

Proof The type $n < m$ and, and for each $x:\mathbb{N}$ both types $x \leq n$ and $(x \mid m) \to (x = 1)$ are decidable, so it follows via Corollary 8.2.5 that the product

$$\prod_{(x:\mathbb{N})}(x \leq n) \to ((x \mid m) \to (x = 1))$$

is decidable. □

Lemma 8.5.5 *There is an element of type $R(n, n!+1)$ for each $n:\mathbb{N}$.*

Proof The fact that $n < n!+1$ follows from the fact that $n \leq n!$, which is shown by induction. We leave this to the reader, and focus on the second aspect of the claim: that every $x \leq n$ that divides $n!+1$ must be equal to 1.

To see this, note that any divisor of $n!+1$ is automatically nonzero, and recall that any nonzero $x \leq n$ divides $n!$ by Exercise 7.3. Therefore it follows that any $x \leq n$ that divides $n!+1$ also divides $n!$, and consequently it divides 1 as well. Now we are done, because if x divides 1 then $x = 1$. □

We finally show that there are infinitely many primes.

Theorem 8.5.6 *For each $n:\mathbb{N}$, there is a prime number $p:\mathbb{N}$ such that $n < p$.*

Proof It suffices to show that for each *nonzero* $n:\mathbb{N}$, there is a prime number $p:\mathbb{N}$ such that $n \leq p$. Let n be a nonzero natural number.

Since the type $R(n,m)$ is decidable for each $m:\mathbb{N}$, and since $R(n,n!+1)$ holds by Lemma 8.5.5, it follows by the well-ordering principle of $\mathbb{N}$ (Theorem 8.3.2) that there is a minimal $m:\mathbb{N}$ such that $R(n,m)$ holds. In order to prove the theorem, we will show that this number m is prime, i.e., that there is an element of type

$$(m \neq 1) \times \prod_{(x:\mathbb{N})} \mathsf{is\text{-}proper\text{-}divisor}(m,x) \to (x = 1).$$

First, we note that $m \neq 1$ because $n < m$ holds by construction, and n is assumed to be nonzero. Therefore it suffices to show that 1 is the only proper divisor of m. Let x be a proper divisor of m. Since $R(n,m)$ holds by construction, we will prove that $x = 1$ by showing that $x \leq n$ holds.

Since m is nonzero, it follows from the assumption that $x \mid m$ that $x < m$. By minimality of m, it therefore follows that $\neg R(n,x)$ holds. However, any divisor of x is also a divisor of m by transitivity of the divisibility relation. Therefore it follows that any $y \leq n$ that divides x must be 1. In other words:

$$\prod_{(y:\mathbb{N})}(y \leq n) \to ((y \mid x) \to (y = 1))$$

holds. Since $\neg R(n,x)$ holds, we conclude now that $n \not< x$. To finish the proof, it follows that $x \leq n$. □

Exercises

8.1 (a) State Goldbach's Conjecture in type theory.
(b) State the Twin Prime Conjecture in type theory.
(c) State the Collatz Conjecture in type theory.
If you have a solution to any of these open problems, you should certainly formalize it before you submit it to the Annals of Mathematics.

8.2 Show that

$$\mathsf{is\text{-}decidable}(\mathsf{is\text{-}decidable}(P)) \to \mathsf{is\text{-}decidable}(P)$$

for any type P.

8.3 For any family P of decidable types indexed by Fin_k, construct a function

$$\neg\left(\prod_{(x:\mathsf{Fin}_k)} P(x)\right) \to \sum_{(x:\mathsf{Fin}_k)} \neg P(x).$$

8.4 (a) Define the function $\mathsf{prime} : \mathbb{N} \to \mathbb{N}$ for which $\mathsf{prime}(n)$ is the n-th prime.
(b) Define the **prime-counting function** $\pi : \mathbb{N} \to \mathbb{N}$, which counts for each $n : \mathbb{N}$ the number of primes $p \leq n$.

8.5 For any natural number n, show that

$$\mathsf{is\text{-}prime}(n) \leftrightarrow (2 \leq n) \times \prod_{(x:\mathbb{N})} (x \mid n) \to (x = 1) + (x = n).$$

8.6 Consider two types A and B. Show that the following are equivalent:

(i) There are functions

$$B \to \mathsf{has\text{-}decidable\text{-}eq}(A)$$
$$A \to \mathsf{has\text{-}decidable\text{-}eq}(B).$$

(ii) The product $A \times B$ has decidable equality.

Conclude that if both A and B have decidable equality, then so does $A \times B$.

8.7 Consider two types A and B, and consider the observational equality $\mathsf{Eq\text{-}copr}$ on the coproduct $A + B$ defined by

$$\mathsf{Eq\text{-}copr}(\mathsf{inl}(x), \mathsf{inl}(x')) :\equiv x = x' \qquad \mathsf{Eq\text{-}copr}(\mathsf{inl}(x), \mathsf{inr}(y')) :\equiv \emptyset$$
$$\mathsf{Eq\text{-}copr}(\mathsf{inr}(y), \mathsf{inl}(x')) :\equiv \emptyset \qquad \mathsf{Eq\text{-}copr}(\mathsf{inr}(y), \mathsf{inr}(y')) :\equiv y = y'.$$

(a) Show that $(x = y) \leftrightarrow \mathsf{Eq\text{-}copr}(x, y)$ for every $x, y : A + B$.
(b) Show that the following are equivalent:
(i) Both A and B have decidable equality.
(ii) The coproduct $A + B$ has decidable equality.

8.8 Consider a family B over A, and consider the following three conditions:

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

(i) The type A has decidable equality.
(ii) The type $B(x)$ has decidable equality for each $x : A$.
(iii) The type $\sum_{(x:A)} B(x)$ has decidable equality.

Show that if (i) holds, then (ii) and (iii) are equivalent, and show that if B has a section $b : \prod_{(x:A)} B(x)$, then (ii) and (iii) together imply (i).

8.9 Consider a family B of types over Fin_k, for some $k : \mathbb{N}$.

(a) Show that if each $B(x)$ is decidable, then $\prod_{(x:\mathsf{Fin}_k)} B(x)$ is again decidable.
(b) Show that if each $B(x)$ has decidable equality, then $\prod_{(x:\mathsf{Fin}_k)} B(x)$ also has decidable equality.

8.10 Consider a decidable type family P over $\mathbb{N}$ equipped with an upper bound m.

(a) Show that the type $\sum_{(n:\mathbb{N})} P(n)$ is decidable.
(b) Construct a function

$$\left(\sum_{(n:\mathbb{N})} P(n)\right) \to \left(\sum_{(n:\mathbb{N})} P(n) \times \mathsf{is\text{-}upper\text{-}bound}_P(n)\right).$$

(c) Use the function of part (b) to give a second construction of the greatest common divisor, and verify that it satisfies the specification of Definition 8.4.1.

8.11 (a) For any three natural numbers x, y, and z, show that the type

$$\sum_{(k:\mathbb{N})} \sum_{(l:\mathbb{N})} \mathsf{dist}_{\mathbb{N}}(kx, ly) = z$$

is decidable.
(b) (Bézout's identity) For any two natural numbers x and y, construct two natural numbers k and l equipped with an identification

$$\mathsf{dist}_{\mathbb{N}}(kx, ly) = \mathsf{gcd}(x, y).$$

8.12 (a) Show that every natural number $n \geq 2$ has a prime factor.
(b) Define a function

$$\mathsf{prime\text{-}factors} : \left(\sum_{(n:\mathbb{N})} 2 \leq n\right) \to \mathsf{list}(\mathbb{N})$$

such that $\mathsf{prime\text{-}factors}(n)$ is an increasing list of primes, and n is the product of the primes in the list $\mathsf{prime\text{-}factors}(n)$.
(c) Show that any increasing list l of primes of which the product is n is equal to the list $\mathsf{prime\text{-}factors}(n)$.

8.13 Show that there are infinitely many primes $p \equiv 3 \mod 4$.

8.14 Show that for each prime p, the ring $\mathbb{Z}_p$ of integers modulo p is a field, i.e., construct a multiplicative inverse

$$(-)^{-1} : \prod_{(x:\mathbb{Z}_p)} \to (x \neq 0) \to \mathbb{Z}_p$$

equipped with identifications

$$x^{-1}x = 1 \qquad\qquad xx^{-1} = 1.$$

8.15 Let $F : \mathbb{N} \to \mathbb{N}$ be the Fibonacci sequence. Construct a function $G : \mathbb{N} \to \mathbb{N}$ such that

$$(G_m \mid n) \leftrightarrow (m \mid F_n)$$

for all $m, n : \mathbb{N}$. Hint: for $m > 0$, G_m is the least $x > 0$ such that $m \mid F_x$.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

II

The Univalent Foundations for Mathematics

The univalent foundations program is an approach to mathematics in which mathematics is formalized in dependent type theory, using the homotopy interpretation and the univalence axiom. Recall that with the homotopy interpretation, we think of types as spaces, type families as fibrations, and identifications as paths. The univalence axiom characterizes the identity type of the universes in type theory, asserting that for any two types A and B in a universe $\mathcal{U}$, we have an equivalence

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

In other words, identifications of types are equivalent to equivalences of types. A consequence of the univalence axiom is that many kinds of isomorphic objects in mathematics, such as isomorphic groups or isomorphic rings, can be identified.

The concept of equivalences generalizes the concept of set-isomorphisms to type theory in a way that is suitable for the homotopy interpretation of type theory. Equivalent types are the same for all practical purposes, just as isomorphic objects are practically the same in everyday mathematics. By the univalence axiom, isomorphic objects get identified.

However, the informal practice of identifying isomorphic objects is technically inconsistent with the set theoretic foundations of mathematics. The extensionality axiom of Zermelo-Fraenkel set theory implies, for instance, that there are many different singleton sets $\{x\}$. All those singleton sets are isomorphic, so the univalence axiom would identify them. We conclude that the univalence axiom would be inconsistent within Zermelo-Fraenkel set theory, and therefore we depart definitively from the set-theoretic foundations by assuming the univalence axiom.

Since the univalence axiom characterizes the identity type of the universe, it is important to understand the general task of characterizing the identity type of any given type. It is a crucial observation, which we already made when we

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

discussed the uniqueness of refl in Section 5.5, that for any $a : A$, the type

$$\sum_{(x:A)} a = x$$

is contractible. Contractible types are types that are singletons up to homotopy, i.e., they are types A that come equipped with a point $a : A$ such that $a = x$ for every $x : A$. We have seen in Proposition 5.5.1 that the total space of all paths starting at a is such a type, so it is an example of a contractible type. The fundamental theorem of identity types asserts that a type family B over A with $b : B(a)$ has a contractible total space

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

if and only if $(a = x) \simeq B(x)$ for all $x : A$. The fundamental theorem of identity types can be used to characterize the identity types of virtually any type that we will encounter. Since types are only fully understood if we also have a clear understanding of their identity types, it is one of the core tasks of a homotopy type theorist to characterize identity types, and the fundamental theorem (Theorem 11.2.2) is the main tool.

Not all types have very complicated identity types. For example, some types have the property that all their identity types are contractible. For example, the types $\emptyset$ and $\mathbf{1}$ satisfy this condition. Any two terms of such a type can therefore be identified, so in this sense they are *proof irrelevant*. The only thing that matters about such types is whether or not they are inhabited by a term. Analogously, this is also the case for propositions in the propositional calculus or first order logic. Therefore we call such types propositions, and we see that propositions are present in type theory as certain types.

Next, there are the types of which the identity types are propositions. In other words, the identity types of such types have the property of proof irrelevance. We are familiar with this situation from set theory, because equality in set theory is a proposition. Therefore we call such types sets. The types $\mathbb{N}$, $\mathbb{Z}$, and Fin_k are all examples of sets.

It is now clear that there is a hierarchy arising: at the bottom of the hierarchy we have the contractible types; then we have the propositions, of which the identity types are contractible; after the propositions we have the sets, of which the identity types are propositions. Defining sets to be of truncation level 0, we define a type to be of truncation level $k + 1$ if its identity types are of truncation level k. Types of truncation level k for $k \geq 1$ are also called k-types or k-groupoids.

This hierarchy of truncation levels is due to Voevodsky, who recognized that, when you are formalizing mathematics in type theory, it is important to specify the truncation level in which you are working. Most mathematics, for example,

takes place at truncation level 0, the level of sets. Groups, rings, posets, and so on are all set-level objects. Categories, on the other hand, are objects of truncation level 1, the level of the 1-groupoids. This is because two objects in a category are considered equal if they are isomorphic, and between any two objects in a category there is a set of isomorphisms.

The fundamental theorem of identity types and the basic facts about truncation levels are proved without assuming any axioms. In other words, they are theorems of Martin-Löf's dependent type theory, as introduced in Chapter I. In particular, the rules of dependent type theory are sufficient to characterize the identity types of Σ-types and of the type of natural numbers, and also to prove the disjointness of coproducts. However, there are still two important characterizations of identity types missing: those of Π-types and those of universes. For those two cases we need axioms:

(i) For any two dependent functions $f, g : \prod_{(x:A)} B(x)$, the canonical map

$$(f = g) \to (f \sim g)$$

that maps refl_f to the constant homotopy, is an equivalence.

(ii) For any two types A and B in a universe $\mathcal{U}$, the canonical map

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

that maps refl_A to the identity equivalence, is an equivalence.

The function extensionality axiom (i) characterizes the identity types of Π-types, and the univalence axiom (ii) characterizes the identity types of universes.

With the addition of the function extensionality axiom and the univalence axiom, we have almost fully specified the univalent foundations for mathematics. The one ingredient missing is that of quotients. In order to obtain quotients, we will postulate two more axioms:

(iii) We will assume that every type A has a propositional truncation.

(iv) We will assume the type theoretic *replacement axiom*.

Propositional truncations are the simplest kind of quotients, identifying all the elements in a type A. In other words, the propositional truncation of a type A is a proposition $\|A\|$ that is true if and only if A is inhabited. Using propositional truncations we can construct the homotopy image of a map. A quotient of a type A by an equivalence relation R can then be constructed as the type of all equivalence classes of R, i.e., as the image of the map $R : A \to (A \to \mathcal{U})$. With this construction of the quotient, we immediately obtain a surjective map

$q : A \to A/R$, and a striking application of the univalence axiom ensures that the quotient is *effective*, i.e., that for any $x, y : A$ we have

$$(q(x) = q(y)) \simeq R(x, y).$$

However, this construction does not guarantee that the quotient A/R is small with respect to the universe $\mathcal{U}$, because it is constructed as a subtype of the type $A \to \mathcal{U}$. This is why we assume the replacement axiom, which will imply that the quotient A/R is *essentially* small. Essentially small types are types that are equivalent to a small type, and the replacement axiom asserts that if $f : A \to B$ is a map from an essentially small type A into a type B of which the identity types are essentially small, then the image of f is also essentially small. The role of the replacement axiom in type theory is similar to the role of the replacement axiom in Zermelo-Fraenkel set theory: to ensure that quotients are small.

9 Equivalences

We introduce equivalences in this section as functions that have a left inverse and a separate right inverse. This choice of definition might seem a little strange: why would we not say that an equivalence map is a map $f : A \to B$ for which there is a map $g : B \to A$ that is at the same time a left and a right inverse? We will be able to show, after all, that if a map has separate left and right inverses, then it has an inverse. For a precise answer to this question we will have to wait until **??**, but we can say already that it turns out to be important that the condition of being an equivalence is a property, not structure. We have chose the definition of equivalences with a separate left and right inverses so that being an equivalence will indeed be a property.

9.1 Homotopies

Remark 9.1.1 Consider the negation function $\mathsf{neg\text{-}bool} : \mathsf{bool} \to \mathsf{bool}$ on the booleans, which was defined in Exercise 4.2. Type theory does not provide any means to show that

$$\mathsf{neg\text{-}bool} \circ \mathsf{neg\text{-}bool} = \mathsf{id}.$$

The best we can do is to construct an identification

$$\mathsf{neg\text{-}neg\text{-}bool}(b) : \mathsf{neg\text{-}bool}(\mathsf{neg\text{-}bool}(b)) = b$$

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

for any $b : \mathsf{bool}$. Indeed, neg-neg-bool is defined using the induction principle of bool, by

$$\begin{aligned}\mathsf{neg\text{-}neg\text{-}bool}(\mathsf{true}) &:\equiv \mathsf{refl}_{\mathsf{true}}\\ \mathsf{neg\text{-}neg\text{-}bool}(\mathsf{false}) &:\equiv \mathsf{refl}_{\mathsf{false}}.\end{aligned}$$

Therefore we see that, while we cannot identify neg-bool $\circ$ neg-bool with id, we can define a *pointwise identification* between the values of neg-bool $\circ$ neg-bool and id.

The observations in Remark 9.1.1 are an instance of a general phenomenon in type theory: it is often much easier to construct a *pointwise identification* between the values of two maps, than it is to construct an identification between those two maps. In fact, the prevalent notion of sameness of maps is the notion of pointwise identifications. Since they are so important, we will give them a name and call them *homotopies*.

Definition 9.1.2 Let $f, g : \prod_{(x:A)} B(x)$ be two dependent functions. The type of **homotopies** from f to g is defined as the type of pointwise identifications, i.e., we define

$$f \sim g :\equiv \prod_{(x:A)} f(x) = g(x).$$

Example 9.1.3 By Remark 9.1.1 we have a homotopy

$$\mathsf{neg\text{-}neg\text{-}bool} : \mathsf{neg\text{-}bool} \circ \mathsf{neg\text{-}bool} \sim \mathsf{id}.$$

Remark 9.1.4 We will use homotopies, for example, to express the commutativity of diagrams. For example, we say that a triangle

$$\begin{array}{ccc} A & \xrightarrow{h} & B \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle g} & \\ & X & \end{array}$$

commutes if it comes equipped with a homotopy $H : f \sim g \circ h$. Similarly, we say that a square

$$\begin{array}{ccc} A & \xrightarrow{g} & A' \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f'} \\ B & \xrightarrow[h]{} & B' \end{array}$$

commutes if it comes equipped with a homotopy $h \circ f \sim f' \circ g$.

Note that the type of homotopies $f \sim g$ is defined for dependent functions, and moreover the type of homotopies is itself a dependent function type. The

definition of homotopies is therefore set up in such a way that we may also consider homotopies *between* homotopies, and even further homotopies between those higher homotopies. More concretely, if $H, K : f \sim g$ are two homotopies, then the type of homotopies $H \sim K$ between them is just the type

$$\prod_{(x:A)} H(x) = K(x).$$

Since homotopies are pointwise identifications, we can use the groupoidal structure of identity types to also define the groupoidal structure of homotopies. In this case, however, we state the groupoid laws as *homotopies* and *homotopies between homotopies* rather than as identifications.

Definition 9.1.5 For any type family B over A we define the operations on homotopies

$$\begin{aligned}
\mathsf{refl\text{-}htpy} &: \prod_{(f:\prod_{(x:A)} B(x))} f \sim f \\
\mathsf{inv\text{-}htpy} &: \prod_{(f,g:\prod_{(x:A)} B(x))} (f \sim g) \to (g \sim f) \\
\mathsf{concat\text{-}htpy} &: \prod_{(f,g,h:\prod_{(x:A)} B(x))} (f \sim g) \to ((g \sim h) \to (f \sim h))
\end{aligned}$$

pointwise by

$$\begin{aligned}
\mathsf{refl\text{-}htpy}(f) &:\equiv \lambda x.\, \mathsf{refl}_{f(x)} \\
\mathsf{inv\text{-}htpy}(H) &:\equiv \lambda x.\, H(x)^{-1} \\
\mathsf{concat\text{-}htpy}(H, K) &:\equiv \lambda x.\, H(x) \cdot K(x).
\end{aligned}$$

We will often write H^{-1} for $\mathsf{inv\text{-}htpy}(H)$, and $H \cdot K$ for $\mathsf{concat\text{-}htpy}(H, K)$.

Proposition 9.1.6 *Homotopies satisfy the groupoid laws:*

(i) *Concatenation of homotopies is associative up to homotopy, i.e., there is a homotopy*

$$\mathsf{assoc\text{-}htpy}(H, K, L) : (H \cdot K) \cdot L \sim H \cdot (K \cdot L)$$

for any homotopies $H : f \sim g$, $K : g \sim h$ and $L : h \sim i$.

(ii) *Homotopies satisfy the left and right unit laws up to homotopy, i.e., there are homotopies*

$$\begin{aligned}
\mathsf{left\text{-}unit\text{-}htpy}(H) &: \mathsf{refl\text{-}htpy}_f \cdot H \sim H \\
\mathsf{right\text{-}unit\text{-}htpy}(H) &: H \cdot \mathsf{refl\text{-}htpy}_g \sim H
\end{aligned}$$

for any homotopy H.

(iii) *Homotopies satisfy the left and right inverse laws up to homotopy, i.e., there are homotopies*

$$\mathsf{left\text{-}inv\text{-}htpy}(H) : H^{-1} \cdot H \sim \mathsf{refl\text{-}htpy}_g$$

$$\mathsf{right\text{-}inv\text{-}htpy}(H) : H \cdot H^{-1} \sim \mathsf{refl\text{-}htpy}_f$$

for any homotopy H.

Proof The homotopy $\mathsf{assoc\text{-}htpy}(H, K, L)$ is defined pointwise by

$$\mathsf{assoc\text{-}htpy}(H, K, L, x) :\equiv \mathsf{assoc}(H(x), K(x), L(x)).$$

The other homotopies are similarly defined pointwise. □

Apart from the groupoid operations and their laws, we will occasionally need *whiskering* operations. Whiskering operations are operations that allow us to compose homotopies with functions. There are two situations where we want this:

Definition 9.1.7 We define the following **whiskering** operations on homotopies:

(i) Suppose $H : f \sim g$ for two functions $f, g : A \to B$, and let $h : B \to C$. We define

$$h \cdot H :\equiv \lambda x.\, \mathsf{ap}_h(H(x)) : h \circ f \sim h \circ g.$$

(ii) Suppose $f : A \to B$ and $H : g \sim h$ for two functions $g, h : B \to C$. We define

$$H \cdot f :\equiv \lambda x.\, H(f(x)) : h \circ f \sim g \circ f.$$

9.2 Bi-invertible maps

We use homotopies to define sections and retractions of a map f, and to define what it means for a map f to be an equivalence.

Definition 9.2.1 Let $f : A \to B$ be a function.

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

$$\mathsf{sec}(f) :\equiv \textstyle\sum_{(g:B\to A)} f \circ g \sim \mathsf{id}_B.$$

In other words, a **section** of f is a map $g : B \to A$ equipped with a homotopy $f \circ g \sim \mathsf{id}$.

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

$$\mathsf{retr}(f) :\equiv \textstyle\sum_{(h:B\to A)} h \circ f \sim \mathsf{id}_A.$$

If a map $f : A \to B$ has a retraction, we also say that A is a **retract** of B.

(iii) We say that a function $f : A \to B$ is an **equivalence** if it has both a section and a retraction, i.e., if it comes equipped with an element of type

$$\mathsf{is\text{-}equiv}(f) :\equiv \mathsf{sec}(f) \times \mathsf{retr}(f).$$

We will write $A \simeq B$ for the type $\sum_{(f:A\to B)} \mathsf{is\text{-}equiv}(f)$ of all equivalences from A to B. For any equivalence $e : A \simeq B$ we define e^{-1} to be the section of e.

Remark 9.2.2 An equivalence, as we defined it here, can be thought of as a *bi-invertible map*, since it comes equipped with a separate left and right inverse. Explicitly, if f is an equivalence, then there are

$$g : B \to A \qquad h : B \to A$$
$$G : f \circ g \sim \mathsf{id}_B \qquad H : h \circ f \sim \mathsf{id}_A.$$

Example 9.2.3 For any type A, the identity function $\mathsf{id} : A \to A$ is an equivalence, since it is its own section and its own retraction

Example 9.2.4 Since we have seen in Remark 9.1.1 that the negation function $\mathsf{neg\text{-}bool} : \mathsf{bool} \to \mathsf{bool}$ on the booleans is its own inverse, it follows that $\mathsf{neg\text{-}bool}$ is an equivalence.

Example 9.2.5 The successor and predecessor functions on $\mathbb{Z}$ are equivalences by Exercise 5.6. Furthermore, the function

$$x \mapsto x + k$$

is an equivalence from $\mathbb{Z}$ to $\mathbb{Z}$, for each $k : \mathbb{Z}$. This follows from the group laws on $\mathbb{Z}$, proven in Exercise 5.7. Indeed, the inverse of $x \mapsto x + k$ is the map $x \mapsto x + (-k)$. Finally, it also follows from the group laws on $\mathbb{Z}$ that the map $x \mapsto -x$ is an equivalence.

The same holds for the finite types: the maps succ_k, pred_k, $\mathsf{add}_k(x)$ and neg_k are all equivalences on Fin_k.

Remark 9.2.6 More generally, if f has an inverse in the sense that we have a function $g : B \to A$ equipped with homotopies $f \circ g \sim \mathsf{id}_B$ and $g \circ f \sim \mathsf{id}_A$, then f is an equivalence. We write

$$\mathsf{has\text{-}inverse}(f) :\equiv \textstyle\sum_{(g:B\to A)} (f \circ g \sim \mathsf{id}_B) \times (g \circ f \sim \mathsf{id}_A).$$

However, we did *not* define equivalences to be functions that have inverses. The reason is that we would like that being an equivalence is a *property*, not a non-trivial structure on the map f. This fact requires the function extensionality axiom, but we can already say that if a map f is an equivalence, then it has up to homotopy only one section and only one retraction (see Exercise 13.4).

The type $\mathsf{has\text{-}inverse}(f)$ on the other hand, turns out to be homotopically complicated. In Exercise 21.6 we will see that the identity function $\mathsf{id}_{\mathbf{S}^1} : \mathbf{S}^1 \to \mathbf{S}^1$ on the circle is an example of a map for which

$$\mathsf{has\text{-}inverse}(\mathsf{id}_{\mathbf{S}^1}) \simeq \mathbb{Z}.$$

Even though $\mathsf{is\text{-}equiv}(f)$ and $\mathsf{has\text{-}inverse}(f)$ can be wildly different types, there are maps back and forth between the two. We have already observed in Remark 9.2.6 that there is a map

$$\mathsf{has\text{-}inverse}(f) \to \mathsf{is\text{-}equiv}(f).$$

The following proposition gives the converse implication.

Proposition 9.2.7 *Any map $f : A \to B$ which is an equivalence, can be given the structure of an invertible map i.e., there is a map*

$$\mathsf{is\text{-}equiv}(f) \to \mathsf{has\text{-}inverse}(f).$$

Proof First we construct for any equivalence f with right inverse g and left inverse h a homotopy $K : g \sim h$. For any $y : B$, we have

$$g(y) \xrightarrow{H(g(y))^{-1}} hfg(y) \xrightarrow{\mathsf{ap}_h(G(y))} h(y).$$

In other words, the homotopy $K : g \sim h$ is defined to be $(H \cdot g)^{-1} \cdot (h \cdot G)$. Using the homotopy K we are able to show that g is also a left inverse of f. For $x : A$ we have the identification

$$gf(x) \xrightarrow{K(f(x))} hf(x) \xrightarrow{H(x)} x. \qquad \square$$

Corollary 9.2.8 *The inverse of an equivalence is again an equivalence.*

Proof Let $f : A \to B$ be an equivalence. By Proposition 9.2.7 it follows that the section of f is also a retraction. Therefore it follows that the section is itself an invertible map, with inverse f. Hence it is an equivalence. $\square$

Example 9.2.9 Types, just as sets in classical mathematics, satisfy the usual laws of coproducts and products, such as unit laws, commutativity, and associativity. These laws are formulated as equivalences:

$$\begin{aligned}
\emptyset + B &\simeq B & A + \emptyset &\simeq A \\
A + B &\simeq B + A & (A + B) + C &\simeq A + (B + C) \\
\emptyset \times B &\simeq \emptyset & A \times \emptyset &\simeq \emptyset \\
\mathbf{1} \times B &\simeq B & A \times \mathbf{1} &\simeq A \\
A \times B &\simeq B \times A & (A \times B) \times C &\simeq A \times (B \times C)
\end{aligned}$$

$$A \times (B + C) \simeq (A \times B) + (A \times C) \qquad (A + B) \times C \simeq (A \times C) + (B \times C).$$

All of these equivalences are constructed in a similar way: the maps back and forth as well as the required homotopies are constructed using induction, or, more efficiently, using pattern matching. For example, to show that cartesian products distribute from the left over coproducts, we construct maps

$$\alpha : A \times (B + C) \to (A \times B) + (A \times C)$$
$$\beta : (A \times B) + (A \times C) \to A \times (B + C)$$

as follows:

$$\alpha(x, \mathsf{inl}(y)) :\equiv \mathsf{inl}(x, y) \qquad \beta(\mathsf{inl}(x, y)) :\equiv (x, \mathsf{inl}(y))$$
$$\alpha(x, \mathsf{inr}(z)) :\equiv \mathsf{inr}(x, z) \qquad \beta(\mathsf{inr}(x, z)) :\equiv (x, \mathsf{inr}(z)).$$

The homotopies $G : \alpha \circ \beta \sim \mathsf{id}$ and $H : \beta \circ \alpha \sim \mathsf{id}$ are then defined by

$$G(\mathsf{inl}(x, y)) :\equiv \mathsf{refl} \qquad H(x, \mathsf{inl}(y)) :\equiv \mathsf{refl}$$
$$G(\mathsf{inr}(x, z)) :\equiv \mathsf{refl} \qquad H(x, \mathsf{inr}(z)) :\equiv \mathsf{refl}.$$

We encourage the reader to write out the definitions of at least a few of these equivalences.

Example 9.2.10 The laws for cartesian products can be generalized to arbitrary Σ-types. The absorption laws and unit laws, for instance, are as follows:

$$\textstyle\sum_{(x:\emptyset)} B(x) \simeq \emptyset \qquad \sum_{(x:A)} \emptyset \simeq \emptyset$$
$$\textstyle\sum_{(x:\mathbf{1})} B(x) \simeq B(\star) \qquad \sum_{(x:A)} \mathbf{1} \simeq A.$$

Note that the right absorption law and the right unit law are exactly the same as the right absorption and unit laws for cartesian products. The left absorption and unit laws are, however, formulated with a type family B over $\emptyset$ and over $\mathbf{1}$, and therefore they are slightly more general.

Commutativity cannot be generalized to Σ-types. Associativity, on the other hand, can be expressed in two ways:

$$\textstyle\sum_{(w:\sum_{(x:A)} B(x))} C(w) \simeq \sum_{(x:A)} \sum_{(y:B)} C(\mathsf{pair}(x, y))$$
$$\textstyle\sum_{(w:\sum_{(x:A)} B(x))} C(\mathsf{pr}_1(w), \mathsf{pr}_2(w)) \simeq \sum_{(x:A)} \sum_{(y:B(x))} C(x, y).$$

In the first of these equivalences associativity is stated using a type family C over $\sum_{(x:A)} B(x)$ while in the second it is stated using a family of types $C(x, y)$ indexed by $x : A$ and $y : B(x)$.

Finally, we note that Σ also distributes over coproducts. In other words, there

are the following two equivalences:

$$\sum_{(x:A)} B(x) + C(x) \simeq \left(\sum_{(x:A)} B(x)\right) + \left(\sum_{(x:A)} C(x)\right)$$
$$\sum_{(w:A+B)} C(w) \simeq \left(\sum_{(x:A)} C(\mathsf{inl}(x))\right) + \left(\sum_{(y:B)} C(\mathsf{inr}(y))\right).$$

Remark 9.2.11 We haven't stated any laws involving function types or dependent function types, because it requires the function extensionality principle to prove them.

9.3 Characterizing the identity types of Σ-types

In this section we characterize the identity type of a Σ-type as a Σ-type of identity types. Characterizing identity types is a task that a homotopy type theorist routinely performs, so we will follow the general outline of how such a characterization goes:

(i) First we define a binary relation $R : A \to A \to \mathcal{U}$ on the type A that we are interested in. This binary relation is intended to be equivalent to its identity type.
(ii) Then we will show that this binary relation is reflexive, by constructing a dependent function of type

$$\prod_{(x:A)} R(x, x)$$

(iii) Using the reflexivity we will show that there is a canonical map

$$(x = y) \to R(x, y)$$

for every $x, y : A$. This map is just constructed by path induction, using the reflexivity of R.
(iv) Finally, it has to be shown that the map

$$(x = y) \to R(x, y)$$

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

The last step is usually the most difficult, and we will refine our methods for this step in Section 11, where we establish the fundamental theorem of identity types.

In this section we consider a type family B over A. Given two pairs

$$(x, y), (x', y') : \sum_{(x:A)} B(x),$$

if we have a path $\alpha : x = x'$ then we can compare $y : B(x)$ to $y' : B(x')$ by first transporting y along α, i.e., we consider the identity type

$$\mathsf{tr}_B(\alpha, y) = y'.$$

Thus it makes sense to think of (x, y) to be identical to (x', y') if there is an identification $\alpha : x = x'$ and an identification $\beta : \mathsf{tr}_B(\alpha, y) = y'$. In the following definition we turn this idea into a binary relation on the Σ-type.

Definition 9.3.1 We will define a relation

$$\mathsf{Eq}_\Sigma : \left(\textstyle\sum_{(x:A)} B(x)\right) \to \left(\textstyle\sum_{(x:A)} B(x)\right) \to \mathcal{U}$$

by defining

$$\mathsf{Eq}_\Sigma(s, t) :\equiv \textstyle\sum_{(\alpha:\mathsf{pr}_1(s)=\mathsf{pr}_1(t))} \mathsf{tr}_B(\alpha, \mathsf{pr}_2(s)) = \mathsf{pr}_2(t).$$

Lemma 9.3.2 *The relation* Eq_Σ *is reflexive, i.e., we can construct*

$$\mathsf{reflexive\text{-}Eq}_\Sigma : \textstyle\prod_{(s:\sum_{(x:A)} B(x))} \mathsf{Eq}_\Sigma(s, s).$$

Construction The element $\mathsf{reflexive\text{-}Eq}_\Sigma$ is constructed by Σ-induction on $s : \sum_{(x:A)} B(x)$. Thus, it suffices to construct a dependent function of type

$$\textstyle\prod_{(x:A)} \prod_{(y:B(x))} \sum_{(\alpha:x=x)} \mathsf{tr}_B(\alpha, y) = y.$$

Here we take $\lambda x.\, \lambda y.\, (\mathsf{refl}_x, \mathsf{refl}_y)$. □

Definition 9.3.3 Consider a type family B over A. Then for any $s, t : \sum_{(x:A)} B(x)$ we define a map

$$\mathsf{pair\text{-}eq} : (s = t) \to \mathsf{Eq}_\Sigma(s, t)$$

by path induction, taking $\mathsf{pair\text{-}eq}(\mathsf{refl}_s) :\equiv \mathsf{reflexive\text{-}Eq}_\Sigma(s)$.

Theorem 9.3.4 *Let B be a type family over A. Then the map*

$$\mathsf{pair\text{-}eq} : (s = t) \to \mathsf{Eq}_\Sigma(s, t)$$

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

Proof The maps in the converse direction

$$\mathsf{eq\text{-}pair} : \mathsf{Eq}_\Sigma(s, t) \to (s = t)$$

are defined by repeated Σ-induction. By Σ-induction on s and t we see that it suffices to define a map

$$\mathsf{eq\text{-}pair} : \left(\textstyle\sum_{(p:x=x')} \mathsf{tr}_B(p, y) = y'\right) \to ((x, y) = (x', y')).$$

A map of this type is again defined by Σ-induction. Thus it suffices to define a dependent function of type

$$\prod_{(p:x=x')} (\mathsf{tr}_B(p, y) = y') \to ((x, y) = (x', y')).$$

Such a dependent function is defined by double path induction by sending $(\mathsf{refl}_x, \mathsf{refl}_y)$ to $\mathsf{refl}_{(x,y)}$. This completes the definition of the function eq-pair.

Next, we must show that eq-pair is a section of pair-eq. In other words, we must construct an identification

$$\mathsf{pair\text{-}eq}(\mathsf{eq\text{-}pair}(\alpha, \beta)) = (\alpha, \beta)$$

for each $(\alpha, \beta) : \sum_{(\alpha:x=x')} \mathsf{tr}_B(\alpha, y) = y'$. We proceed by path induction on α, followed by path induction on β. Then our goal becomes to construct an identification of type

$$\mathsf{pair\text{-}eq}(\mathsf{eq\text{-}pair}(\mathsf{refl}_x, \mathsf{refl}_y)) = (\mathsf{refl}_x, \mathsf{refl}_y)$$

By the definition of eq-pair we have $\mathsf{eq\text{-}pair}(\mathsf{refl}_x, \mathsf{refl}_y) \doteq \mathsf{refl}_{(x,y)}$, and by the definition of pair-eq we have $\mathsf{pair\text{-}eq}(\mathsf{refl}_{(x,y)}) \doteq (\mathsf{refl}_x, \mathsf{refl}_y)$. Thus we may take $\mathsf{refl}_{(\mathsf{refl}_x, \mathsf{refl}_y)}$ to complete the construction of the homotopy $\mathsf{pair\text{-}eq} \circ \mathsf{eq\text{-}pair} \sim \mathsf{id}$.

To complete the proof, we must show that eq-pair is a retraction of pair-eq. In other words, we must construct an identification

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

for each $p : s = t$. We proceed by path induction on $p : s = t$, so it suffices to construct an identification

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

Now we proceed by Σ-induction on $s : \sum_{(x:A)} B(x)$, so it suffices to construct an identification

$$\mathsf{eq\text{-}pair}(\mathsf{refl}_x, \mathsf{refl}_y) = \mathsf{refl}_{(x,y)}.$$

Since $\mathsf{eq\text{-}pair}(\mathsf{refl}_x, \mathsf{refl}_y)$ computes to $\mathsf{refl}_{(x,y)}$, we may simply take $\mathsf{refl}_{\mathsf{refl}_{(x,y)}}$. □

Exercises

9.1 Show that the functions

$$\begin{aligned} \mathsf{inv} &: (x = y) \to (y = x) \\ \mathsf{concat}(p) &: (y = z) \to (x = z) \\ \mathsf{concat}'(q) &: (x = y) \to (x = z) \end{aligned}$$

$$\mathsf{tr}_B(p) : B(x) \to B(y)$$

are equivalences, where $\mathsf{concat}'(q, p) :\equiv p \cdot q$. Give their inverses explicitly.

9.2 (a) Use Exercise 6.2 (c) to show that the constant function $\mathsf{const}_b : \mathsf{bool} \to \mathsf{bool}$ is not an equivalence, for any $b : \mathsf{bool}$.

(b) Show that $\mathsf{bool} \neq \mathbf{1}$.

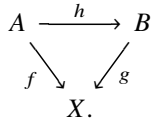
(c) Show that $\mathbb{N} \neq \mathsf{Fin}_k$ for any $k : \mathbb{N}$.

9.3 (a) Consider two functions $f, g : A \to B$ and a homotopy $H : f \sim g$. Then

$$\mathsf{is\text{-}equiv}(f) \leftrightarrow \mathsf{is\text{-}equiv}(g).$$

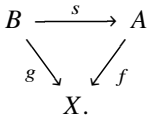
(b) Show that for any two homotopic equivalences $e, e' : A \simeq B$, their inverses are also homotopic.

9.4 Consider a commuting triangle



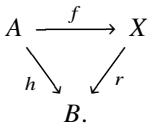
with $H : f \sim g \circ h$.

(a) Suppose that the map h has a section $s : B \to A$. Show that the triangle



commutes, and that f has a section if and only if g has a section.

(b) Suppose that the map g has a retraction $r : X \to B$. Show that the triangle



commutes, and that f has a retraction if and only if h has a retraction.

(c) (The **3-for-2 property** for equivalences.) Show that if any two of the functions

$$f, \qquad g, \qquad h$$

are equivalences, then so is the third. Conclude that any section and any retraction of an equivalence is again an equivalence.

9.5 (a) Let A and B be types, and let C be a family over $x : A, y : B$. Construct an equivalence

$$\left(\sum_{(x:A)}\sum_{(y:B)} C(x,y)\right) \simeq \left(\sum_{(y:B)}\sum_{(x:A)} C(x,y)\right).$$

(b) Let A be a type, and let B and C be type families over A. Construct an equivalence

$$\left(\sum_{(u:\sum_{(x:A)} B(x))} C(\mathsf{pr}_1(u))\right) \simeq \left(\sum_{(v:\sum_{(x:A)} C(x))} B(\mathsf{pr}_1(v))\right).$$

9.6 In this exercise we will construct the **functorial action** of coproducts.

(a) Construct for any two maps $f : A \to A'$ and $g : B \to B'$, a map

$$f + g : A + B \to A' + B'.$$

(b) Show that $\mathsf{id}_A + \mathsf{id}_B \sim \mathsf{id}_{A+B}$.

(c) Show that for any two pairs of composable functions

$$A \xrightarrow{f} A' \xrightarrow{f'} A'' \qquad \text{and} \qquad B \xrightarrow{g} B' \xrightarrow{g'} B''$$

there is a homotopy $(f' \circ f) + (g' \circ g) \sim (f' + g') \circ (f + g)$.

(d) Show that if $H : f \sim f'$ and $K : g \sim g'$, then there is a homotopy

$$H + K : (f + g) \sim (f' + g').$$

(e) Show that if both f and g are equivalences, then so is $f + g$. (The converse of this statement also holds, see Exercise 11.7.)

9.7 (a) Construct for any two maps $f : A \to A'$ and $g : B \to B'$, a map

$$f \times g : A \times B \to A' \times B'$$

(b) Show that $\mathsf{id}_A \times \mathsf{id}_B \sim \mathsf{id}_{A\times B}$.

(c) Show that for any two pairs of composable functions

$$A \xrightarrow{f} A' \xrightarrow{f'} A'' \qquad \text{and} \qquad B \xrightarrow{g} B' \xrightarrow{g'} B''$$

there is a homotopy $(f' \circ f) \times (g' \circ g) \sim (f' \times g') \circ (f \times g)$.

(d) Show that if $H : f \sim f'$ and $K : g \sim g'$, then there is a homotopy

$$H \times K : (f \times g) \sim (f' \times g').$$

(e) Show that for any two maps $f : A \to A'$ and $g : B \to B'$, the following are equivalent:

(i) The map $f \times g$ is an equivalence.

(ii) There are functions

$$\alpha : B \to \mathsf{is\text{-}equiv}(f)$$
$$\beta : A \to \mathsf{is\text{-}equiv}(g).$$

9.8 Construct equivalences

$$\mathsf{Fin}_{k+l} \simeq \mathsf{Fin}_k + \mathsf{Fin}_l$$
$$\mathsf{Fin}_{kl} \simeq \mathsf{Fin}_k \times \mathsf{Fin}_l.$$

9.9 A map $f : X \to X$ is said to be **finitely cyclic** if it comes equipped with an element of type

$$\mathsf{is\text{-}finitely\text{-}cyclic}(f) :\equiv \prod_{(x,y:X)} \sum_{(k:\mathbb{N})} f^k(x) = y.$$

(a) Show that any finitely cyclic map is an equivalence.
(b) Show that $\mathsf{succ} : \mathsf{Fin}_k \to \mathsf{Fin}_k$ is finitely cyclic for any $k : \mathbb{N}$.

10 Contractible types and contractible maps

A contractible type is a type which has, up to identification, only one element. In other words, a contractible type is a type that comes equipped with a point, and an identification of this point with any point.

We may think of contractible types as singletons up to homotopy, and indeed we show that the unit type is an example of a contractible type. Moreover, we show that contractible types satisfy an induction principle that is very similar to the induction principle of the unit type.

Another case of an inductive type with a single constructor is the type of identifications $p : a = x$ with a fixed starting point $a : A$. To specify such an identification, we have to give its end point $x : A$ as well as the identification $p : a = x$, and the path induction principle asserts that in order to show something about all such identifications, it suffices to show that thing in the case where the end point is a, and the path is refl_a. This suggests that the total space

$$\sum_{(x:A)} a = x$$

of all paths with starting point $a : A$ is contractible. This important fact will be shown in Theorem 10.1.4, and it is the basis for the fundamental theorem of identity types (Section 11).

Next, we introduce the *fiber* of a map $f : A \to B$. The fiber of f at $b : B$ consists of the type of elements $a : A$ equipped with an identification $p : f(a) = b$. In other words, the fiber of f at b is the preimage of f at

b. In Theorems 10.3.5 and 10.4.6 we show that a map is an equivalence if and only if its fibers are contractible. The condition that the fibers of a map are contractible is analogous to the set theoretic notion of bijective map, or 1-to-1-correspondence.

10.1 Contractible types

Definition 10.1.1 We say that a type A is **contractible** if it comes equipped with an element of type

$$\mathsf{is\text{-}contr}(A) :\equiv \sum_{(c:A)} \prod_{(x:A)} c = x.$$

Given a pair $(c, C) : \mathsf{is\text{-}contr}(A)$, we call $c : A$ the **center of contraction** of A, and we call $C : \prod_{(x:A)} c = x$ the **contraction** of A.

Remark 10.1.2 Suppose A is a contractible type with center of contraction c and contraction C. Then the type of C is (judgmentally) equal to the type

$$\mathsf{const}_c \sim \mathsf{id}_A.$$

In other words, the contraction C is a *homotopy* from the constant function to the identity function.

Example 10.1.3 The unit type is easily seen to be contractible. For the center of contraction we take $\star : \mathbf{1}$. Then we define a contraction $\prod_{(x:\mathbf{1})} \star = x$ by the induction principle of $\mathbf{1}$. Applying the induction principle, it suffices to construct an identification of type $\star = \star$, for which we just take $\mathsf{refl}_\star$.

Theorem 10.1.4 *For any $a : A$, the type*

$$\sum_{(x:A)} a = x$$

is contractible.

Proof For the center of contraction we take

$$(a, \mathsf{refl}_a) : \sum_{(x:A)} a = x.$$

The contraction is constructed in Proposition 5.5.1. □

10.2 Singleton induction

Contractible types are singletons up to homotopy. Indeed, every element of a contractible type can be identified with the center of contraction. Therefore we can prove an induction principle for contractible types that is similar to the induction principle of the unit type.

Definition 10.2.1 Suppose A comes equipped with an element $a : A$. Then we say that A satisfies **singleton induction** if for every type family B over A, the map

$$\mathsf{ev\text{-}pt} : \left(\prod_{(x:A)} B(x)\right) \to B(a)$$

defined by $\mathsf{ev\text{-}pt}(f) :\equiv f(a)$ has a section. In other words, if A satisfies singleton induction we have a function and a homotopy

$$\mathsf{ind\text{-}sing}_a : B(a) \to \prod_{(x:A)} B(x)$$
$$\mathsf{comp\text{-}sing}_a : \mathsf{ev\text{-}pt} \circ \mathsf{ind\text{-}sing}_a \sim \mathsf{id}$$

for any type family B over A.

Example 10.2.2 Note that the singleton induction principle is almost the same as the induction principle for the unit type, the difference being that the 'computation rule' in the singleton induction for A is stated using an *identification* rather than as a judgmental equality. The unit type $\mathbf{1}$ comes equipped with a function

$$\mathsf{ind}_{\mathbf{1}} : B(\star) \to \prod_{(x:\mathbf{1})} B(x)$$

for every type family B over $\mathbf{1}$, satisfying the judgmental equality $\mathsf{ind}_{\mathbf{1}}(b, \star) \doteq b$ for every $b : B(\star)$ by the computation rule. Therefore, we obtain the homotopy

$$\lambda b.\, \mathsf{refl}_b : \mathsf{ev\text{-}pt} \circ \mathsf{ind}_{\mathbf{1}} \sim \mathsf{id},$$

and we conclude that the unit type satisfies singleton induction.

Theorem 10.2.3 *Let A be a type. The following are equivalent:*

(i) *The type A is contractible.*
(ii) *The type A comes equipped with an element $a : A$, and satisfies singleton induction.*

Proof Suppose A is contractible with center of contraction a and contraction C. First we observe that, without loss of generality, we may assume that C comes equipped with an identification $p : C(a) = \mathsf{refl}_a$. To see this, note that we can always define a new contraction C' by

$$C'(x) :\equiv C(a)^{-1} \cdot C(x),$$

which satisfies the requirement by the left inverse law, constructed in Definition 5.2.5.

To show that A satisfies singleton induction let B be a type family over A, and suppose we have $b : B(a)$. Our goal is to define

$$\mathsf{ind\text{-}sing}_a(b) : \prod_{(x:A)} B(x).$$

Let $x : A$. Since we have an identification $C(x) : a = x$, and an element b in $B(a)$, we may transport b along the path $C(x)$ to obtain

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

Therefore, the function $\mathsf{ind\text{-}sing}_a(b)$ is defined to be the dependent function $\lambda x.\, \mathsf{tr}_B(C(x), b)$. Now we have to show that $\mathsf{ind\text{-}sing}_a(b, a) = b$. Then we have the identifications

$$\mathsf{tr}_B(C(a), b) \xrightarrow{\mathsf{ap}_{\lambda\omega.\,\mathsf{tr}_B(\omega,b)}(p)} \mathsf{tr}_B(\mathsf{refl}_a, b) \xrightarrow{\mathsf{refl}_b} b.$$

This shows that the computation rule is satisfied, which completes the proof that A satisfies singleton induction.

For the converse, suppose that $a : A$ and that A satisfies singleton induction. Our goal is to show that A is contractible. For the center of contraction we take the element $a : A$. By singleton induction applied to $B(x) :\equiv a = x$ we have the map

$$\mathsf{ind\text{-}sing}_a : a = a \to \prod_{(x:A)} a = x.$$

Therefore $\mathsf{ind\text{-}sing}_a(\mathsf{refl}_a)$ is a contraction. □

10.3 Contractible maps

Definition 10.3.1 Let $f : A \to B$ be a function, and let $b : B$. The **fiber** of f at b is defined to be the type

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

In other words, the fiber of f at b is the type of $a : A$ that get mapped by f to b. One may think of the fiber as a type theoretic version of the preimage of a point.

It will be useful to have a characterization of the identity type of a fiber. In order to identify any (x, p) and (x', p') in $\mathsf{fib}_f(y)$, we may first construct an identification $\alpha : x = x'$. Then we obtain a triangle

$$\begin{array}{ccc} f(x) & \xrightarrow{\mathsf{ap}_f(\alpha)} & f(x') \\ & {\scriptstyle p}\searrow \quad \swarrow{\scriptstyle p'} & \\ & y, & \end{array}$$

so we may consider the type of identifications $\beta : p = \mathsf{ap}_f(\alpha) \cdot p'$. We will show that the type of all identifications $(x, p) = (x', p')$ is equivalent to the type of such pairs (α, β).

Definition 10.3.2 Let $f : A \to B$ be a map, and let $(x, p), (x', p') : \mathsf{fib}_f(y)$ for some $y : B$. Then we define

$$\mathsf{Eq\text{-}fib}_f((x, p), (x', p')) :\equiv \textstyle\sum_{(\alpha : x = x')} p = \mathsf{ap}_f(\alpha) \cdot p'$$

The relation $\mathsf{Eq\text{-}fib}_f : \mathsf{fib}_f(y) \to \mathsf{fib}_f(y) \to \mathcal{U}$ is a reflexive relation, since we have

$$\lambda(x, p).\, (\mathsf{refl}_x, \mathsf{refl}_p) : \textstyle\prod_{((x,p):\mathsf{fib}_f(y))} \mathsf{Eq\text{-}fib}_f((x, p), (x, p)).$$

Proposition 10.3.3 *Consider a map $f : A \to B$ and let $y : B$. The canonical map*

$$((x, p) = (x', p')) \to \mathsf{Eq\text{-}fib}_f((x, p), (x', p'))$$

induced by the reflexivity of $\mathsf{Eq\text{-}fib}_f$ is an equivalence for any $(x, p), (x', p') : \mathsf{fib}_f(y)$.

Proof The converse map

$$\mathsf{Eq\text{-}fib}_f((x, p), (x', p')) \to ((x, p) = (x', p'))$$

is easily defined by Σ-induction, and then path induction twice. The homotopies witnessing that this converse map is indeed a right inverse as well as a left inverse are similarly constructed by induction. □

Now we arrive at the notion of contractible map.

Definition 10.3.4 We say that a function $f : A \to B$ is **contractible** if it comes equipped with an element of type

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

Theorem 10.3.5 *Any contractible map is an equivalence.*

Proof Let $f : A \to B$ be a contractible map. Using the center of contraction of each $\mathsf{fib}_f(y)$, we obtain the dependent function

$$\lambda y.\, (g(y), G(y)) : \textstyle\prod_{(y:B)} \mathsf{fib}_f(y).$$

Thus, we get map $g : B \to A$, and a homotopy $G : \prod_{(y:B)} f(g(y)) = y$. In other words, we get a section of f.

It remains to construct a retraction of f. Taking g as our retraction, we have to show that $\prod_{(x:A)} g(f(x)) = x$. Note that we get an identification $p : f(g(f(x))) = f(x)$ since g is a section of f. Therefore, it follows that $(g(f(x)), p) : \mathsf{fib}_f(f(x))$. Moreover, since $\mathsf{fib}_f(f(x))$ is contractible we get an identification $q : (g(f(x)), p) = (x, \mathsf{refl}_{f(x)})$. The base path $\mathsf{ap}_{\mathsf{pr}_1}(q)$ of this identification is an identification of type $g(f(x)) = x$, as desired. □

10.4 Equivalences are contractible maps

In Theorem 10.4.6 we will show the converse to Theorem 10.3.5, i.e., we will show that any equivalence is a contractible map. We will do this in two steps.

First we introduce a new notion of *coherently invertible map*, for which we can easily show that such maps have contractible fibers. Then we show that any equivalence is a coherently invertible map.

Recall that an invertible map is a map $f : A \to B$ equipped with $g : B \to A$ and homotopies

$$G : f \circ g \sim \mathsf{id} \qquad \text{and} \qquad H : g \circ f \sim \mathsf{id}.$$

Then we observe that both $G \cdot f$ and $f \cdot H$ are homotopies of the same type

$$f \circ g \circ f \sim f.$$

A coherently invertible map is an invertible map for which there is a further homotopy $G \cdot f \sim f \cdot H$.

Definition 10.4.1 Consider a map $f : A \to B$. We say that f is **coherently invertible** if it comes equipped with

$$\begin{aligned} g &: B \to A \\ G &: f \circ g \sim \mathsf{id} \\ H &: g \circ f \sim \mathsf{id} \\ K &: G \cdot f \sim f \cdot H. \end{aligned}$$

We will write $\mathsf{is\text{-}coh\text{-}invertible}(f)$ for the type of quadruples (g, G, H, K).

Although we will encounter the notion of coherently invertible map on some further occasions, the following proposition is our main motivation for considering it.

Proposition 10.4.2 *Any coherently invertible map has contractible fibers.*

Proof Consider a map $f : A \to B$ equipped with

$$\begin{aligned} g &: B \to A \\ G &: f \circ g \sim \mathsf{id} \\ H &: g \circ f \sim \mathsf{id} \\ K &: G \cdot f \sim f \cdot H, \end{aligned}$$

and let $y : B$. Our goal is to show that $\mathsf{fib}_f(y)$ is contractible. For the center of

contraction we take $(g(y), G(y))$. In order to construct a contraction, it suffices to construct a dependent function of type

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

By path induction on $p : f(x) = y$ it suffices to construct a dependent function of type

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

By definition of $\mathsf{Eq\text{-}fib}_f$, we have to construct for each $x : A$ an identification $\alpha : g(f(x)) = x$ equipped with a further identification

$$G(f(x)) = \mathsf{ap}_f(\alpha) \cdot \mathsf{refl}_{f(x)}.$$

Such a dependent function is constructed as $\lambda x.\, (H(x), K'(x))$, where the homotopy $H : g \circ f \sim \mathsf{id}$ is given by assumption, and the homotopy

$$K' : \prod_{(x:A)} G(f(x)) = \mathsf{ap}_f(H(x)) \cdot \mathsf{refl}_{f(x)}$$

is defined as

$$K' \coloneqq K \cdot \mathsf{right\text{-}unit\text{-}htpy}(f \cdot H)^{-1}. \qquad \square$$

Our next goal is to show that for any map $f : A \to B$ equipped with

$$g : B \to A, \qquad G : f \circ g \sim \mathsf{id}, \qquad \text{and} \qquad H : g \circ f \sim \mathsf{id},$$

we can improve the homotopy G to a new homotopy $G' : f \circ g \sim \mathsf{id}$ for which there is a further homotopy

$$f \cdot H \sim G' \cdot f.$$

Note that this situation is analogous to the situation in the proof of Theorem 10.2.3, where we improved the contraction C so that it satisfied $C(c) = \mathsf{refl}$. The extra coherence $f \cdot H \sim G' \cdot f$ is then used in the proof that the fibers of an equivalence are contractible.

Definition 10.4.3 Let $f, g : A \to B$ be functions, and consider $H : f \sim g$ and $p : x = y$ in A. We define the identification

$$\mathsf{nat\text{-}htpy}(H, p) \coloneqq \mathsf{ap}_f(p) \cdot H(y) = H(x) \cdot \mathsf{ap}_g(p)$$

witnessing that the square

$$\begin{array}{ccc} f(x) & \overset{H(x)}{=\!=} & g(x) \\ {\scriptstyle \mathsf{ap}_f(p)}\Big\| & & \Big\|{\scriptstyle \mathsf{ap}_g(p)} \\ f(y) & \underset{H(y)}{=\!=} & g(y) \end{array}$$

commutes. This square is also called the **naturality square** of the homotopy H at p.

Construction By path induction on p it suffices to construct an identification

$$\mathsf{ap}_f(\mathsf{refl}_x) \cdot H(x) = H(x) \cdot \mathsf{ap}_g(\mathsf{refl}_x)$$

since $\mathsf{ap}_f(\mathsf{refl}_x) \doteq \mathsf{refl}_{f(x)}$ and $\mathsf{ap}_g(\mathsf{refl}_x) \doteq \mathsf{refl}_{g(x)}$, and since $\mathsf{refl}_{f(x)} \cdot H(x) \doteq H(x)$, we see that the path $\mathsf{right\text{-}unit}(H(x))^{-1}$ is of the asserted type. □

Definition 10.4.4 Consider $f : A \to A$ and $H : f \sim \mathsf{id}_A$. We construct an identification $H(f(x)) = \mathsf{ap}_f(H(x))$, for any $x : A$.

Construction By the naturality of homotopies with respect to identifications the square

$$\begin{array}{ccc} ff(x) & \xrightarrow{H(f(x))} & f(x) \\ {\scriptstyle \mathsf{ap}_f(H(x))}\downarrow & & \downarrow{\scriptstyle H(x)} \\ f(x) & \xrightarrow[H(x)]{} & x \end{array}$$

commutes. This gives the desired identification $H(f(x)) = \mathsf{ap}_f(H(x))$. □

Lemma 10.4.5 *Let $f : A \to B$ be a map equipped with an inverse, i.e., consider*

$$\begin{aligned} g &: B \to A \\ G &: f \circ g \sim \mathsf{id} \\ H &: g \circ f \sim \mathsf{id}. \end{aligned}$$

Then there is a homotopy $G' : f \circ g \sim \mathsf{id}$ equipped with a further homotopy

$$K : f \cdot H \sim G' \cdot f.$$

Thus we obtain a map $\mathsf{has\text{-}inverse}(f) \to \mathsf{is\text{-}coh\text{-}invertible}(f)$.

Proof For each $y : B$, we construct the identification $G'(y)$ as the concatenation

$$fg(y) \xrightarrow{G(fg(y))^{-1}} fgfg(y) \xrightarrow{\mathsf{ap}_f(H(g(y)))} fg(y) \xrightarrow{G(y)} y.$$

In order to construct a homotopy $f \cdot H \sim G' \cdot f$, it suffices to show that the square

$$\begin{array}{ccc} fgfgf(x) & \xrightarrow{G(fgf(x))} & fgf(x) \\ {\scriptstyle \mathsf{ap}_f(H(gf(x)))}\downarrow & & \downarrow{\scriptstyle \mathsf{ap}_f(H(x))} \\ fgf(x) & \xrightarrow[G(f(x))]{} & f(x) \end{array}$$

commutes for every $x : A$. Recall from Definition 10.4.4 that we have $H(gf(x)) = \mathsf{ap}_{gf}(H(x))$. Using this identification, we see that it suffices to show that the square

$$\begin{array}{ccc} fgfgf(x) & \xrightarrow{(G\cdot f)(gf(x))} & fgf(x) \\ {\scriptstyle \mathsf{ap}_{fgf}(H(x))}\Big\| & & \Big\|{\scriptstyle \mathsf{ap}_{f}(H(x))} \\ fgf(x) & \xrightarrow[(G\cdot f)(x)]{} & f(x) \end{array}$$

commutes. Now we observe that this is just a naturality square the homotopy $G \cdot f : fgf \sim f$, which commutes by Definition 10.4.3. □

Now we put the pieces together to conclude that any equivalence has contractible fibers.

Theorem 10.4.6 *Any equivalence is a contractible map.*

Proof We have seen in Proposition 10.4.2 that any coherently invertible map is a contractible map. Moreover, any equivalence has the structure of an invertible map by Proposition 9.2.7, and any invertible map is coherently invertible by Lemma 10.4.5. □

The following corollary is very similar to Theorem 10.1.4, which asserts that the type $\sum_{(x:A)} a = x$ is contractible. However, we haven't yet established that the equivalence $(a = x) \simeq (x = a)$ induces an equivalence on total spaces. However, using the fact that equivalences are contractible maps we can give a direct proof.

Corollary 10.4.7 *Let A be a type, and let $a : A$. Then the type*

$$\textstyle\sum_{(x:A)} x = a$$

is contractible.

Proof By Example 9.2.3, the identity function is an equivalence. Therefore, the fibers of the identity function are contractible by Theorem 10.4.6. Note that $\sum_{(x:A)} x = a$ is exactly the fiber of id_A at $a : A$. □

Exercises

10.1 Show that if A is contractible, then for any $x, y : A$ the identity type $x = y$ is also contractible.

10.2 Suppose that A is a retract of B. Show that

$$\mathsf{is\text{-}contr}(B) \to \mathsf{is\text{-}contr}(A).$$

10.3 (a) Show that for any type A, the map $\mathsf{const}_\star : A \to \mathbf{1}$ is an equivalence if and only if A is contractible.

(b) Apply Exercise 9.4 to show that for any map $f : A \to B$, if any two of the three assertions

(i) A is contractible

(ii) B is contractible

(iii) f is an equivalence

hold, then so does the third.

10.4 Show that Fin_k is not contractible for all $k \neq 1$.

10.5 Show that for any two types A and B, the following are equivalent:

(i) Both A and B are contractible.

(ii) The type $A \times B$ is contractible.

10.6 Let A be a contractible type with center of contraction $a : A$. Furthermore, let B be a type family over A. Show that the map

$$y \mapsto (a, y) : B(a) \to \textstyle\sum_{(x:A)} B(x)$$

is an equivalence.

10.7 Let B be a family of types over A, and consider the projection map

$$\mathsf{pr}_1 : \left(\textstyle\sum_{(x:A)} B(x)\right) \to A.$$

(a) Show that for any $a : A$, the map

$$\lambda((x, y), p).\, \mathsf{tr}_B(p, y) : \mathsf{fib}_{\mathsf{pr}_1}(a) \to B(a),$$

is an equivalence.

(b) Show that the following are equivalent:

(i) The projection map pr_1 is an equivalence.

(ii) The type $B(x)$ is contractible for each $x : A$.

(c) Consider a dependent function $b : \prod_{(x:A)} B(x)$. Show that the following are equivalent:

(i) The map

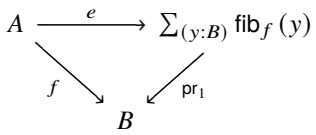
$$\lambda x.\, (x, b(x)) : A \to \textstyle\sum_{(x:A)} B(x)$$

is an equivalence.

(ii) The type $B(x)$ is contractible for each $x : A$.

10.8 Construct for any map $f : A \to B$ an equivalence $e : A \simeq \sum_{(y:B)} \mathsf{fib}_f(y)$

and a homotopy $H : f \sim \mathsf{pr}_1 \circ e$ witnessing that the triangle



commutes. The projection $\mathsf{pr}_1 : (\sum_{(y:B)} \mathsf{fib}_f(y)) \to B$ is sometimes also called the **fibrant replacement** of f, because first projection maps are fibrations in the homotopy interpretation of type theory.

11 The fundamental theorem of identity types

For many types it is useful to have a characterization of their identity types. For example, we have used a characterization of the identity types of the fibers of a map in order to conclude that any equivalence is a contractible map. The fundamental theorem of identity types is our main tool to carry out such characterizations, and with the fundamental theorem it becomes a routine task to characterize an identity type whenever that is of interest. We note that the fundamental theorem also appears as Theorem 5.8.4 in [5].

In our first application of the fundamental theorem of identity types we show that any equivalence is an embedding. Embeddings are maps that induce equivalences on identity types, i.e., they are the homotopical analogue of injective maps. In our second application we characterize the identity types of coproducts.

Throughout this book we will encounter many more occasions to characterize identity types. For example, we will show in Theorem 11.3.1 that the identity type of the natural numbers is equivalent to its observational equality, and we will show in Theorem 21.5.2 that the loop space of the circle is equivalent to $\mathbb{Z}$.

In order to prove the fundamental theorem of identity types, we first prove the basic fact that a family of maps is a family of equivalences if and only if it induces an equivalence on total spaces.

11.1 Families of equivalences

Definition 11.1.1 Consider a family of maps

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

We define the map

$$\mathsf{tot}(f) : \textstyle\sum_{(x:A)} B(x) \to \sum_{(x:A)} C(x)$$

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

Lemma 11.1.2 *For any family of maps $f : \prod_{(x:A)} B(x) \to C(x)$ and any $t : \sum_{(x:A)} C(x)$, there is an equivalence*

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

Proof We first define

$$\varphi : \textstyle\prod_{(t:\sum_{(x:A)} C(x))} \mathsf{fib}_{\mathsf{tot}(f)}(t) \to \mathsf{fib}_{f(\mathsf{pr}_1(t))}(\mathsf{pr}_2(t))$$

by pattern matching, by taking

$$\varphi((x, f(x, y)), ((x, y), \mathsf{refl})) :\equiv (y, \mathsf{refl}).$$

For the proof that $\varphi(t)$ is an equivalence, for each $t : \sum_{(x:A)} C(x)$, we construct a map

$$\psi(t) : \mathsf{fib}_{f(\mathsf{pr}_1(t))}(\mathsf{pr}_2(t)) \to \mathsf{fib}_{\mathsf{tot}(f)}(t)$$

equipped with homotopies $G(t) : \varphi(t) \circ \psi(t) \sim \mathsf{id}$ and $H(t) : \psi(t) \circ \varphi(t) \sim \mathsf{id}$. Each of these definitions is given by pattern matching, as follows:

$$\begin{aligned} \psi((x, f(x, y)), (y, \mathsf{refl})) &:\equiv ((x, y), \mathsf{refl}) \\ G((x, f(x, y)), (y, \mathsf{refl})) &:\equiv \mathsf{refl} \\ H((x, f(x, y)), ((x, y), \mathsf{refl})) &:\equiv \mathsf{refl}. \end{aligned}$$

□

Theorem 11.1.3 *Let $f : \prod_{(x:A)} B(x) \to C(x)$ be a family of maps. The following are equivalent:*

(i) *For each $x : A$, the map $f(x)$ is an equivalence. In this case we say that f is a **family of equivalences**.*
(ii) *The map $\mathsf{tot}(f) : \sum_{(x:A)} B(x) \to \sum_{(x:A)} C(x)$ is an equivalence.*

Proof By Theorems 10.3.5 and 10.4.6 it suffices to show that $f(x)$ is a contractible map for each $x : A$, if and only if $\mathsf{tot}(f)$ is a contractible map. Thus, we will show that $\mathsf{fib}_{f(x)}(c)$ is contractible if and only if $\mathsf{fib}_{\mathsf{tot}(f)}(x, c)$ is contractible, for each $x : A$ and $c : C(x)$. However, by Lemma 11.1.2 these types are equivalent, so the result follows by Exercise 10.3. □

Now consider the situation where we have a map $f : A \to B$, and a family C over B. Then we have the map

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

We claim that this map is an equivalence when f is an equivalence. The technique to prove this claim is the same as the technique we used in Theorem 11.1.3: first we note that the fibers are equivalent to the fibers of f, and then we use the fact that a map is an equivalence if and only if its fibers are contractible to finish the proof.

The converse of the following lemma does not hold. Why not?

Lemma 11.1.4 *Consider a map $f : A \to B$, and let C be a type family over B. If f is an equivalence, then the map*

$$\sigma_f(C) :\equiv \lambda(x,z).\,(f(x),z) : \textstyle\sum_{(x:A)} C(f(x)) \to \sum_{(y:B)} C(y)$$

is an equivalence.

Proof We claim that for each $t : \sum_{(y:B)} C(y)$ there is an equivalence

$$\mathsf{fib}_{\sigma_f(C)}(t) \simeq \mathsf{fib}_f(\mathsf{pr}_1(t)).$$

We obtain such an equivalence by constructing the following functions and homotopies:

$$\begin{aligned}
&\varphi(t) : \mathsf{fib}_{\sigma_f(C)}(t) \to \mathsf{fib}_f(\mathsf{pr}_1(t)) && \varphi((f(x),z),((x,z),\mathsf{refl})) :\equiv (x,\mathsf{refl})\\
&\psi(t) : \mathsf{fib}_f(\mathsf{pr}_1(t)) \to \mathsf{fib}_{\sigma_f(C)}(t) && \psi((f(x),z),(x,\mathsf{refl})) :\equiv ((x,z),\mathsf{refl})\\
&G(t) : \varphi(t) \circ \psi(t) \sim \mathsf{id} && G((f(x),z),(x,\mathsf{refl})) :\equiv \mathsf{refl}\\
&H(t) : \psi(t) \circ \varphi(t) \sim \mathsf{id} && H((f(x),z),((x,z),\mathsf{refl})) :\equiv \mathsf{refl}.
\end{aligned}$$

Now the claim follows, since we see that φ is a contractible map if and only if f is a contractible map. □

Now we use Lemma 11.1.4 to obtain a generalization of Theorem 11.1.3.

Definition 11.1.5 Consider a map $f : A \to B$ and a family of maps

$$g : \textstyle\prod_{(x:A)} C(x) \to D(f(x)),$$

where C is a type family over A, and D is a type family over B. In this situation we also say that g is a **family of maps over** f. Then we define

$$\mathsf{tot}_f(g) : \textstyle\sum_{(x:A)} C(x) \to \sum_{(y:B)} D(y)$$

by $\mathsf{tot}_f(g)(x,z) :\equiv (f(x), g(x,z))$.

Theorem 11.1.6 *Suppose that g is a family of maps over f as in Definition 11.1.5, and suppose that f is an equivalence. Then the following are equivalent:*

(i) *The family of maps g over f is a family of equivalences.*

(ii) *The map* $\mathsf{tot}_f(g)$ *is an equivalence.*

Proof Note that we have a commuting triangle

$$\begin{array}{ccc} \sum_{(x:A)} C(x) & \xrightarrow{\mathsf{tot}_f(g)} & \sum_{(y:B)} D(y) \\ & {\scriptstyle \mathsf{tot}(g)} \searrow \quad \nearrow {\scriptstyle \lambda(x,z).\,(f(x),z)} & \\ & \sum_{(x:A)} D(f(x)) & \end{array}$$

By the assumption that f is an equivalence, it follows that the map

$$\textstyle\sum_{(x:A)} D(f(x)) \to \sum_{(y:B)} D(y)$$

is an equivalence. Therefore it follows that $\mathsf{tot}_f(g)$ is an equivalence if and only if $\mathsf{tot}(g)$ is an equivalence. Now the claim follows, since $\mathsf{tot}(g)$ is an equivalence if and only if g if a family of equivalences. □

11.2 The fundamental theorem

The fundamental theorem of identity types (Theorem 11.2.2) is a general theorem that can be used to characterize the identity type of a given type. It describes a necessary and sufficient condition on a type family B over a type A equipped with a point $a : A$, for there to be a family of equivalences $\prod_{(x:A)}(a = x) \simeq B(x)$.

Before we state the fundamental theorem of identity types we introduce the notion of *identity systems*. Those are families B over a A that satisfy an induction principle that is similar to the path induction principle.

Definition 11.2.1 Let A be a type equipped with a term $a : A$. A **(unary) identity system** on A at a consists of a type family B over A equipped with $b : B(a)$, such that for any family of types $P(x, y)$ indexed by $x : A$ and $y : B(x)$, the function

$$h \mapsto h(a, b) : \left(\textstyle\prod_{(x:A)} \prod_{(y:B(x))} P(x, y)\right) \to P(a, b)$$

has a section.

In other words, if B is an identity system on A at a and P is a family of types indexed by $x : A$ and $y : B(x)$, then there is for each $p : P(a, b)$ a dependent function

$$f : \textstyle\prod_{(x:A)} \prod_{(y:B(x))} P(x, y)$$

such that $f(a, b) = p$. This is of course a variant of path induction, where the computation rule is given by an identification.

The fundamental theorem of identity types asserts that three conditions are

equivalent. The most important implication in the fundamental theorem is that (ii) implies (i). Occasionally we will also use the third equivalent statement.

Theorem 11.2.2 (The fundamental theorem of identity types) *Let A be a type with $a : A$, and let B be be a type family over A. Then the following are logically equivalent for any family of maps*

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

(i) *The family of maps f is a family of equivalences.*
(ii) *The total space*

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

is contractible.
(iii) *The family B is an identity system.*

In particular, we see that for any $b : B(a)$, the canonical family of maps

$$\mathsf{ind\text{-}eq}_a(b) : \prod_{(x:A)} (a = x) \to B(x)$$

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

Proof First we show that (i) and (ii) are equivalent. By Theorem 11.1.3 it follows that the family of maps f is a family of equivalences if and only if it induces an equivalence

$$\left(\sum_{(x:A)} a = x\right) \simeq \left(\sum_{(x:A)} B(x)\right)$$

on total spaces. We have that $\sum_{(x:A)} a = x$ is contractible, so it follows by Exercise 10.3 that $\mathsf{tot}(f)$ is an equivalence if and only if $\sum_{(x:A)} B(x)$ is contractible.

Now we show that (ii) and (iii) are equivalent. Note that we have the following commuting triangle

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

In this diagram the top map has a section. Therefore it follows by Exercise 9.4 that the left map has a section if and only if the right map has a section. Recall from Definition 10.2.1 that the type $\sum_{(x:A)} B(x)$ satisfies singleton induction if and only if the left map in the triangle has a section for each P. Therefore we conclude our proof with Theorem 10.2.3, which shows that the type $\sum_{(x:A)} B(x)$ satisfies singleton induction if and only if it is contractible. □

11.3 Equality on the natural numbers

As a first application of the fundamental theorem of identity types, we characterize the identity type of the natural numbers. We will use the observational equality $\mathsf{Eq}_{\mathbb{N}}$ on $\mathbb{N}$. Recall from Definition 6.3.1 that $\mathsf{Eq}_{\mathbb{N}}$ is defined by

$$\begin{aligned} \mathsf{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) &:\equiv \mathbf{1} & \mathsf{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, n+1) &:\equiv \emptyset \\ \mathsf{Eq}_{\mathbb{N}}(m+1, 0_{\mathbb{N}}) &:\equiv \emptyset & \mathsf{Eq}_{\mathbb{N}}(m+1, n+1) &:\equiv \mathsf{Eq}_{\mathbb{N}}(m, n). \end{aligned}$$

This relation is an equivalence relation. In particular, the reflexivity term $\mathsf{refl\text{-}Eq}_{\mathbb{N}}(m) : \mathsf{Eq}_{\mathbb{N}}(m, m)$ is defined inductively by

$$\begin{aligned} \mathsf{refl\text{-}Eq}_{\mathbb{N}}(0_{\mathbb{N}}) &:\equiv \star \\ \mathsf{refl\text{-}Eq}_{\mathbb{N}}(m+1) &:\equiv \mathsf{refl\text{-}Eq}_{\mathbb{N}}(m). \end{aligned}$$

Using the reflexivity term, we obtain a canonical map

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

for every $m, n : \mathbb{N}$.

Theorem 11.3.1 *For each $m, n : \mathbb{N}$, the canonical map*

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

is an equivalence.

Proof By Theorem 11.2.2 it suffices to show that the type

$$\textstyle\sum_{(n:\mathbb{N})} \mathsf{Eq}_{\mathbb{N}}(m, n)$$

is contractible, for each $m : \mathbb{N}$. The center of contraction is defined to be $(m, \mathsf{refl\text{-}Eq}_{\mathbb{N}}(m))$.

The contraction

$$\gamma(m) : \textstyle\prod_{(n:\mathbb{N})} \prod_{(e:\mathsf{Eq}_{\mathbb{N}}(m,n))} (m, \mathsf{refl\text{-}Eq}_{\mathbb{N}}(m)) = (n, e)$$

is defined for each m by induction on $m, n : \mathbb{N}$. In the base case we define

$$\gamma(0_{\mathbb{N}}, 0_{\mathbb{N}}, \star) :\equiv \mathsf{refl}.$$

If one of m and n is zero and the other is a successor, then the type $\mathsf{Eq}_{\mathbb{N}}(m, n)$ is empty, so the desired path can be obtained via the induction principle of the empty type.

The inductive step remains, in which we have to define the identification

$$\gamma(m+1, n+1, e) : (m+1, \mathsf{refl\text{-}Eq}_{\mathbb{N}}(m+1)) = (n+1, e)$$

for each $m, n : \mathbb{N}$ equipped with $e : \mathsf{Eq}_{\mathbb{N}}(m, n)$. We first observe that there is a map

$$\left(\sum_{(n:\mathbb{N})} \mathsf{Eq}_{\mathbb{N}}(m, n)\right) \xrightarrow{f} \left(\sum_{(n:\mathbb{N})} \mathsf{Eq}_{\mathbb{N}}(m+1, n)\right)$$

given by $(n, e) \mapsto (n+1, e)$. With this definition of f we have

$$f(m, \mathsf{refl\text{-}Eq}_{\mathbb{N}}(m)) \doteq (m+1, \mathsf{refl\text{-}Eq}_{\mathbb{N}}(m+1)).$$

Therefore we can define

$$\gamma(m+1, n+1, e) :\equiv \mathsf{ap}_f(\gamma(m, n, e)). \qquad \square$$

11.4 Embeddings

As an application of the fundamental theorem we show that equivalences are embeddings. The notion of embedding is the homotopical analogue of the set theoretic notion of injective map.

Definition 11.4.1 An **embedding** is a map $f : A \to B$ that satisfies the property that

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

is an equivalence, for every $x, y : A$. We write $\mathsf{is\text{-}emb}(f)$ for the type of witnesses that f is an embedding.

Another way of phrasing the following statement is that equivalent types have equivalent identity types.

Theorem 11.4.2 *Any equivalence is an embedding.*

Proof Let $e : A \simeq B$ be an equivalence, and let $x : A$. Our goal is to show that

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

is an equivalence for every $y : A$. By Theorem 11.2.2 it suffices to show that

$$\textstyle\sum_{(y:A)} e(x) = e(y)$$

is contractible. Now observe that there is an equivalence

$$\begin{aligned}\textstyle\sum_{(y:A)} e(x) = e(y) &\simeq \textstyle\sum_{(y:A)} e(y) = e(x)\\ &\doteq \mathsf{fib}_e(e(x))\end{aligned}$$

by Theorem 11.1.3, since for each $y : A$ the map

$$\mathsf{inv} : (e(x) = e(y)) \to (e(y) = e(x))$$

is an equivalence by Exercise 9.1. The fiber $\mathsf{fib}_e(e(x))$ is contractible by Theorem 10.4.6, so it follows by Exercise 10.3 that the type $\sum_{(y:A)} e(x) = e(y)$ is indeed contractible. □

11.5 Disjointness of coproducts

To give a second application of the fundamental theorem of identity types, we characterize the identity types of coproducts. Our goal in this section is to prove the following theorem.

Theorem 11.5.1 *Let A and B be types. Then there are equivalences*

$$\begin{aligned}(\mathsf{inl}(x) = \mathsf{inl}(x')) &\simeq (x = x')\\ (\mathsf{inl}(x) = \mathsf{inr}(y')) &\simeq \emptyset\\ (\mathsf{inr}(y) = \mathsf{inl}(x')) &\simeq \emptyset\\ (\mathsf{inr}(y) = \mathsf{inr}(y')) &\simeq (y = y')\end{aligned}$$

for any $x, x' : A$ and $y, y' : B$.

In order to prove Theorem 11.5.1, we first define a binary relation $\mathsf{Eq\text{-}copr}_{A,B}$ on the coproduct $A + B$.

Definition 11.5.2 Let A and B be types. We define

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

by double induction on the coproduct, postulating

$$\begin{aligned}\mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inl}(x), \mathsf{inl}(x')) &:\equiv (x = x')\\ \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inl}(x), \mathsf{inr}(y')) &:\equiv \emptyset\\ \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inr}(y), \mathsf{inl}(x')) &:\equiv \emptyset\\ \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inr}(y), \mathsf{inr}(y')) &:\equiv (y = y').\end{aligned}$$

The relation $\mathsf{Eq\text{-}copr}_{A,B}$ is also called the **observational equality of coproducts**.

Lemma 11.5.3 *The observational equality relation $\mathsf{Eq\text{-}copr}_{A,B}$ on $A + B$ is reflexive, and therefore there is a map*

$$\mathsf{Eq\text{-}copr\text{-}eq} : \prod_{(s,t:A+B)} (s = t) \to \mathsf{Eq\text{-}copr}_{A,B}(s, t).$$

Construction The reflexivity term ρ is constructed by induction on $t : A + B$, using

$$\begin{aligned}\rho(\mathsf{inl}(x)) &:\equiv \mathsf{refl}_x : \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inl}(x), \mathsf{inl}(x))\\ \rho(\mathsf{inr}(y)) &:\equiv \mathsf{refl}_y : \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inr}(y), \mathsf{inr}(y)).\end{aligned}$$

□

To show that Eq-copr-eq is a family of equivalences, we will use the fundamental theorem of identity types, Theorem 11.2.2. Therefore, we need to prove the following proposition.

Proposition 11.5.4 *For any $s : A + B$ the total space*

$$\textstyle\sum_{(t:A+B)} \mathsf{Eq\text{-}copr}_{A,B}(s,t)$$

is contractible.

Proof By induction on s, it suffices to show that the total spaces

$$\textstyle\sum_{(t:A+B)} \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inl}(x),t) \qquad \text{and} \qquad \sum_{(t:A+B)} \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inr}(y),t)$$

are contractible. The two proofs are similar, so we only prove that the type on the left is contractible. By the laws of coproducts and Σ-types given in Examples 9.2.9 and 9.2.10, we simply compute

$$\begin{aligned}
&\textstyle\sum_{(t:A+B)} \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inl}(x),t) \\
&\simeq \left(\textstyle\sum_{(x':A)} \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inl}(x),\mathsf{inl}(x'))\right) + \left(\textstyle\sum_{(y':B)} \mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inl}(x),\mathsf{inr}(y'))\right) \\
&\simeq \left(\textstyle\sum_{(x':A)} x = x'\right) + \left(\textstyle\sum_{(y':B)} \emptyset\right) \\
&\simeq \left(\textstyle\sum_{(x':A)} x = x'\right) + \emptyset \\
&\simeq \textstyle\sum_{(x':A)} x = x'.
\end{aligned}$$

The last type in this computation is contractible by Theorem 10.1.4, so we conclude that the total space of $\mathsf{Eq\text{-}copr}_{A,B}(\mathsf{inl}(x))$ is contractible. □

Proof of Theorem 11.5.1 The proof is now concluded with an application of Theorem 11.2.2, using Proposition 11.5.4. □

11.6 The structure identity principle

We often encounter a type consisting of certain objects equipped with further structure. For example, the fiber of a map $f : A \to B$ at $b : B$ is the type of elements $a : A$ equipped with an identification $p : f(a) = b$. Such *structure* types occur all over mathematics, and it is important to have an efficient characterization of their identity types. A general structure type is just a Σ-type, and we're asking for a characterization of its identity type.

Recall from Theorem 9.3.4 that the identity type of the type $\sum_{(x:A)} B(x)$ can be characterized as

$$((x,y) = (x',y')) \simeq \sum_{(p:x=x')} \mathsf{tr}_B(p,y) = y'.$$

However, this characterization of $\sum_{(x:A)} B(x)$ is not as clear and useful as we like it to be, because it uses the transport function, which is completely generic. Moreover, if we already have a characterization of the identity type of A and of the identity type of $B(a)$ at each $a : A$, then we can use those characterizations in the characterization of the identity type of $\sum_{(x:A)} B(x)$.

Definition 11.6.1 Consider a type A equipped with an identity system C based at $a : A$, and let $c : C(a)$. Furthermore, consider a type family B over A. A **dependent identity system** over C at $b : B(a)$ consists of a type family

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

such that $y \mapsto D(a,y,c)$ is an identity system at b.

Theorem 11.6.2 *Consider a type family B over A, and let $a : A$ and $b : B(a)$. If C is an identity system of A at a and D is an identity system over C at $b : B(a)$, then the type family*

$$w \mapsto \sum_{(z:C(\mathsf{pr}_1(w)))} D(\mathsf{pr}_1(w), \mathsf{pr}_2(w), z)$$

indexed by $w : \sum_{(x:A)} B(x)$ is an identity system over $\sum_{(x:A)} B(x)$ at (a,b).

Proof First, we note that there is an equivalence

$$\sum_{(w:\sum_{(x:A)} B(x))} \sum_{(z:C(\mathsf{pr}_1(w)))} D(\mathsf{pr}_1(w), \mathsf{pr}_2(w), z) \simeq \sum_{(w:\sum_{(x:A)} C(x))} \sum_{(y:B(\mathsf{pr}_1(w)))} D(\mathsf{pr}_1(w), y, \mathsf{pr}_2(w)).$$

This equivalence, its inverse, and the homotopies that belong to it are all straightforward to construct using pattern matching.

By Theorem 11.2.2 it therefore suffices to show that the type

$$\sum_{(w:\sum_{(x:A)} C(x))} \sum_{(y:B(\mathsf{pr}_1(w)))} D(\mathsf{pr}_1(w), y, \mathsf{pr}_2(w))$$

is contractible. However, since C is an identity system of A at a, it follows that the type $\sum_{(x:A)} C(x)$ is contractible with center of contraction (a,c). Therefore we obtain by Exercise 10.6 an equivalence

$$\left(\sum_{(y:B(a))} D(a,y,c)\right) \simeq \left(\sum_{(w:\sum_{(x:A)} C(x))} \sum_{(y:B(\mathsf{pr}_1(w)))} D(\mathsf{pr}_1(w), y, \mathsf{pr}_2(w))\right),$$

so it suffices to show that the type in the domain of this equivalence is contractible. This follows from the assumption that D is a dependent identity system over C at $b : B(a)$. □

Example 11.6.3 By the structure identity principle of Theorem 11.6.2 in combination with the fundamental theorem of identity types (Theorem 11.2.2), it becomes completely routine to characterize identity types of structures: We only have to show that the types

$$\textstyle\sum_{(x:A)} C(x) \qquad \text{and} \qquad \sum_{(y:B(a))} D(a,y,c)$$

are contractible. To illustrate this use of the structure identity principle, we give an alternative characterization of the fiber of a map $f : A \to B$ at $b : B$. We claim that

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

To see this, we apply Theorem 11.6.2. Note that $\sum_{(y:A)} x = y$ is contractible by Theorem 10.1.4 with center of contraction $(x, \mathsf{refl}_{f(x)})$. Therefore it suffices to show that the type

$$\textstyle\sum_{(q : f(x) = b)} \mathsf{refl}_{f(x)} = p \cdot q^{-1}$$

is contractible. Of course, this type is equivalent to $\sum_{(q:f(x)=b)} p = q$, which is again contractible by Theorem 10.1.4.

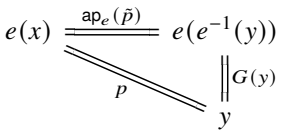
Exercises

11.1 (a) Show that the map $\emptyset \to A$ is an embedding for every type A.
(b) Show that $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$ are embeddings for any two types A and B.
(c) Show that $\mathsf{inl} : A \to A + B$ is an equivalence if and only if B is empty, and that $\mathsf{inr} : B \to A + B$ is an equivalence if and only if A is empty.

11.2 Consider an equivalence $e : A \simeq B$. Construct an equivalence

$$p \mapsto \tilde{p} : (e(x) = y) \simeq (x = e^{-1}(y))$$

for every $x : A$ and $y : B$, such that the triangle



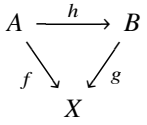
commutes for every $p : e(x) = y$. In this diagram, the homotopy $G : e \circ e^{-1} \sim \mathsf{id}$ is the homotopy witnessing that e^{-1} is a section of e.

11.3 Show that

$$(f \sim g) \to (\mathsf{is\text{-}emb}(f) \leftrightarrow \mathsf{is\text{-}emb}(g))$$

for any $f, g : A \to B$.

11.4 Consider a commuting triangle



with $H : f \sim g \circ h$.

(a) Suppose that g is an embedding. Show that f is an embedding if and only if h is an embedding.

(b) Suppose that h is an equivalence. Show that f is an embedding if and only if g is an embedding.

11.5 Consider two embeddings $f : A \hookrightarrow B$ and $g : B \hookrightarrow C$. Show that the following are equivalent:

(i) The composite $g \circ f$ is an equivalence.

(ii) Both f and g are equivalences.

11.6 Consider two maps $f : A \to C$ and $g : B \to C$. Use Exercise 11.1 (b) to show that the following are equivalent:

(i) The map $[f, g] : A + B \to C$ is an embedding.

(ii) Both f and g are embeddings, and

$$f(a) \neq g(b)$$

for all $a : A$ and $b : B$.

11.7 Consider two maps $f : A \to A'$ and $g : B \to B'$.

(a) Show that if the map

$$f + g : (A + B) \to (A' + B')$$

is an equivalence, then so are both f and g (this is the converse of Exercise 9.6 (e)).

(b) Show that $f + g$ is an embedding if and only if both f and g are embeddings.

11.8 (a) Let $f, g : \prod_{(x:A)} B(x) \to C(x)$ be two families of maps. Show that

$$\left(\prod_{(x:A)} f(x) \sim g(x)\right) \to \left(\mathsf{tot}(f) \sim \mathsf{tot}(g)\right).$$

(b) Let $f : \prod_{(x:A)} B(x) \to C(x)$ and let $g : \prod_{(x:A)} C(x) \to D(x)$. Show that

$$\mathsf{tot}(\lambda x.\, g(x) \circ f(x)) \sim \mathsf{tot}(g) \circ \mathsf{tot}(f).$$

(c) For any family B over A, show that

$$\mathsf{tot}(\lambda x.\, \mathsf{id}_{B(x)}) \sim \mathsf{id}.$$

(d) Let $a : A$, and let B be a type family over A. Use Exercise 10.2 to show that if each $B(x)$ is a retract of $a = x$, then $B(x)$ is equivalent to $a = x$ for every $x : A$.

(e) Conclude that for any family of maps

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

if each $f(x)$ has a section, then f is a family of equivalences.

11.9 Use Exercise 11.8 to show that for any map $f : A \to B$, if

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

has a section for each $x, y : A$, then f is an embedding.

11.10 We say that a map $f : A \to B$ is **path-split** if f has a section, and for each $x, y : A$ the map

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

also has a section. We write $\mathsf{is\text{-}path\text{-}split}(f)$ for the type

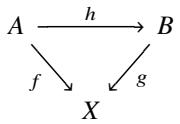
$$\mathsf{sec}(f) \times \prod_{(x,y:A)} \mathsf{sec}(\mathsf{ap}_f(x, y)).$$

Show that for any map $f : A \to B$ the following are equivalent:

(i) The map f is an equivalence.

(ii) The map f is path-split.

11.11 Consider a triangle

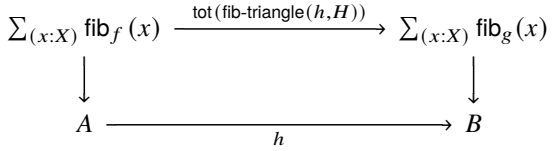


with a homotopy $H : f \sim g \circ h$ witnessing that the triangle commutes.

(a) Construct a family of maps

$$\mathsf{fib\text{-}triangle}(h, H) : \prod_{(x:X)} \mathsf{fib}_f(x) \to \mathsf{fib}_g(x),$$

for which the square



commutes, where the vertical maps are as constructed in Exercise 10.8.

(b) Show that h is an equivalence if and only if $\mathsf{fib\text{-}triangle}(h, H)$ is a family of equivalences.

12 Propositions, sets, and the higher truncation levels

The set theoretic foundations of mathematics arise in two stages. The first stage is to specify the formal system of first order logic; the second stage is to give an axiomatizaion of set theory in this formal system. Unlike set theory, type theory is its own formal system. The logic of dependent types, as given by the inference rules, is all we need.

However, even though type theory is not built upon a separate system of logic such as first order logic, we can find logic in type theory by recognizing certain types as propositions. Note that the propositions of first order logic have a virtue that could be rather useful sometimes: First order logic does not offer any way to distinguish between any two proofs of the same proposition. Therefore we say that propositions in type theory are those types that have at most one element.

This condition can be expressed with the identity type: any two elements must be equal. Examples of such types include the empty type $\emptyset$ and the unit type $\mathbf{1}$. We call such types propositions. Propositions are useful, because if we know that a certain type is a proposition, then we know that any of its inhabitants are equal. Many important conditions, such as the condition that a map is an equivalence, will turn out to be propositions. This fact implies that two equivalences $A \simeq B$ are equal if and only if their underlying maps $A \to B$ are equal. However, the claim that being an equivalence is a proposition requires function extensionality, the topic of the next section.

In this section we use the idea of propositions in a different way. After we establish some basic properties of propositions, we will introduce the *sets* as the types of which the identity types are propositions. This is again reminiscent of the situation in set theory, where equality is a predicate in first order logic. We will see in Example 12.3.2 that the type of natural numbers is a set.

Next, one might ask about the types of which the identity types are *sets*. Such types are called 1-*types*. There is an entire hierarchy of special types that arises this way, where a type is said to be a $(k+1)$-type if its identity types are k-types. Since the identity types of the 1-types are sets, we see that sets are in fact 0-types. Most of mathematics takes place at this level, the level of sets. The types in higher levels, as well as types that do not belong to any finite level in this hierarchy, are studied extensively in synthetic homotopy theory.

However, we can also go a step further down: Since the identity types of sets are propositions, we see that the propositions are (-1)-types. Moreover, the identity types of propositions are contractible. Hence we find at the bottom of this hierarchy the contractible types as the (-2)-types. There is no point in going down further, since we have seen in Exercise 10.1 that the identity types of contractible types are again contractible.

12.1 Propositions

Definition 12.1.1 A type A is said to be a **proposition** if its identity types are contractible, i.e., if it comes equipped with a term of type

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

Given a universe $\mathcal{U}$, we define $\mathsf{Prop}_{\mathcal{U}}$ to be the type of all small propositions, i.e.,

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

Example 12.1.2 Any contractible type is a proposition by Exercise 10.1. In particular, the unit type is a proposition. The empty type is also a proposition, since we have

$$\prod_{(x,y:\emptyset)} \mathsf{is\text{-}contr}(x = y)$$

by the induction principle of the empty type.

There are many conditions on a type A that are equivalent to the condition that A is a proposition. In the following proposition we state four such conditions.

Proposition 12.1.3 *Let A be a type. Then the following are equivalent:*

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

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

(iii) *The type A is contractible as soon as it is inhabited, i.e., there is a function of type*

$$A \to \mathsf{is\text{-}contr}(A).$$

(iv) *The map* $\mathsf{const}_\star : A \to \mathbf{1}$ *is an embedding.*

Proof If A is a proposition, then we can use the center of contraction of the identity types of A to identify any two terms in A. This shows that (i) implies (ii).

To show that (ii) implies (iii), suppose that A comes equipped with $p : \prod_{(x,y:A)} x = y$. Then for any $x : A$ the dependent function $p(x) : \prod_{(y:A)} x = y$ is a contraction of A. Thus we obtain the function

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

To show that (iii) implies (iv), suppose that $A \to \mathsf{is\text{-}contr}(A)$. We first make the simple observation that

$$(X \to \mathsf{is\text{-}emb}(f)) \to \mathsf{is\text{-}emb}(f)$$

for any map $f : X \to Y$, so it suffices to show that $A \to \mathsf{is\text{-}emb}(\mathsf{const}_\star)$. However, assuming we have $x : A$, it follows by assumption that A is contractible. Therefore, it follows by Exercise 10.3 that the map $\mathsf{const}_\star : A \to \mathbf{1}$ is an equivalence, and any equivalence is an embedding by Theorem 11.4.2.

To show that (iv) implies (i), note that if $A \to \mathbf{1}$ is an embedding, then the identity types of A are equivalent to contractible types and therefore they must be contractible. □

One useful feature of propositions, is that in order to construct an equivalence $e : P \simeq Q$ between propositions, it suffices to construct maps back and forth between them.

Proposition 12.1.4 *A map $f : P \to Q$ between two propositions P and Q is an equivalence if and only if there is a map $g : Q \to P$. Consequently, we have for any two propositions P and Q that*

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

Proof Of course, if we have an equivalence $e : P \simeq Q$, then we get maps back and forth between P and Q. Therefore it remains to show that

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

Suppose we have $f : P \to Q$ and $g : Q \to P$. Then we obtain the 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. Therefore f is an equivalence with inverse g. □

12.2 Subtypes

In set theory, a set y is said to be a subset of a set x, if any element of y is an element of x, i.e., if the condition

$$\forall_z\,(z \in y) \to (z \in x)$$

holds. We have already noted that type theory is different from set theory in that terms in type theory come equipped with a *unique* type. Moreover, in set theory the proposition $x \in y$ is well-formed for any two sets x and y, whereas in type theory we can only judge that $a : A$ by applying the rules of inference of type theory in such a manner that we arrive at the conclusion that $a : A$. Because of these differences we must find a different way to talk about subtypes.

Note that in set theory there is a correspondence between the subsets of a set x, and the *predicates* on x. A predicate on x is just a proposition $P(z)$ that varies over the elements $z \in x$. Indeed, if y is a subset of x, then the corresponding predicate is the proposition $z \in y$. Conversely, if P is a predicate on x, then we obtain the subset

$$\{z \in x \mid P(z)\}$$

of x. This observation suggests that in type theory we should define a subtype of a type A to be a family of propositions over A.

Definition 12.2.1 A type family B over A is said to be a **subtype** of A if for each $x : A$ the type $B(x)$ is a proposition. When B is a subtype of A, we also say that $B(x)$ is a **property** of $x : A$.

One reason why subtypes are important and useful, is that for any

$$(x, p), (y, q) : \textstyle\sum_{(x:A)} P(x)$$

in a subtype of A, we have $(x, p) = (y, q)$ if and only if $x = y$. In other words, two terms of a subtype of A are equal if and only if they are equal as terms of A. This fact is properly expressed using embeddings: we claim that the projection map

$$\mathsf{pr}_1 : \left(\textstyle\sum_{(x:A)} P(x)\right) \to A$$

is an embedding, for any subtype P of A. This claim can be strengthened slightly. We will prove the following two closely related facts:

(i) A map $f : A \to B$ is an embedding if and only if its fibers are propositions.
(ii) A family of types B over A is a subtype of A if and only if the projection map

$$\left(\textstyle\sum_{(x:A)} B(x)\right) \to A$$

is an embedding.

The first fact is analogous to the fact that a map is an equivalence if and only if its fibers are contractible, which we saw in Theorems 10.3.5 and 10.4.6. To prove the above claims, we will need that propositions are closed under equivalences.

Lemma 12.2.2 *Let A and B be types, and let $e : A \simeq B$. Then we have*

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

Proof We will show that $\mathsf{is\text{-}prop}(B)$ implies $\mathsf{is\text{-}prop}(A)$. This suffices, because the converse follows from the fact that $e^{-1} : B \to A$ is also an equivalence.

Since e is assumed to be an equivalence, it follows by Theorem 11.4.2 that

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

is an equivalence for any $x, y : A$. If B is a proposition, then in particular the type $e(x) = e(y)$ is contractible for any $x, y : A$, so the claim follows from Theorem 10.4.6. □

Theorem 12.2.3 *Consider a map $f : A \to B$. The following are equivalent:*

(i) *The map f is an embedding.*
(ii) *The fiber $\mathsf{fib}_f(b)$ is a proposition for each $b : B$.*

Proof By the fundamental theorem of identity types, it follows that f is an embedding if and only if

$$\textstyle\sum_{(x:A)} f(x) = f(y)$$

is contractible for each $y : A$. In other words, f is an embedding if and only if $\mathsf{fib}_f(f(y))$ is contractible for each $y : A$. Note that we obtain equivalences

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

for any $b : B$ and $p : f(y) = b$, by transporting along p. Therefore it follows by Lemma 12.2.2 that $\mathsf{fib}_f(f(y))$ is contractible for each $y : A$ if and only if $\mathsf{fib}_f(b)$ is contractible for each $y : A$, and each $b : B$ such that $p : f(y) = b$. The latter condition holds if and only if we have

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

for any $b : B$, which is by Proposition 12.1.3 equivalent to the condition that each $\mathsf{fib}_f(b)$ is a proposition. □

Corollary 12.2.4 *Consider a family B of types over A. The following are equivalent:*

(i) *The map* $\mathsf{pr}_1 : (\sum_{(x:A)} B(x)) \to A$ *is an embedding.*
(ii) *The type* $B(x)$ *is a proposition for each* $x : A$.

Proof This corollary follows at once from Exercise 10.7, where we showed that

$$\mathsf{fib}_{\mathsf{pr}_1}(x) \simeq B(x).$$ □

12.3 Sets

Definition 12.3.1 A type A is said to be a **set** if its identity types are propositions, i.e., if it comes equipped with a term of type

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

Example 12.3.2 The type of natural numbers is a set. To see this, recall from Theorem 11.3.1 that we have an equivalence

$$(m = n) \simeq \mathsf{Eq}_{\mathbb{N}}(m, n)$$

for every $m, n : \mathbb{N}$. Therefore it suffices to show that each $\mathsf{Eq}_{\mathbb{N}}(m, n)$ is a proposition. This follows easily by induction on both m and n.

Proposition 12.3.3 *Consider a type* A. *The following are equivalent:*

(i) *The type* A *is a set.*
(ii) *The type* A *satisfies* ***axiom K****, i.e., if and only if it comes equipped with a term of type*

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

Proof If A is a set, then $x = x$ is a proposition, so any two of its elements are equal. This implies axiom K.

For the converse, if A satisfies axiom K, then for any $p, q : x = y$ we have $p \cdot q^{-1} = \mathsf{refl}_x$, and hence $p = q$. This shows that $x = y$ is a proposition, and hence that A is a set. □

Theorem 12.3.4 *Let* A *be a type, and let* $R : A \to A \to \mathcal{U}$ *be a binary relation on* A *satisfying*

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

$$R(x, y) \to (x = y)$$

for each $x, y : A$.

Then any family of maps

$$\prod_{(x,y:A)}(x = y) \to R(x, y)$$

is a family of equivalences. Consequently, the type A is a set.

Proof Let $f : \prod_{(x,y:A)} R(x, y) \to (x = y)$. Since R is assumed to be reflexive, we also have a family of maps

$$\mathsf{ind\text{-}eq}_x(\rho(x)) : \prod_{(y:A)}(x = y) \to R(x, y).$$

Since each $R(x, y)$ is assumed to be a proposition, it therefore follows that each $R(x, y)$ is a retract of $x = y$. Therefore it follows that $\sum_{(y:A)} R(x, y)$ is a retract of $\sum_{(y:A)} x = y$, which is contractible. We conclude that $\sum_{(y:A)} R(x, y)$ is contractible, and therefore that any family of maps

$$\prod_{(y:A)}(x = y) \to R(x, y)$$

is a family of equivalences.

Now it also follows that A is a set, since its identity types are equivalent to propositions, and therefore they are propositions by Lemma 12.2.2. □

Theorem 12.3.5 (Hedberg) *Any type with decidable equality is a set.*

Proof Let A be a type, and let $d : \prod_{(x,y:A)}(x = y) + (x \neq y)$ be the witness that A has decidable equality. Furthermore, let $\mathcal{U}$ be a universe containing the type A. We will prove that A is a set by applying Theorem 12.3.4.

For every $x, y : A$, we first define a type family $R'(x, y) : ((x{=}y){+}(x \neq y)) \to \mathcal{U}$ by

$$R'(x, y, \mathsf{inl}(p)) :\equiv \mathbf{1}$$
$$R'(x, y, \mathsf{inr}(p)) :\equiv \emptyset.$$

Note that $R'(x, y, q)$ is a proposition for each $x, y : A$ and $q : (x = y) + (x \neq y)$. Now we define $R(x, y) :\equiv R'(x, y, d(x, y))$. Then R is a reflexive binary relation on A, and furthermore each $R(x, y)$ is a proposition. In order to apply Theorem 12.3.4, it therefore it remains to show that R implies identity.

Since R is defined as an instance of R', it suffices to construct a function

$$f(q) : R'(q) \to (x = y).$$

for each $q : (x = y) + (x \neq y)$. Such a function is defined by

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

12.4 General truncation levels

Consider a type A in a universe $\mathcal{U}$. The conditions

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

define the first few layers of the hierarchy of truncation levels. This hierarchy starts at the level of the contractible types, which we call level -2. The next level is the level of propositions, and at level 0 we have the sets.

The indexing type of the truncation levels, which will be equivalent to the type $\mathbb{Z}_{\geq -2}$ of integers greater than -2, is an inductive type $\mathbb{T}$ equipped with the constructors

$$\begin{aligned}-2_{\mathbb{T}} &: \mathbb{T}\\ \mathsf{succ}_{\mathbb{T}} &: \mathbb{T} \to \mathbb{T}.\end{aligned}$$

The natural inclusion $i : \mathbb{N} \to \mathbb{T}$ is defined recursively by

$$\begin{aligned}i(0_{\mathbb{N}}) &:\equiv \mathsf{succ}_{\mathbb{T}}(\mathsf{succ}_{\mathbb{T}}(-2_{\mathbb{T}}))\\ i(\mathsf{succ}_{\mathbb{N}}(n)) &:\equiv \mathsf{succ}_{\mathbb{T}}(i(n)).\end{aligned}$$

Of course, we will simply write -2 for $-2_{\mathbb{T}}$ and $k+1$ for $\mathsf{succ}_{\mathbb{T}}(k)$.

Definition 12.4.1 We define $\mathsf{is\text{-}trunc} : \mathbb{T} \to \mathcal{U} \to \mathcal{U}$ recursively by

$$\begin{aligned}\mathsf{is\text{-}trunc}_{-2}(A) &:\equiv \mathsf{is\text{-}contr}(A)\\ \mathsf{is\text{-}trunc}_{k+1}(A) &:\equiv \textstyle\prod_{(x,y:A)} \mathsf{is\text{-}trunc}_k(x = y).\end{aligned}$$

For any type A, we say that A is **k-truncated**, or a **k-type**, if there is a term of type $\mathsf{is\text{-}trunc}_k(A)$. We also say that a type A is a **proper** $(k+1)$**-type** if A is a $(k+1)$-type and not a k-type.

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

$$\mathcal{U}^{\leq k} :\equiv \textstyle\sum_{(X:\mathcal{U})} \mathsf{is\text{-}trunc}_k(X).$$

Furthermore, we say that a map $f : A \to B$ is k-truncated if its fibers are k-truncated.

Remark 12.4.2 There is a subtlety in the definition of is-trunc regarding universes. Note that the truncation levels are defined with respect to a universe $\mathcal{U}$. To be completely precise, we should therefore write $\mathsf{is\text{-}trunc}_k^{\mathcal{U}}(A)$ for the

type $\mathsf{is\text{-}trunc}_k(A)$ defined with respect to the universe $\mathcal{U}$. If A is also contained in a second universe $\mathcal{V}$, then it is legitimate to ask whether

$$\mathsf{is\text{-}trunc}_k^{\mathcal{U}}(A) \leftrightarrow \mathsf{is\text{-}trunc}_k^{\mathcal{V}}(A).$$

A simple inductive argument shows that this is indeed the case, where the base case follows from the judgmental equalities

$$\mathsf{is\text{-}trunc}_{-2}^{\mathcal{U}}(A) \doteq \textstyle\sum_{(x:A)} \prod_{(y:A)} x = y$$
$$\mathsf{is\text{-}trunc}_{-2}^{\mathcal{V}}(A) \doteq \textstyle\sum_{(x:A)} \prod_{(y:A)} x = y.$$

We may therefore safely omit explicit reference to the universes when considering truncatedness of a type.

We show in the following theorem that the truncation levels are successively contained in one another.

Proposition 12.4.3 *If A is a k-type, then A is also a $(k+1)$-type.*

Proof We have seen in Example 12.1.2 that contractible types are propositions. This proves the base case. For the inductive step, note that if any k-type is also a $(k+1)$-type, then any $(k+1)$-type is a $(k+2)$-type, since its identity types are k-types and therefore $(k+1)$-types. □

It is immediate from the proof of Proposition 12.4.3 that the identity types of k-types are also k-types.

Corollary 12.4.4 *If A is a k-type, then its identity types are also k-types.* □

Proposition 12.4.5 *If $e : A \simeq B$ is an equivalence, and B is a k-type, then so is A.*

Proof We have seen in Exercise 10.3 that if B is contractible and $e : A \simeq B$ is an equivalence, then A is also contractible. This proves the base case.

For the inductive step, assume that the k-types are stable under equivalences, and consider $e : A \simeq B$ where B is a $(k+1)$-type. In Theorem 11.4.2 we have seen that

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

is an equivalence for any x, y. Note that $e(x) = e(y)$ is a k-type, so by the induction hypothesis it follows that $x = y$ is a k-type. This proves that A is a $(k+1)$-type. □

Corollary 12.4.6 *If $f : A \to B$ is an embedding, and B is a $(k+1)$-type, then so is A.*

Proof By the assumption that f is an embedding, the action on paths

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

is an equivalence for every $x, y : A$. Since B is assumed to be a $(k+1)$-type, it follows that $f(x) = f(y)$ is a k-type for every $x, y : A$. Therefore we conclude by Proposition 12.4.5 that $x = y$ is a k-type for every $x, y : A$. In other words, A is a $(k+1)$-type. □

We end this section with a theorem that characterizes $(k+1)$-truncated maps. Note that it generalises Theorem 12.2.3, which asserts that a map is an embedding if and only if its fibers are propositions.

Theorem 12.4.7 *Let $f : A \to B$ be a map. The following are equivalent:*

(i) *The map f is $(k+1)$-truncated.*
(ii) *For each $x, y : A$, the map*

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

is k-truncated.

Proof First we show that for any $s, t : \mathsf{fib}_f(b)$ there is an equivalence

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

We do this by Σ-induction on s and t, and then we calculate

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

By these equivalences, it follows that if ap_f is k-truncated, then for each $s, t : \mathsf{fib}_f(b)$ the identity type $s = t$ is equivalent to a k-truncated type, and therefore we obtain by Proposition 12.4.5 that f is $(k+1)$-truncated.

For the converse, note that we have equivalences

$$\mathsf{fib}_{\mathsf{ap}_f}(p) \simeq ((x,p) = (y, \mathsf{refl}_{f(y)})).$$

It follows that if f is $(k+1)$-truncated, then the identity type $(x,p) = (y, \mathsf{refl}_{f(y)})$ in $\mathsf{fib}_f(f(y))$ is k-truncated for any $p : f(x) = f(y)$. We conclude by Proposition 12.4.5 that the fiber $\mathsf{fib}_{\mathsf{ap}_f}(p)$ is k-truncated. □

Exercises

12.1 Show that bool is a set by applying Theorem 12.3.4 with the observational equality on bool defined in Exercise 6.2.

12.2 Recall that a **partially ordered set (poset)** is defined to be a type A equipped with a relation

$$- \leq - : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$$

that is reflexive, anti-symmetric, and transitive. Show that the underlying type of any poset is a set.

12.3 (a) Show that any injective map $f : A \to B$ into a set B is an embedding, and conclude that A is automatically a set in this case.

(b) Show that $n \mapsto m + n$ is an embedding, for each $m : \mathbb{N}$. Moreover, conclude that there is an equivalence

$$(m \leq n) \simeq \textstyle\sum_{(k:\mathbb{N})} m + k = n.$$

(c) Show that $n \mapsto mn$ is an embedding, for each nonzero number $m : \mathbb{N}$. Conclude that the divisibility relation

$$d \mid n$$

is a proposition for each $d, n : \mathbb{N}$ such that $d > 0$.

12.4 (a) Show that for any two contractible types A and B, the coproduct $A + B$ is not contractible.

(b) Show that for any two propositions P and Q, the coproduct $P + Q$ is a proposition if and only if $P \to \neg Q$.

(c) Show that for any two $(k+2)$-types A and B, the coproduct $A + B$ is again a $(k+2)$-type. Conclude that $\mathbb{Z}$ is a set.

12.5 Let A be a type, and let the **diagonal** of A be the map $\delta_A : A \to A \times A$ given by $\lambda x.\, (x, x)$.

(a) Show that

$$\mathsf{is\text{-}equiv}(\delta_A) \leftrightarrow \mathsf{is\text{-}prop}(A).$$

(b) Construct an equivalence $\mathsf{fib}_{\delta_A}(x, y) \simeq (x = y)$ for any $x, y : A$.

(c) Show that A is $(k+1)$-truncated if and only if $\delta_A : A \to A \times A$ is k-truncated.

12.6 (a) Consider a type family B over a k-truncated type A. Show that the following are equivalent:

(i) The type $B(x)$ is k-truncated for each $x : A$.
(ii) The type $\sum_{(x:A)} B(x)$ is k-truncated.

Hint: for the base case, use Exercises 10.3 and 10.6.

(b) Consider a map $f : A \to B$ into a k-type B. Show that the following are equivalent:

(i) The type A is k-truncated.

(ii) The map f is k-truncated.

12.7 Consider two types A and B. Show that the following are equivalent:

(i) There are functions

$$f : B \to \mathsf{is\text{-}trunc}_{k+1}(A)$$
$$g : A \to \mathsf{is\text{-}trunc}_{k+1}(B).$$

(ii) The type $A \times B$ is $(k+1)$-truncated.

Conclude with Exercise 10.5 that, if both A and B come equipped with an element, then both A and B are k-truncated if and only if the product $A \times B$ is k-truncated.

12.8 (a) Consider a section-retraction pair

$$A \xrightarrow{i} B \xrightarrow{r} A,$$

with $H : r \circ i \sim \mathsf{id}$. Show that $x = y$ is a retract of $i(x) = i(y)$.

(b) Use Exercise 10.2 to show that if A is a retract of a k-type B, then A is also a k-type.

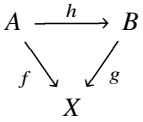
12.9 Consider an arbitrary type A. Recall that concatenation of lists was defined in Exercise 4.4. Show that the map

$$f : \mathsf{list}(A) \times \mathsf{list}(A) \to \mathsf{list}(A).$$

given by $f(x, y) :\equiv \mathsf{concat\text{-}list}(x, y)$ is 0-truncated.

12.10 Show that a type A is a $(k+1)$-type if and only if the map $\mathsf{const}_x : \mathbf{1} \to A$ is k-truncated for every $x : A$.

12.11 Consider a commuting triangle



with $H : f \sim g \circ h$, and suppose that g is k-truncated. Show that f is k-truncated if and only if h is k-truncated.

12.12 Let $f : \prod_{(x:A)} B(x) \to C(x)$ be a family of maps. Show that the following are equivalent:

(i) For each $x : A$ the map $f(x)$ is k-truncated.

(ii) The induced map

$$\mathsf{tot}(f) : \left(\sum_{(x:A)} B(x)\right) \to \left(\sum_{(x:A)} C(x)\right)$$

is k-truncated.

12.13 Consider a type A. Show that the following are equivalent:

(i) The type A is $(k+1)$-truncated.

(ii) For any type family B over A and any $a : A$, the **fiber inclusion**

$$i_a : B(a) \to \sum_{(x:A)} B(x)$$

given by $y \mapsto (a, y)$ is a k-truncated map.

In particular, if A is a set then any fiber inclusion $i_a : B(a) \to \sum_{(x:A)} B(x)$ is an embedding.

12.14 Consider a type A equipped with an element $a : A$. We say that a is an **isolated point** of A if it comes equipped with an element of type

$$\mathsf{is\text{-}isolated}(a) :\equiv \prod_{(x:A)} (a = x) + (a \neq x).$$

(a) Show that a is isolated if and only if the map $\mathsf{const}_a : \mathbf{1} \to A$ has decidable fibers.

(b) Show that if a is isolated, then $a = x$ is a proposition, for every $x : A$. Conclude that if a is isolated, then the map $\mathsf{const}_a : \mathbf{1} \to A$ is an embedding.

13 Function extensionality

The function extensionality axiom asserts that for any two dependent functions $f, g : \prod_{(x:A)} B(x)$, the type of identifications $f = g$ is equivalent to the type of homotopies $f \sim g$ from f to g. In other words, two (dependent) functions can only be distinguished by their values. The function extensionality axiom therefore provides a characterization of the identity type of (dependent) function types. By the fundamental theorem of identity types it follows immediately that the function extensionality axiom has at least three equivalent forms. There is, however, a fourth useful equivalent form of the function extensionality axiom: the *weak* function extensionality axiom. This axiom asserts that any dependent product of contractible types is again contractible. A simple consequence of the weak function extensionality axiom is that any dependent product of a family of k-types is again a k-type.

The function extensionality axiom is used to derive many important properties

in type theory. One class of such properties are (dependent) universal properties. Universal properties give a characterization of the type of functions into, or out of a type. For example, the universal property of the coproduct $A + B$ characterizes the type of maps $(A + B) \to X$ as the type of pairs of maps (f, g) consisting of $f : A \to X$ and $g : B \to X$, i.e., the universal property of the coproduct $A + B$ is an equivalence

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

Note that there are function types on both sides of this equivalence. Therefore we will need function extensionality in order to construct the homotopies witnessing that the inverse map is both a left and a right inverse. In fact, we leave this particular universal property as Exercise 13.9. The universal properties that we do show in the main text, are the universal properties of Σ-types and of the identity type.

We end this section with two further applications of the function extensionality axiom. In the first, Theorem 13.4.1, we show that precomposition by an equivalence is again an equivalence. More precisely we show that $f : A \to B$ is an equivalence if and only if for every type family P over B, the precomposition map

$$- \circ f : \left(\prod_{(y:B)} P(y)\right) \to \left(\prod_{(x:A)} P(f(x))\right)$$

is an equivalence. To prove this fact we will make use of coherently invertible maps, which were introduced in Section 10.4. In the second application, Theorem 13.5.1, we prove the strong induction principle of the natural numbers. Function extensionality is needed in order to derive the computation rule for the strong induction principle.

Many important consequences of the function extensionality axiom are left as exercises. For example, in Exercise 13.3 you are asked to show that both $\mathsf{is\text{-}contr}(A)$ and $\mathsf{is\text{-}trunc}_k(A)$ are propositions, and in Exercise 13.4 you are asked to show that $\mathsf{is\text{-}equiv}(f)$ is a proposition. The universal properties of $\emptyset$, $\mathbf{1}$, and $A + B$ are left as Exercises 13.7 to 13.9. A few more advanced properties, such as the fact that post-composition

$$g \circ - : (A \to X) \to (A \to Y)$$

by a k-truncated map $g : X \to Y$ is itself a k-truncated map, appear in the later exercises. We encourage you to read through all of them, and get at least a basic idea of why they are true.

13.1 Equivalent forms of function extensionality

The function extensionality principle characterizes the identity type of an arbitrary dependent function type. It asserts that the type $f = g$ of identifications between two dependent functions is equivalent to the type of homotopies $f \sim g$. By Theorem 11.2.2 there are three equivalent ways of doing this.

Proposition 13.1.1 *Consider a dependent function $f : \prod_{(x:A)} B(x)$. The following are equivalent:*

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

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

defined by $\mathsf{htpy\text{-}eq}(\mathsf{refl}_f) := \mathsf{refl\text{-}htpy}_f$ is a family of equivalences.

(ii) *The total space*

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

is contractible.

(iii) *The principle of **homotopy induction**: for any family of types $P(g, H)$ indexed by $g : \prod_{(x:A)} B(x)$ and $H : f \sim g$, the evaluation function*

$$\left(\textstyle\prod_{(g:\prod_{(x:A)} B(x))} \prod_{(H:f\sim g)} P(g, H)\right) \to P(f, \mathsf{refl\text{-}htpy}_f),$$

given by $s \mapsto s(f, \mathsf{refl\text{-}htpy}_f)$, has a section.

Proof This theorem follows directly from Theorem 11.2.2. □

There is, however, yet a fourth condition equivalent to the function extensionality principle: the *weak* function extensionality principle. The weak function extensionality principle asserts that any dependent product of contractible types is again contractible.

The following theorem is stated with respect to an arbitrary universe $\mathcal{U}$, because we will use it in Theorem 17.2.2 to show that the univalence axiom implies function extensionality.

Theorem 13.1.2 *Consider a universe $\mathcal{U}$. The following are equivalent:*

(i) *The function extensionality principle holds in $\mathcal{U}$: For every type family B over A in $\mathcal{U}$ and any $f, g : \prod_{(x:A)} B(x)$, the map*

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

is an equivalence.

(ii) *The **weak function extensionality principle** holds in $\mathcal{U}$: For every type family B over A in $\mathcal{U}$ one has*

$$\left(\prod_{(x:A)} \mathsf{is\text{-}contr}(B(x))\right) \to \mathsf{is\text{-}contr}\left(\prod_{(x:A)} B(x)\right).$$

Proof First, we show that function extensionality implies weak function extensionality, suppose that each $B(a)$ is contractible with center of contraction $c(a)$ and contraction $C_a : \prod_{(y:B(a))} c(a) = y$. Then we take $c :\equiv \lambda a.\, c(a)$ to be the center of contraction of $\prod_{(x:A)} B(x)$. To construct the contraction we have to define a term of type

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

Let $f : \prod_{(x:A)} B(x)$. By function extensionality we have a map $(c \sim f) \to (c = f)$, so it suffices to construct a term of type $c \sim f$. Here we take $\lambda a.\, C_a(f(a))$. This completes the proof that function extensionality implies weak function extensionality.

It remains to show that weak function extensionality implies function extensionality. By Proposition 13.1.1 it suffices to show that the type

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

is contractible for any $f : \prod_{(x:A)} B(x)$. In order to do this, we first note that we have a section-retraction pair

$$\begin{aligned}\left(\sum_{(g:\prod_{(x:A)} B(x))} f \sim g\right) &\xrightarrow{i} \left(\prod_{(x:A)} \sum_{(b:B(x))} f(x) = b\right) \\ &\xrightarrow{r} \left(\sum_{(g:\prod_{(x:A)} B(x))} f \sim g\right)\end{aligned}$$

Here we have the functions

$$\begin{aligned} i &:\equiv \lambda(g, H).\, \lambda x.\, (g(x), H(x)) \\ r &:\equiv \lambda p.\, (\lambda x.\, \mathsf{pr}_1(p(x)), \lambda x.\, \mathsf{pr}_2(p(x))). \end{aligned}$$

Their composite is homotopic to the identity function by the computation rule for Σ-types and the η-rule for Π-types:

$$\begin{aligned} r(i(g, H)) &\doteq r(\lambda x.\, (g(x), H(x))) \\ &\doteq (\lambda x.\, g(x), \lambda x.\, H(x)) \\ &\doteq (g, H). \end{aligned}$$

Now we observe that the type $\prod_{(x:A)} \sum_{(b:B(x))} f(x) = b$ is a product of contractible types, so it is contractible by our assumption of the weak function extensionality principle. The claim now follows, because retracts of contractible types are contractible by Exercise 10.2. □

We will henceforth assume the function extensionality principle as an axiom.

Axiom 13.1.3 (Function Extensionality) For any type family B over A, and any two dependent functions $f, g : \prod_{(x:A)} B(x)$, the map

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

is an equivalence. We will write $\mathsf{eq\text{-}htpy}$ for its inverse.

Remark 13.1.4 The function extensionality axiom is added to type theory by adding the rule

$$\frac{\Gamma, x : A \vdash B(x)\ \mathsf{type} \qquad \Gamma \vdash f : \prod_{(x:A)} B(x) \qquad \Gamma \vdash g : \prod_{(x:A)} B(x)}{\Gamma \vdash \mathsf{funext} : \mathsf{is\text{-}equiv}(\mathsf{htpy\text{-}eq}_{f,g})}$$

In the following theorem we extend the weak function extensionality principle to general truncation levels.

Theorem 13.1.5 *For any type family B over A one has*

$$\left(\prod_{(x:A)} \mathsf{is\text{-}trunc}_k(B(x))\right) \to \mathsf{is\text{-}trunc}_k\left(\prod_{(x:A)} B(x)\right).$$

Proof The theorem is proven by induction on $k \geq -2$. The base case is just the weak function extensionality principle, which was shown to follow from function extensionality in Theorem 13.1.2.

For the inductive step, assume that the k-truncated types are closed under Π-types, and consider a family B of $(k+1)$-truncated types. To show that the type $\prod_{(x:A)} B(x)$ is $(k+1)$-truncated, we have to show that the type $f = g$ is k-truncated for every $f, g : \prod_{(x:A)}$. By function extensionality, the type $f = g$ is equivalent to $f \sim g$ for any two dependent functions $f, g : \prod_{(x:A)} B(x)$. Now observe that $f \sim g$ is a dependent product of k-truncated types, and therefore it is k-truncated by the inductive hypothesis. Since the k-truncated types are closed under equivalences by Proposition 12.4.5, it follows that the type $f = g$ is k-truncated. □

Corollary 13.1.6 *Suppose B is a k-type. Then $A \to B$ is also a k-type, for any type A.*

Remark 13.1.7 It follows that $\neg A$ is a proposition for each type A. Note that it requires function extensionality even just to prove that $\neg P$ is a proposition for any proposition P.

13.2 Identity systems on Π-types

Recall from Section 11.6 that the *structure identity principle* is a way to obtain an identity system on a Σ-type. Identity systems were defined in Definition 11.2.1. In this section we will describe how to obtain identity systems on a Π-type. We will first show that Π-types distribute over Σ-types. This theorem is sometimes called the *type theoretic principle of choice* because it can be seen as the Curry-Howard interpretation of the axiom of choice.

Theorem 13.2.1 *Consider a family of types $C(x, y)$ indexed by $x : A$ and $y : B(x)$. Then the map*

$$\mathsf{choice} : \left(\prod_{(x:A)} \sum_{(y:B(x))} C(x, y)\right) \to \left(\sum_{(f:\prod_{(x:A)} B(x))} \prod_{(x:A)} C(x, f(x))\right)$$

given by

$$\mathsf{choice}(h) :\equiv (\lambda x.\, \mathsf{pr}_1(h(x)), \lambda x.\, \mathsf{pr}_2(h(x))).$$

is an equivalence.

Proof We define the map

$$\mathsf{choice}^{-1} : \left(\sum_{(f:\prod_{(x:A)} B(x))} \prod_{(x:A)} C(x, f(x))\right) \to \prod_{(x:A)} \sum_{(y:B(x))} C(x, y)$$

by $\mathsf{choice}^{-1}(f, g) :\equiv \lambda x.\, (f(x), g(x))$. Then we have to construct homotopies

$$\mathsf{choice} \circ \mathsf{choice}^{-1} \sim \mathsf{id}, \qquad \text{and} \qquad \mathsf{choice}^{-1} \circ \mathsf{choice} \sim \mathsf{id}.$$

For the first homotopy it suffices to construct an identification

$$\mathsf{choice}(\mathsf{choice}^{-1}(f, g)) = (f, g)$$

for any $f : \prod_{(x:A)} B(x)$ and any $g : \prod_{(x:A)} C(x, f(x))$. We compute the left-hand side as follows:

$$\begin{aligned} \mathsf{choice}(\mathsf{choice}^{-1}(f, g)) &\doteq \mathsf{choice}(\lambda x.\, (f(x), g(x))) \\ &\doteq (\lambda x.\, f(x), \lambda x.\, g(x)). \end{aligned}$$

By the η-rule for Π-types we have the judgmental equalities $f \doteq \lambda x.\, f(x)$ and $g \doteq \lambda x.\, g(x)$. Therefore we have the identification

$$\mathsf{refl}_{(f,g)} : \mathsf{choice}(\mathsf{choice}^{-1}(f, g)) = (f, g).$$

This completes the construction of the first homotopy.

For the second homotopy we have to construct an identification

$$\mathsf{choice}^{-1}(\mathsf{choice}(h)) = h$$

for any $h : \prod_{(x:A)} \sum_{(y:B(x))} C(x,y)$. We compute the left-hand side as follows:

$$\begin{aligned}\mathsf{choice}^{-1}(\mathsf{choice}(h)) &\doteq \mathsf{choice}^{-1}(\lambda x.\, \mathsf{pr}_1(h(x)), (\lambda x.\, \mathsf{pr}_2(h(x))))\\ &\doteq \lambda x.\, (\mathsf{pr}_1(h(x)), \mathsf{pr}_2(h(x)))\end{aligned}$$

However, it is *not* the case that $(\mathsf{pr}_1(h(x)), \mathsf{pr}_2(h(x))) \doteq h(x)$ for any $h : \prod_{(x:A)} \sum_{(y:B(x))} C(x,y)$. Nevertheless, we have the identification

$$\mathsf{eq\text{-}pair}(\mathsf{refl}, \mathsf{refl}) : (\mathsf{pr}_1(h(x)), \mathsf{pr}_2(h(x))) = h(x).$$

Therefore we obtain the required homotopy by function extensionality:

$$\lambda h.\, \mathsf{eq\text{-}htpy}(\lambda x.\, \mathsf{eq\text{-}pair}(\mathsf{refl}_{\mathsf{pr}_1(h(x))}, \mathsf{refl}_{\mathsf{pr}_2(h(x))})) : \mathsf{choice}^{-1} \circ \mathsf{choice} \sim \mathsf{id}. \quad \square$$

The fact that Π-types distribute over Σ-types has many useful consequences. The most straightforward consequence is the following.

Corollary 13.2.2 *For any two types A and B, and any type family C over B, we have an equivalence*

$$\left(A \to \sum_{(y:B)} C(y)\right) \simeq \left(\sum_{(f:A\to B)} \prod_{(x:A)} C(f(x))\right).$$

Another direct consequence of the distributivity of Π-types over Σ-types is the fact that

$$\prod_{(b:B)} \mathsf{fib}_f(b) \simeq \sum_{(g:B\to A)} f \circ g \sim \mathsf{id}.$$

In the following corollary we use the distributivity of Π-types over Σ-types to show that dependent functions are sections of projection maps.

Corollary 13.2.3 *Consider a type family B over A, and consider the projection map*

$$\mathsf{pr}_1 : \left(\sum_{(x:A)} B(x)\right) \to A.$$

Then we have an equivalence

$$\mathsf{sec}(\mathsf{pr}_1) \simeq \prod_{(x:A)} B(x).$$

Proof Theorem 13.2.1 gives the first equivalence in the following calculation:

$$\begin{aligned}\sum_{(h:A\to\sum_{(x:A)} B(x))} \mathsf{pr}_1 \circ h \sim \mathsf{id} &\simeq \sum_{((f,g):\sum_{(f:A\to A)} \prod_{(x:A)} B(f(x)))} f \sim \mathsf{id}\\ &\simeq \sum_{((f,H):\sum_{(f:A\to A)} f\sim\mathsf{id})} \prod_{(x:A)} B(f(x))\\ &\simeq \prod_{(x:A)} B(x)\end{aligned}$$

In the second equivalence we used Exercise 9.5 to swap the family $f \mapsto$

$\prod_{(x:A)} B(f(x))$ with the family $f \mapsto f \sim \mathsf{id}$, and in the third equivalence we used the fact that

$$\sum_{(f:A\to A)} f \sim \mathsf{id}$$

is contractible, with center of contraction $(\mathsf{id}, \mathsf{refl\text{-}htpy})$. One way to see that it is contractible is by Exercise 13.1. A direct way to see this, is by another application of Theorem 13.2.1. This gives an equivalence

$$\left(\sum_{(f:A\to A)} f \sim \mathsf{id}\right) \simeq \left(\prod_{(x:A)} \sum_{(y:A)} y = x\right),$$

and the right-hand side is a product of contractible types. □

In the final application of distributivity of Π-types over Σ-types we obtain a general way of constructing identity systems of Π-types.

Theorem 13.2.4 *Consider a family B of types over A, and for each $b : B(a)$ consider an identity system $E(b)$ at b. Furthermore, consider a dependent function $f : \prod_{(x:A)} B(x)$. Then the family of types*

$$\prod_{(x:A)} E(f(x), g(x))$$

indexed by $g : \prod_{(x:A)} B(x)$ is an identity system at f.

Proof By Theorem 11.2.2 it suffices to show that the type

$$\sum_{(g:\prod_{(x:A)} B(x))} \prod_{(x:A)} E(f(x), g(x))$$

is contractible. By Theorem 13.2.1 it follows that this type is equivalent to the type

$$\prod_{(x:A)} \sum_{(y:B(x))} E(f(x), y).$$

This is a product of contractible types because each $E(f(x))$ is an identity system at $f(x) : B(x)$. This product is therefore contractible by the weak function extensionality principle. □

13.3 Universal properties

The function extensionality principle allows us to prove *universal properties*. Universal properties are characterizations of all maps out of or into a given type, so they are very important. Among other applications, universal properties characterize a type up to equivalence. We prove here the universal properties of dependent pair types and of identity types. In the exercises, you are asked to prove the universal properties of $\mathbf{1}$, $\emptyset$, and coproducts.

The universal property of Σ-types

The **universal property of Σ-types** characterizes maps *out of* a dependent pair type $\sum_{(x:A)} B(x)$. It asserts that the map

$$\mathsf{ev\text{-}pair} : \left(\left(\sum_{(x:A)} B(x)\right) \to X\right) \to \left(\prod_{(x:A)} (B(x) \to X)\right),$$

given by $f \mapsto \lambda x.\, \lambda y.\, f(x, y)$, is an equivalence for any type X. In fact, we will prove a slight generalization of this universal property. We will prove the **dependent universal property** of Σ-types, which characterizes *dependent* functions out of $\sum_{(x:A)} B(x)$.

Theorem 13.3.1 *Let B be a type family over A, and let C be a type family over $\sum_{(x:A)} B(x)$. Then the map*

$$\mathsf{ev\text{-}pair} : \left(\prod_{(z:\sum_{(x:A)} B(x))} C(z)\right) \to \left(\prod_{(x:A)} \prod_{(y:B(x))} C(x, y)\right),$$

given by $f \mapsto \lambda x.\, \lambda y.\, f(x, y)$, is an equivalence.

Proof The map in the converse direction is obtained by the induction principle of Σ-types. It is simply the map

$$\mathsf{ind}_\Sigma : \left(\prod_{(x:A)} \prod_{(y:B(x))} C(x, y)\right) \to \left(\prod_{(z:\sum_{(x:A)} B(x))} C(z)\right).$$

By the computation rule for Σ-types we have the homotopy

$$\mathsf{refl\text{-}htpy} : \mathsf{ev\text{-}pair} \circ \mathsf{ind}_\Sigma \sim \mathsf{id}.$$

This shows that ind_Σ is a section of $\mathsf{ev\text{-}pair}$.

To show that $\mathsf{ind}_\Sigma \circ \mathsf{ev\text{-}pair} \sim \mathsf{id}$ we will apply the function extensionality principle. Therefore it suffices to show that $\mathsf{ind}_\Sigma(\lambda x.\, \lambda y.\, f(x, y)) = f$. We apply function extensionality again, so it suffices to show that

$$\prod_{(t:\sum_{(x:A)} B(x))} \mathsf{ind}_\Sigma\big(\lambda x.\, \lambda y.\, f(x, y)\big)(t) = f(t).$$

We obtain this homotopy by another application of Σ-induction. □

Corollary 13.3.2 *Let A, B, and X be types. Then the map*

$$\mathsf{ev\text{-}pair} : (A \times B \to X) \to (A \to (B \to X))$$

given by $f \mapsto \lambda a.\, \lambda b.\, f(a, b)$ is an equivalence.

The universal property of identity types

The universal property of identity types is the fact that families of maps out of the identity type are uniquely determined by their action on the reflexivity identification. More precisely, the map

$$\mathsf{ev\text{-}refl} : \left(\prod_{(x:A)} (a = x) \to B(x)\right) \to B(a)$$

given by $\lambda f.\, f(a, \mathsf{refl}_a)$ is an equivalence, for every type family B over A. Since this result is similar to the Yoneda lemma of category theory, the universal property of identity types is sometimes referred to as the *type theoretic Yoneda lemma*. We will prove the *dependent* universal property of identity types, a slight generalization of the universal property.

Theorem 13.3.3 *Consider a type A equipped with $a : A$, and consider a family of types $B(x, p)$ indexed by $x : A$ and $p : a = x$. Then the map*

$$\mathsf{ev\text{-}refl} : \left(\prod_{(x:A)} \prod_{(p:a=x)} B(x, p)\right) \to B(a, \mathsf{refl}_a),$$

given by $\lambda f.\, f(a, \mathsf{refl}_a)$, is an equivalence.

Proof The inverse is the function

$$\mathsf{ind\text{-}eq}_a : B(a, \mathsf{refl}_a) \to \prod_{(x:A)} \prod_{(p:a=x)} B(x, p).$$

It is immediate from the computation rule of the path induction principle that $\mathsf{ev\text{-}refl} \circ \mathsf{ind\text{-}eq}_a \sim \mathsf{id}$.

To see that $\mathsf{ind\text{-}eq}_a \circ \mathsf{ev\text{-}refl} \sim \mathsf{id}$, let $f : \prod_{(x:A)} (a = x) \to B(x, p)$. To show that $\mathsf{ind\text{-}eq}_a(f(a, \mathsf{refl}_a)) = f$ we apply function extensionality twice. Therefore it suffices to show that

$$\prod_{(x:A)} \prod_{(p:a=x)} \mathsf{ind\text{-}eq}_a(f(a, \mathsf{refl}_a), x, p) = f(x, p).$$

This follows by path induction on p, since $\mathsf{ind\text{-}eq}_a(f(a, \mathsf{refl}_a), a, \mathsf{refl}_a) \doteq f(a, \mathsf{refl}_a)$ by the computation rule of path induction. □

13.4 Composing with equivalences

We show in the following theorem that a map $f : A \to B$ is an equivalence if and only if precomposing by f is an equivalence.

Theorem 13.4.1 *For any map $f : A \to B$, the following are equivalent:*

(i) *f is an equivalence.*

(ii) *For any type family P over B the map*

$$\left(\prod_{(y:B)} P(y)\right) \to \left(\prod_{(x:A)} P(f(x))\right)$$

given by $h \mapsto h \circ f$ is an equivalence.

(iii) *For any type X the map*

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

given by $g \mapsto g \circ f$ is an equivalence.

Proof To show that (i) implies (ii), we first recall from Lemma 10.4.5 that any equivalence is also coherently invertible. Therefore f comes equipped with

$$\begin{aligned} g &: B \to A \\ G &: f \circ g \sim \mathsf{id}_B \\ H &: g \circ f \sim \mathsf{id}_A \\ K &: G \cdot f \sim f \cdot H. \end{aligned}$$

Then we define the inverse of $- \circ f$ to be the map

$$\varphi : \left(\prod_{(x:A)} P(f(x))\right) \to \left(\prod_{(y:B)} P(y)\right)$$

given by $h \mapsto \lambda y.\, \mathsf{tr}_P(G(y), h(g(y)))$.

To see that φ is a section of $- \circ f$, let $h : \prod_{(x:A)} P(f(x))$. By function extensionality it suffices to construct a homotopy $\varphi(h) \circ f \sim h$. In other words, we have to show that

$$\mathsf{tr}_P(G(f(x)), h(g(f(x))) = h(x)$$

for any $x : A$. Now we use the additional homotopy K from our assumption that f is coherently invertible. Since we have $K(x) : G(f(x)) = \mathsf{ap}_f(H(x))$ it suffices to show that

$$\mathsf{tr}_P(\mathsf{ap}_f(H(x)), h(g(f(x)))) = h(x).$$

A simple path-induction argument yields that

$$\mathsf{tr}_P(\mathsf{ap}_f(p)) \sim \mathsf{tr}_{P \circ f}(p)$$

for any path $p : x = y$ in A, so it suffices to construct an identification

$$\mathsf{tr}_{P \circ f}(H(x), h(g(f(x)))) = h(x).$$

We have such an identification by $\mathsf{apd}_h(H(x))$.

To see that φ is a retraction of $- \circ f$, let $h : \prod_{(y:B)} P(y)$. By function

extensionality it suffices to construct a homotopy $\varphi(h \circ f) \sim h$. In other words, we have to show that

$$\mathsf{tr}_P(G(y), h(f(g(y)))) = h(y)$$

for any $y : B$. We have such an identification by $\mathsf{apd}_h(G(y))$. This completes the proof that (i) implies (ii).

Note that (iii) is an immediate consequence of (ii), since we can just choose P to be the constant family X.

It remains to show that (iii) implies (i). Suppose that

$$- \circ f : (B \to X) \to (A \to X)$$

is an equivalence for every type X. Then its fibers are contractible by Theorem 10.4.6. In particular, choosing $X \doteq A$ we see that the fiber

$$\mathsf{fib}_{-\circ f}(\mathsf{id}_A) \doteq \textstyle\sum_{(h:B\to A)} h \circ f = \mathsf{id}_A$$

is contractible. Thus we obtain a function $h : B \to A$ and a homotopy $H : h \circ f \sim \mathsf{id}_A$ showing that h is a retraction of f. We will show that h is also a section of f. To see this, we use that the fiber

$$\mathsf{fib}_{-\circ f}(f) \doteq \textstyle\sum_{(i:B\to B)} i \circ f = f$$

is contractible (choosing $X :\equiv B$). Of course we have $(\mathsf{id}_B, \mathsf{refl}_f)$ in this fiber. However we claim that there also is an identification $p : (f \circ h) \circ f = f$, showing that $(f \circ h, p)$ is in this fiber, because

$$\begin{aligned}(f \circ h) \circ f &\doteq f \circ (h \circ f)\\ &= f \circ \mathsf{id}_A\\ &\doteq f\end{aligned}$$

From the contractibility of the fiber we obtain an identification $(\mathsf{id}_B, \mathsf{refl}_f) = (f \circ h, p)$. In particular we obtain that $\mathsf{id}_B = f \circ h$, showing that h is a section of f. □

13.5 The strong induction principle of $\mathbb{N}$

In the final application of the function extensionality principle we prove the strong induction principle for the type of natural numbers. Function extensionality is used to derive the computation rules of the strong induction principle.

Theorem 13.5.1 *Consider a type family P over $\mathbb{N}$ equipped with*

$$p_0 : P(0)$$

$$p_S : \prod_{(n:\mathbb{N})} \left(\prod_{(m:\mathbb{N})} (m \leq n) \to P(m) \right) \to P(n+1).$$

Then there is a dependent function

$$\mathsf{strong\text{-}ind}_{\mathbb{N}}(p_0, p_S) : \prod_{(n:\mathbb{N})} P(n)$$

that satisfies the following computation rules

$$\mathsf{strong\text{-}ind}_{\mathbb{N}}(p_0, p_S, 0) = p_0$$
$$\mathsf{strong\text{-}ind}_{\mathbb{N}}(p_0, p_S, n+1) = p_S(n, (\lambda m.\, \lambda p.\, \mathsf{strong\text{-}ind}_{\mathbb{N}}(p_0, p_S, m))).$$

In order to construct $\mathsf{strong\text{-}ind}_{\mathbb{N}}(p_0, p_S)$, we first define the type family $\tilde{P}$ over $\mathbb{N}$ by

$$\tilde{P}(n) :\equiv \prod_{(m:\mathbb{N})} (m \leq n) \to P(m).$$

The idea is then to first use p_0 and p_S to construct

$$\tilde{p}_0 : \tilde{P}(0)$$
$$\tilde{p}_S : \prod_{(n:\mathbb{N})} \tilde{P}(n) \to \tilde{P}(n+1).$$

The ordinary induction principle of $\mathbb{N}$ then gives a function

$$\mathsf{ind}_{\mathbb{N}}(\tilde{p}_0, \tilde{p}_S) : \prod_{(n:\mathbb{N})} \tilde{P}(n),$$

which can be used to define a function $\prod_{(n:\mathbb{N})} P(n)$.

Before we start by the proof of Theorem 13.5.1 we state two lemmas in which we construct $\tilde{p}_0$ and $\tilde{p}_S$ with computation rules of their own. We will assume a type family P over $\mathbb{N}$ equipped with

$$p_0 : P(0)$$
$$p_S : \prod_{(n:\mathbb{N})} \tilde{P}(n) \to P(n+1),$$

as in the hypotheses of Theorem 13.5.1.

Lemma 13.5.2 *There is an element $\tilde{p}_0 : \tilde{P}(0)$ that satisfies the judgmental equality*

$$\tilde{p}_0(0, p) \doteq p_0$$

for any $p : 0 \leq 0$.

Proof The fact that we have such a dependent function $\tilde{p}_0$ follows immediately by induction on m and $p : m \leq 0$. □

Lemma 13.5.3 *There is a function*

$$\tilde{p}_S : \prod_{(n:\mathbb{N})} \tilde{P}(n) \to \tilde{P}(n+1)$$

equipped with

(i) *an identification*

$$\tilde{p}_S(n, H, m, p) = H(m, q)$$

for every $H : \tilde{P}(n)$ *and every* $p : m \leq n+1$ *and* $q : m \leq n$, *and*

(ii) *an identification*

$$\tilde{p}_S(n, H, n+1, p) = p_S(n, H)$$

for every $p : n+1 \leq n+1$.

Proof To define the function $\tilde{p}_S(n, H)$, note that there is a function

$$f : (m \leq n+1) \to (m \leq n) + (m = n+1) \tag{$*$}$$

which can be defined by induction on n and m. Using the fact that the domain and codomain of this map are both propositions, this function is easily seen to be an equivalence. Therefore we define first a function

$$h(n, H) : \textstyle\prod_{(m:\mathbb{N})}((m \leq n) + (m = n+1)) \to P(m)$$

by case analysis on $x : (m \leq n) + (m = n+1)$. There are two cases to consider: one where we have $q : m \leq n$, and one where we have $q : m = n+1$. Note that in the second case it suffices to make a definition for $q \doteq \mathsf{refl}$. Therefore we define

$$h(n, H, m, x) = \begin{cases} H(m, q) & \text{if } x \doteq \mathsf{inl}(q) \\ p_S(n, H) & \text{if } x \doteq \mathsf{inr}(\mathsf{refl}). \end{cases}$$

Now we define $\tilde{p}_S$ by

$$\tilde{p}_S(n, H, m, p) :\equiv h(n, H, m, f(p)),$$

where $f : (m \leq n+1) \to (m \leq n) + (m = n+1)$ is the map we mentioned in $(*)$.

To construct identificiations claimed in (i) and (ii), note that there is an equivalence

$$\big(\tilde{p}_S(n, H, m, p) = y\big) \simeq \big(h(n, H, m, x) = y\big),$$

for any $y : P(m)$. This equivalence is obtained from the fact that $f(p) = x$ for any $x : (m \leq n) + (m = n+1)$, i.e., the fact that $(m \leq n) + (m = n+1)$ is a proposition. Now the identifications in (i) and (ii) are obtained as a simple consequence of the computation rule for coproducts. □

We are now ready to finish the proof of Theorem 13.5.1.

Proof of Theorem 13.5.1 Using $\tilde{p}_0$ and $\tilde{p}_S$, we obtain by induction on n a function

$$\tilde{s} : \prod_{(n:\mathbb{N})} \tilde{P}(n)$$

satisfying the computation rules

$$\begin{aligned} \tilde{s}(0) &\doteq \tilde{p}_0 \\ \tilde{s}(n+1) &\doteq \tilde{p}_S(n, \tilde{s}(n)). \end{aligned}$$

Now we define

$$\mathsf{strong\text{-}ind}_{\mathbb{N}}(p_0, p_S, n) :\equiv \tilde{s}(n, n, \mathsf{refl\text{-}leq}_{\mathbb{N}}(n)),$$

where $\mathsf{refl\text{-}leq}_{\mathbb{N}}(n) : n \leq n$ is the proof of reflexivity of $\leq$.

It remains to show that $\mathsf{strong\text{-}ind}_{\mathbb{N}}$ satisfies the computation rules of the strong induction principle. The identification that computes $\mathsf{strong\text{-}ind}_{\mathbb{N}}$ at 0 is easy to obtain, because we have the judgmental equalities

$$\begin{aligned} \mathsf{strong\text{-}ind}_{\mathbb{N}}(p_0, p_S, 0) &\doteq \tilde{s}(0, 0, \mathsf{refl\text{-}leq}_{\mathbb{N}}(0)) \\ &\doteq \tilde{p}_0(0, \mathsf{refl\text{-}leq}_{\mathbb{N}}(0)) \\ &\doteq p_0. \end{aligned}$$

To construct the identification that computes $\mathsf{strong\text{-}ind}_{\mathbb{N}}$ at a successor, we start by a similar computation:

$$\begin{aligned} \mathsf{strong\text{-}ind}_{\mathbb{N}}(p_0, p_S, n+1) &\doteq \tilde{s}(n+1, n+1, \mathsf{refl\text{-}leq}_{\mathbb{N}}(n+1)) \\ &\doteq \tilde{p}_S(n, \tilde{s}(n), n+1, \mathsf{refl\text{-}leq}_{\mathbb{N}}(n+1)) \\ &= p_S(n, \tilde{s}(n)). \end{aligned}$$

The last identification is obtained from Lemma 13.5.3 (ii). Therefore we see that, in order to show that

$$p_S(n, \tilde{s}(n)) = p_S(n, (\lambda m.\, \lambda p.\, \tilde{s}(m, m, \mathsf{refl\text{-}leq}_{\mathbb{N}}(m)))),$$

we need to prove that

$$\tilde{s}(n) = \lambda m.\, \lambda p.\, \tilde{s}(m, m, \mathsf{refl\text{-}leq}_{\mathbb{N}}(m)).$$

Here we apply function extensionality, so it suffices to show that

$$\tilde{s}(n, m, p) = \tilde{s}(m, m, \mathsf{refl\text{-}leq}_{\mathbb{N}}(m))$$

for every $m : \mathbb{N}$ and $p : m \leq n$. We proceed by induction on $n : \mathbb{N}$. The base case is trivial. For the inductive step, we note that

$$\tilde{s}(n+1, m, p) = \tilde{p}_S(n, \tilde{s}(n), m, p) = \begin{cases} \tilde{s}(n, m, p) & \text{if } m \leq n \\ p_S(n, \tilde{s}(n)) & \text{if } m = n+1. \end{cases}$$

Therefore it follows by the inductive hypothesis that

$$\tilde{s}(n+1, m, p) = \tilde{s}(m, m, \mathsf{refl\text{-}leq}_{\mathbb{N}}(m))$$

if $m \leq n$ holds. In the remaining case, where $m = n+1$, note that we have

$$\begin{aligned}\tilde{s}(n+1, n+1, \mathsf{refl\text{-}leq}_{\mathbb{N}}(n+1)) &= \tilde{p}_S(n, \tilde{s}(n), n+1, \mathsf{refl\text{-}leq}_{\mathbb{N}}(n+1)) \\ &= p_S(n, \tilde{s}(n)).\end{aligned}$$

Therefore we see that we also have an identification

$$\tilde{s}(n+1, m, p) = \tilde{s}(m, m, \mathsf{refl\text{-}leq}_{\mathbb{N}}(m))$$

when $m = n+1$. This completes the proof of the computation rules for the strong induction principle of $\mathbb{N}$. □

Exercises

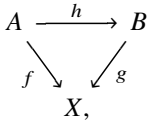
13.1 Show that the functions

$$\begin{aligned}\mathsf{inv\text{-}htpy} &: (f \sim g) \to (g \sim f) \\ \mathsf{concat\text{-}htpy}(H) &: (g \sim h) \to (f \sim h) \\ \mathsf{concat\text{-}htpy}'(K) &: (f \sim g) \to (f \sim h)\end{aligned}$$

are equivalences for every $f, g, h : \prod_{(x:A)} B(x)$. Here, $\mathsf{concat\text{-}htpy}'(K)$ is the function defined by $H \mapsto H \cdot K$.

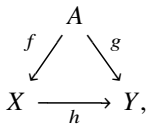
13.2 Characterize the identity types of the following types:

(a) The type $\sum_{(h:A\to B)} f \sim g \circ h$ of commuting triangles



where $f : A \to X$ and $g : B \to X$ are given.

(b) The type $\sum_{(h:X\to Y)} h \circ f \sim g$ of commuting triangles



where $f : A \to X$ and $g : A \to Y$ are given.

(c) The type $\sum_{(h:A\to B)} h(a) = b$ of base-point preserving maps, where $a : A$ and $b : B$ are given.

(d) The type $\sum_{(i:A\to X)} \sum_{(j:B\to Y)} j \circ f \sim g \circ i$ of commuting squares

$$\begin{array}{ccc} A & \xrightarrow{i} & X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[j]{} & Y, \end{array}$$

where $f : A \to B$ and $g : X \to Y$ are given.

13.3 (a) Show that for any type A the type $\mathsf{is\text{-}contr}(A)$ is a proposition.

(b) Show that for any type A and any $k \geq -2$, the type $\mathsf{is\text{-}trunc}_k(A)$ is a proposition.

13.4 Let $f : A \to B$ be a function.

(a) Show that if f is an equivalence, then the type $\sum_{(g:B\to A)} f \circ g \sim \mathsf{id}$ of sections of f is contractible.

(b) Show that if f is an equivalence, then the type $\sum_{(h:B\to A)} h \circ f \sim \mathsf{id}$ of retractions of f is contractible.

(c) Show that $\mathsf{is\text{-}equiv}(f)$ is a proposition.

(d) Show that for any two equivalences $e, e' : A \simeq B$, the canonical map

$$(e = e') \to (e \sim e')$$

is an equivalence.

(e) Show that the type $A \simeq B$ is a k-type if both A and B are k-types.

13.5 (a) Show that $\mathsf{is\text{-}path\text{-}split}(f)$ and $\mathsf{is\text{-}coh\text{-}invertible}(f)$ are propositions for any map $f : A \to B$. Conclude that we have equivalences

$$\mathsf{is\text{-}equiv}(f) \simeq \mathsf{is\text{-}path\text{-}split}(f) \simeq \mathsf{is\text{-}coh\text{-}invertible}(f).$$

(b) Construct for any type A an equivalence

$$\mathsf{has\text{-}inverse}(\mathsf{id}_A) \simeq \left(\mathsf{id}_A \sim \mathsf{id}_A\right).$$

Note: We will use this fact in Exercise 21.6 to show that there are types for which $\mathsf{is\text{-}invertible}(\mathsf{id}_A) \not\simeq \mathsf{is\text{-}equiv}(\mathsf{id}_A)$.

13.6 The type A is empty.

13.7 Consider a type A. Show that the following are equivalent:

(i) The type $\prod_{(x:A)} P(x)$ is contractible for any family P of types over A. This property is the **dependent universal property of an empty type**.

(ii) The type $A \to X$ is contractible for any type X. This property is the **universal property of an empty type**.

13.8 Consider a type A. Show that the following are equivalent:

(i) The type A is contractible.
(ii) The type A comes equipped with a point $a : A$, and the map

$$\left(\prod_{(x:A)} P(x)\right) \to P(a)$$

given by $f \mapsto f(a)$ is an equivalence for any type family P over A. This property is the **dependent universal property of a contractible type**.
(iii) The type A comes equipped with a point $a : A$, and the map

$$(A \to X) \to X$$

given by $f \mapsto f(a)$ is an equivalence for any type X. This property is the **universal property of a contractible type**.
(iv) The type A comes equipped with a point $a : A$, and the map

$$(A \to A) \to A$$

given by $f \mapsto f(a)$ is an equivalence.
(v) The map

$$X \to (A \to X)$$

given by $x \mapsto \lambda y.\, x$ is an equivalence for any type X.
(vi) The map

$$A \to (A \to A)$$

given by $x \mapsto \lambda y.\, x$ is an equivalence.

13.9 Consider two types A and B. Show that the map

$$\left(\prod_{(z:A+B)} P(z)\right) \to \left(\prod_{(x:A)} P(\mathsf{inl}(x))\right) \times \left(\prod_{(y:B)} P(\mathsf{inr}(b))\right)$$

given by $f \mapsto (f \circ \mathsf{inl}, f \circ \mathsf{inr})$ is an equivalence for any type family P over $A + B$. This property is the **dependent universal property of the coproduct of A and B**. Conclude that the map

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

given by $f \mapsto (f \circ \mathsf{inl}, f \circ \mathsf{inr})$ is an equivalence for any type X. This latter property is the **universal property of the coproduct of A and B**.

13.10 Prove the **universal property of** $\mathbb{N}$: For any type X equipped with $x : X$ and $f : X \to X$, the type

$$\sum_{(h:\mathbb{N}\to X)} (h(0_{\mathbb{N}}) = x) \times (h \circ \mathsf{succ}_{\mathbb{N}} \sim f \circ h)$$

is contractible.

13.11 Show that $\mathbb{N}$ satisfies **ordinal induction**, i.e., construct for any type family P over $\mathbb{N}$ a function $\mathsf{ord\text{-}ind}_{\mathbb{N}}$ of type

$$\left(\prod_{(k:\mathbb{N})}\left(\prod_{(m:\mathbb{N})}(m < k) \to P(m)\right) \to P(k)\right) \to \prod_{(n:\mathbb{N})} P(n).$$

Moreover, prove that

$$\mathsf{ord\text{-}ind}_{\mathbb{N}}(h, n) = h(n, \lambda m.\, \lambda p.\, \mathsf{ord\text{-}ind}_{\mathbb{N}}(h, m))$$

for any $n : \mathbb{N}$ and any $h : \prod_{(k:\mathbb{N})}\left(\prod_{(m:\mathbb{N})}(m < k) \to P(m)\right) \to P(k)$.

13.12 (a) Consider a family of k-truncated maps $f_i : A_i \to B_i$ indexed by $i : I$. Show that the map

$$\lambda h.\, \lambda i.\, f_i(h(i)) : \left(\prod_{(i:I)} A_i\right) \to \left(\prod_{(i:I)} B_i\right)$$

is also k-truncated.

(b) Consider an equivalence $e : I \simeq J$, and a family of equivalences $f_i : A_i \simeq B_{e(i)}$ indexed by $i : I$, where A is a family of types indexed by I and B family of types indexed by J. Show that the map

$$\lambda h.\, \lambda j.\, f_{e^{-1}(j)}(h(e^{-1}(j))) : \left(\prod_{(i:I)} A_i\right) \to \left(\prod_{(j:J)} B_j\right)$$

is an equivalence.

(c) Consider a family of maps $f_i : A_i \to B_i$ indexed by $i : I$. Show that the the following are equivalent:

(i) Each f_i is k-truncated.

(ii) For every map $\alpha : X \to I$, the map

$$\lambda h.\, \lambda x.\, f_{\alpha(x)}(h(x)) : \left(\prod_{(x:X)} A_{\alpha(x)}\right) \to \left(\prod_{(x:X)} B_{\alpha(x)}\right)$$

is k-truncated.

(d) Show that for any map $f : A \to B$ the following are equivalent:

(i) The map f is k-truncated.

(ii) For every type X, the postcomposition function

$$f \circ - : (X \to A) \to (X \to B)$$

is k-truncated.

In particular, f is an equivalence if and only if $f \circ -$ is an equivalence, and f is an embedding if and only if $f \circ -$ is an embedding.

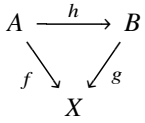
13.13 Show that Π-*types distribute over coproducts*, i.e., construct for any type X and any two families A and B over X an equivalence from the type $\prod_{(x:X)} A(x) + B(x)$ to the type

$$\sum_{(f:X\to\mathsf{Fin}_2)}\left(\prod_{(x:X)} A(x)^{f(x)=0}\right) \times \left(\prod_{(x:X)} B(x)^{f(x)=1}\right).$$

There are 306 exercises.

Ljubljana, November 10, 2021

13.14 Consider a commuting triangle



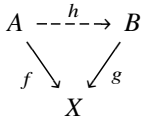
with $H : f \sim g \circ h$.

(a) Show that if h has a section, then $\mathsf{sec}(g)$ is a retract of $\mathsf{sec}(f)$.
(b) Show that if g has a retraction, then $\mathsf{retr}(h)$ is a retract of $\mathsf{sec}(f)$.

13.15 For any two maps $f : A \to X$ and $g : B \to X$, define the type of **morphisms from f to g over X** by

$$\mathsf{hom}_X(f, g) :\equiv \textstyle\sum_{(h:A\to B)} f \sim g \circ h.$$

In other words, the type $\mathsf{hom}_X(f, g)$ is the type of maps $h : A \to B$ equipped with a homotopy witnessing that the triangle

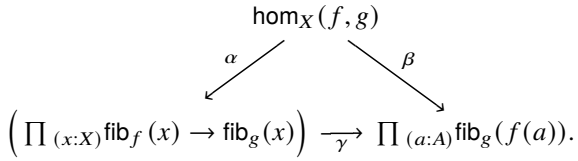


commutes.

(a) Consider a family P of types over X. Show that the map

$$\left(\textstyle\prod_{(x:X)} \mathsf{fib}_f(x) \to P(x)\right) \to \left(\textstyle\prod_{(a:A)} P(f(a))\right)$$

given by $h \mapsto h_{f(a)}(a, \mathsf{refl}_{f(a)})$ is an equivalence.
(b) Construct three equivalences α, β, and γ as shown in the following diagram, and show that this triangle commutes:



Given a morphism $(h, H) : \mathsf{hom}_X(f, g)$ over X, we also say that $\alpha(h, H)$ is its **action on fibers**.
(c) Given $(h, H) : \mathsf{hom}_X(f, g)$, show that the following are equivalent:
  (i) The map $h : A \to B$ is an equivalence.
  (ii) The action on fibers

$$\alpha(h, H) : \textstyle\prod_{(x:X)} \mathsf{fib}_f(x) \to \mathsf{fib}_g(x)$$

is a family of equivalences.

(iii) The precomposition function

$$- \circ (h, H) : \mathsf{hom}_X(g, i) \to \mathsf{hom}_X(f, i)$$

given by $(k, K) \circ (h, H) :\equiv (k \circ h, H \cdot (K \cdot h))$ is an equivalence for each map $i : C \to X$.

Conclude that the type $\sum_{(h:A \simeq B)} f \sim g \circ h$ is equivalent to the type of families of equivalences

$$\prod_{(x:X)} \mathsf{fib}_f(x) \simeq \mathsf{fib}_g(x).$$

13.16 Let A and B be sets. Show that type type $A \simeq B$ of equivalences from A to B is equivalent to the type $A \cong B$ of **isomorphisms** from A to B, i.e., the type of quadruples (f, g, H, K) consisting of

$$f : A \to B$$
$$g : B \to A$$
$$H : f \circ g = \mathsf{id}_B$$
$$K : g \circ f = \mathsf{id}_A.$$

13.17 Suppose that $A : I \to \mathcal{U}$ is a type family over a set I with decidable equality. Show that

$$\left(\prod_{(i:I)} \mathsf{is\text{-}contr}(A_i)\right) \leftrightarrow \mathsf{is\text{-}contr}\left(\prod_{(i:I)} A_i\right).$$

14 Propositional truncations

It is common in mathematics to express the property that a certain type of objects is inhabited, without imposing extra structure on those objects. For example, when we assert the property that a set is finite, then we only claim that there exists a bijection to a standard finite set $\{0, \ldots, n-1\}$ for some n, not that the set is equipped with such a bijection. There is indeed a conceptual difference between a finite set and a set equipped with a bijection to a standard finite set. The latter concept is that of a finite *totally ordered* set. This difference is due to the fact finiteness is a property, whereas there may be many different bijections to a standard finite set.

A similar observation can be made in the case of the image of a map. Note that being in the image of a given map $f : A \to B$ is a property. When we claim that $b : B$ is in the image of f, then we only claim that the type of $a : A$ such that $f(a) = b$ is inhabited. On the other hand, we saw in Exercise 10.8 that the

type of $b : B$ equipped with an $a : A$ such that $f(a) = b$ is equivalent to the type A, i.e., we have an equivalence

$$A \simeq \sum_{(b:B)} \sum_{(a:A)} f(a) = b.$$

Something is clearly off here, because the type A is often not a subtype of the type B, while we would expect the image of f to be a subtype of B. Therefore we see that the type $\sum_{(a:A)} f(a) = b$ does not quite capture the concept of b being in the image of f. The difference is again due to the fact that $\mathsf{fib}_f(b)$ is often not a proposition, whereas we are looking to express the proposition that the preimage of f at b is inhabited.

To correctly capture the concepts of finiteness and the image of a map in type theory, and many further mathematical concepts, we need a way to assert the *proposition* that a type is inhabited. The proposition that a type A is inhabited is called the propositional truncation of A.

14.1 The universal property of propositional truncations

The propositional truncation of a type A is a proposition $\|A\|$ equipped with a map

$$\eta : A \to \|A\|.$$

This map ensures that if we have an element $a : A$, then the proposition $\|A\|$ that A is inhabited holds. The complete specification of the propositional truncation includes the universal property of the map η. In this section we will specify in full generality when a map $f : A \to P$ into a proposition P is a propositional truncation.

Definition 14.1.1 Let A be a type, and let $f : A \to P$ be a map into a proposition P. We say that f **is a propositional truncation** of A if for every proposition Q, the precomposition map

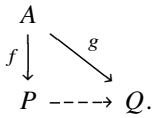
$$- \circ f : (P \to Q) \to (A \to Q)$$

is an equivalence. This property of f is called the **universal property of the propositional truncation of** A.

Remark 14.1.2 Using the fact that equivalences are maps that have contractible fibers, we can reformulate the universal property of the propositional truncation. Note that the fiber of the precomposition map $- \circ f : (P \to Q) \to (A \to Q)$ at a map $g : A \to Q$ is the type.

$$\sum_{(h:P\to Q)} h \circ f = g$$

Therefore we see that if f satisfies the universal property of the propositional truncation, then these fibers are contractible. In other words, for each map $g : A \to Q$ into a proposition Q there is a unique map $h : P \to Q$ for which $h \circ f = g$. We also say that every map $g : A \to Q$ into a proposition Q *extends* uniquely along f, as indicated in the diagram



Remark 14.1.3 For any two propositions P and P', a map $f : P \to P'$ is an equivalence if and only if there is a function $g : P' \to P$. To see this, simply note that any such function g is an inverse of f, because any two elements in P and any two elements in P' are equal.

Note that the type $X \to Q$ is a proposition, for any type X and any proposition Q. Using the previous observation, it therefore follows that the map $(P \to Q) \to (A \to Q)$ is an equivalence as soon as there is a map in the converse direction. In other words, to prove that a map $f : A \to P$ into a proposition P satisfies the universal property of the propositional truncation of A, it suffices to construct a function

$$(A \to Q) \to (P \to Q)$$

for every proposition Q.

In the following proposition we show that the propositional truncation of a type A is uniquely determined up to equivalence, if it exists. In other words, any two propositional truncations of a type A must be equivalent.

Proposition 14.1.4 *Let A be a type, and consider two maps*

$$f : A \to P \qquad \text{and} \qquad f' : A \to P'$$

into two propositions P and P'. If any two of the following three assertions hold, so does the third:

(i) *The map f is a propositional truncation of A.*
(ii) *The map f' is a propositional truncation of A.*
(iii) *There is a (unique) equivalence $P \simeq P'$.*

Proof We first show that (i) and (ii) together imply (iii). If f and f' are both propositional truncations of A, then we have maps $P \to P'$ and $P' \to P$ by the universal properties of f and f'. Since P and P' are both propositions, it follows that $P \simeq P'$. For the uniqueness claim, note that the type $P \simeq P'$ is itself a proposition.

Finally we show that (iii) implies that (i) holds if and only if (ii) holds. Suppose we have an equivalence $P \simeq P'$, let Q be an arbitrary proposition, and consider the triangle

$$\begin{array}{ccc} & (A \to Q) & \\ \swarrow & & \searrow \\ (P \to Q) & \longleftrightarrow & (P' \to Q), \end{array}$$

where the fact that $(P \to Q) \leftrightarrow (P' \to Q)$ holds follows from the assumption that P is equivalent to P'. We see from this triangle that

$$\Big((A \to Q) \to (P \to Q)\Big) \leftrightarrow \Big((A \to Q) \to (P' \to Q)\Big),$$

and this implies that (i) holds if and only if (ii) holds. □

Remark 14.1.5 One might be tempted to think that a type is inhabited if and only if it is nonempty. Recall that a type A is nonempty if it satisfies the property $\neg\neg A$. Indeed, the type $\neg\neg A$ is a proposition, and it comes equipped with a map $A \to \neg\neg A$. It is therefore natural to wonder whether the map $A \to \neg\neg A$ satisfies the universal property of the propositional truncation.

Recall that we have shown in Exercise 4.3 (b) that any map $A \to \neg\neg Q$ extends to a map $\neg\neg A \to \neg\neg Q$, as indicated in the diagram

$$\begin{array}{ccc} A & & \\ \downarrow & \searrow & \\ \neg\neg A & \dashrightarrow & \neg\neg Q. \end{array}$$

It follows that the natural map

$$(\neg\neg A \to \neg\neg Q) \to (A \to \neg\neg Q)$$

given by precomposition by $A \to \neg\neg A$ is an equivalence. However, this only gives us a universal property with respect to doubly negated propositions and there is no way to prove the more general universal property of the propositional truncation for the map $A \to \neg\neg A$. In fact, propositional truncations are not guaranteed to exist in Martin Löf's dependent type theory, the way it is set up in Chapter I. We will therefore add new rules to the type theory to ensure their existence.

14.2 Propositional truncations as higher inductive types

We have given a specification of the propositional truncation of a type A, and we have seen that this specification by a universal property determines the

propositional truncation up to equivalence if it exists. However, the propositional truncation is not guaranteed to exist, so we will add new rules to the type theory that ensure that any type has a propositional truncation. We do this by presenting the propositional truncation of a type A as a higher inductive type. The propositional truncation $\|A\|$ of a type A was one of the first examples of a higher inductive type, along with the circle, which we will discuss in Sections 20 and 21.

The idea of higher inductive types is similar to the idea of ordinary inductive types, with the added feature that constructors of higher inductive types can also be used to generate *identifications*. In other words, higher inductive types may be specified by two kinds of constructors:

(i) The *point constructors* are used to generate elements of the higher inductive types.
(ii) The *path constructors* are used to generate identifications between elements of the higher inductive type.

The induction principle of the higher inductive type then tells us how to construct sections of families over it. The rules for higher inductive types therefore come in four sets, just as the rules for ordinary inductive types in Section 4: the formation rule, the constructors, the induction principle, and the computation rules.

The formation rules and the constructors

The formation rule of the propositional truncation postulates that for every type A we can form the propositional truncation of A. The formation rule is therefore as follows:

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \|A\| \text{ type.}}$$

Furthermore, we will assume that all universes are closed under propositional truncations. In other words, for any universe $\mathcal{U}$ we will assume the rules

$$\frac{}{X : \mathcal{U} \vdash \|X\|^{\check{}} : \mathcal{U}} \qquad \frac{}{X : \mathcal{U} \vdash \mathcal{T}(\|X\|^{\check{}}) \doteq \|\mathcal{T}(X)\| \text{ type}}$$

The constructors of a (higher) inductive type tell what structure the type comes equipped with. In the case of a higher inductive type there may be point constructors and path constructors. The point constructors generate elements of the higher inductive type, and the path constructors generate identifications between those elements. In the case of the propositional truncation, there is one

point constructor and one path constructor:

$$\eta : A \to \|A\|$$
$$\alpha : \prod_{(x,y:\|A\|)} x = y.$$

The point constructor η is sometimes called the **unit** of the propositional truncation. It gives us that any element of A also generates an element of $\|A\|$. The path constructor α simply identifies any two elements of $\|A\|$. Therefore it follows immediately that $\|A\|$ is a proposition.

Lemma 14.2.1 *For any type A, the type $\|A\|$ is a proposition.* □

The induction principle and computation rules

The induction principle for the propositional truncation tells us how to construct dependent functions

$$h : \prod_{(t:\|A\|)} Q(t).$$

The induction principle will imply that such a dependent function h is entirely determined by its behavior on the constructors of $\|A\|$. The type $\|A\|$ has two constructors: a point constructor η and a path constructor α, so we have two cases to consider:

(i) Applying h to points of the form $\eta(a)$ gives us a dependent function

$$h \circ \eta : \prod_{(a:A)} Q(\eta(a)).$$

The induction principle of $\|A\|$ has therefore the requirement that we can construct

$$f : \prod_{(a:A)} Q(\eta(a))$$

(ii) To apply h to the paths $\alpha(x, y)$, we need to use the dependent action on paths from Definition 5.4.2. For each $x, y : \|A\|$ we obtain an identification

$$\mathsf{apd}_h(\alpha(x, y)) : \mathsf{tr}_Q(\alpha(x, y), h(x)) = h(y)$$

in the type $Q(y)$. Note, however, that $h(x)$ and $h(y)$ are not determined by our choice of $f : \prod_{(a:A)} Q(\eta(a))$. The second requirement of the induction principle of $\|A\|$ is therefore that, no matter what values h takes, they must always be related via the dependent action on paths of h. This second requirement is therefore that

$$\mathsf{tr}_P(\alpha(x, y), u) = v$$

for any $u : Q(x)$ and $v : Q(y)$.

Definition 14.2.2 The **induction principle** of the propositional truncation $\|A\|$ of A asserts that for any family Q of types 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 any $u : Q(x)$, $v : Q(y)$ and any $x, y : \|A\|$, then we obtain a dependent function

$$h : \prod_{(t:\|A\|)} Q(t)$$

equipped with a homotopy $h \circ \eta \sim f$.

Remark 14.2.3 In fact, a family Q over $\|A\|$ satisfies the second requirement in the induction principle of the propositional truncation if and only if Q is a family of propositions. To see this, simply note that transporting along $\alpha(x, y)$ is an embedding. Therefore we have

$$(\mathsf{tr}_Q(\alpha(x,y),u) = \mathsf{tr}_Q(\alpha(x,y),v)) \simeq (u = v)$$

for any $u, v : Q(x)$. By assumption, there is an identification on the left hand side, so any two elements u and v in $Q(x)$ are equal.

Since the induction principle of the propositional truncation is only applicable to families of propositions over $\|A\|$, it also follows that there are no interesting computation rules to state: any identification in a proposition just holds.

The universal property

We have now completed the description of the propositional truncation as a higher inductive type, so it is time to show that it meets the specification we gave for the propositional truncations. In other words, we have to show that the map $\eta : A \to \|A\|$ satisfies the universal property of the propositional truncation.

Theorem 14.2.4 *The map $\eta : A \to \|A\|$ satisfies the universal property of the propositional truncation.*

Proof In order to prove that $\eta : A \to \|A\|$ satisfies the universal property of the propositional truncation of A, it suffices to construct a map

$$(A \to Q) \to (\|A\| \to Q)$$

for any proposition Q. Consider a map $f : A \to Q$. Then we will construct a function $\|A\| \to Q$ by the induction principle of the propositional truncation.

We have to provide a function $A \to Q$, which we have assumed already, and we have to show that

$$\mathsf{tr}_{\lambda x.\, Q}(\alpha(x, y), u) = v.$$

for any $u, v : Q$ and any $x, y : \|A\|$. However, we have such identifications by the assumption that Q is a proposition, so the proof is complete. □

One simple application of the universal property of the propositional truncation is that $\|-\|$ acts on functions in a functorial way.

Proposition 14.2.5 *There is a map*

$$\|-\| : (A \to B) \to (\|A\| \to \|B\|)$$

for any two types A and B, such that

$$\|\mathsf{id}\| \sim \mathsf{id}$$
$$\|g \circ f\| \sim \|g\| \circ \|f\|.$$

Proof For any $f : A \to B$, the map $\|f\| : \|A\| \to \|B\|$ is defined to be the unique extension

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle\eta}\downarrow & & \downarrow{\scriptstyle\eta} \\ \|A\| & \xrightarrow[\|f\|]{} & \|B\|. \end{array}$$

To see that $\|-\|$ preserves identity maps and compositions, simply note that $\mathsf{id}_{\|A\|}$ is an extension of id_A, and that $\|g\| \circ \|f\|$ is an extension of $g \circ f$. Hence the homotopies are obtained by uniqueness. □

14.3 Logic in type theory

In Section 7.2 we interpreted logic in type theory via the Curry-Howard correspondence, which stiuplates that disjunction ($\vee$) is interpreted by coproducts and the existential quantifier ($\exists$) is interpreted by Σ-types. However, when the existential quantifier is interpreted by Σ-types, then it is not possible to express certain concepts correctly, such as finiteness of a type or being in the image a map, and therefore we will add a second interpretation of logic in type theory, where logical propositions are interpreted by type theoretic propositions, i.e., the types of truncation level -1.

We have seen that the propositions are closed under cartesian products, implication, and dependent products indexed by arbitrary types. However, they are not closed under coproducts, and if P is a family of propositions over a type

A, then it is not necessarily the case that $\sum_{(x:A)} P(x)$ is a proposition. We will therefore use propositional truncations to interpret disjunctions and existential quantifiers in type theory.

Definition 14.3.1 Given two propositions P and Q, we define their **disjunction**

$$P \vee Q :\equiv \|P + Q\|.$$

Proposition 14.3.2 *Consider two propositions P and Q. Then the disjunction $P \vee Q$ comes equipped with maps $i : P \to P \vee Q$ and $j : Q \to P \vee Q$. Moreover, for any proposition R, we have*

$$(P \vee Q \to R) \leftrightarrow ((P \to R) \times (Q \to R)).$$

Proof The maps i and j are defined by

$$i :\equiv \eta \circ \mathsf{inl}$$
$$j :\equiv \eta \circ \mathsf{inr}.$$

Now consider the following composition of maps, for an arbitrary proposition R:

$$(P \vee Q \to R) \xrightarrow{-\circ\eta} (P + Q \to R) \xrightarrow{h \mapsto (h\circ\mathsf{inl}, h\circ\mathsf{inr})} (P \to R) \times (Q \to R).$$

The first map is an equivalence by the universal property of the propositional truncation, and the second map is an equivalence by the universal property of coproducts (Exercise 13.9). □

Definition 14.3.3 Given a family P of propositions over a type A, we define the **existential quantification**

$$\exists_{(x:A)} P(x) :\equiv \left\| \sum_{(x:A)} P(x) \right\|.$$

Proposition 14.3.4 *Consider a family P of propositions over a type A. Then the existential quantification $\exists_{(x:A)} P(x)$ comes equipped with a dependent function*

$$\prod_{(a:A)} \left(P(a) \to \exists_{(x:A)} P(x)\right).$$

Furthermore, for any proposition Q, we have

$$\left(\left(\exists_{(x:A)} P(x)\right) \to Q\right) \leftrightarrow \left(\prod_{(x:A)} P(x) \to Q\right).$$

Proof The dependent function $\varepsilon : \prod_{(a:A)} \left(P(a) \to \exists_{(x:A)} P(x)\right)$ is given by $\varepsilon(a, p) :\equiv \eta(a, p)$. Now consider the following composition of maps

$$\left(\left(\exists_{(x:A)} P(x)\right) \to Q\right) \longrightarrow \left(\left(\sum_{(x:A)} P(x)\right) \to Q\right) \longrightarrow \left(\prod_{(x:A)} P(x) \to Q\right).$$

The first map in this composite is an equivalence by the universal property of the

propositional truncation, and the second map is an equivalence by the universal property of Σ-types (Theorem 13.3.1). □

In the following table we give an overview of the interpretation of the logical connectives using the propositions in type theory.

logical connective	interpretation in type theory
$\top$	$\mathbf{1}$
$\bot$	$\emptyset$
$P \Rightarrow Q$	$P \to Q$
$P \wedge Q$	$P \times Q$
$P \vee Q$	$\|P + Q\|$
$P \Leftrightarrow Q$	$P \leftrightarrow Q$
$\exists_{(x:A)} P(x)$	$\left\| \sum_{(x:A)} P(x) \right\|$
$\forall_{(x:A)} P(x)$	$\prod_{(x:A)} P(x)$

14.4 Mapping propositional truncations into sets

The universal property of the propositional truncation only applies when we want to define a map into a proposition. However, in some situations we might want to map the propositional truncation into a type that is not a proposition. Here we will see what we might do in such a case.

One strategy, if we want to define a map $\|A\| \to X$, is to find a type family P over X such that the type $\sum_{(x:X)} P(x)$ is a proposition. In that case, we may use the universal property of the propositional truncation to obtain a map $\|A\| \to \sum_{(x:X)} P(x)$ from a map $A \to \sum_{(x:X)} P(x)$, and then we simply compose with the projection map.

Example 14.4.1 Consider a decidable subtype P of the natural numbers. We claim that there is a function

$$\left\| \sum_{(x:\mathbb{N})} P(x) \right\| \to \sum_{(x:\mathbb{N})} P(x).$$

Of course, we cannot directly use the universal property of the propositional truncation here. However, there is at most one *minimal* natural number x in P. In other words, we claim that the type

$$\sum_{(x:\mathbb{N})} P(x) \times \mathsf{is\text{-}lower\text{-}bound}_P(x) \tag{$*$}$$

is a proposition. To see this, note that the type $\mathsf{is\text{-}lower\text{-}bound}_P(x)$ is a proposition. By the assumption that each $P(x)$ is a proposition, it now follows that

any two natural numbers $x, y : \mathbb{N}$ that are in P and that are both lower bounds of P are equal as elements in the type of Eq. (∗) if and only if they are equal as natural numbers. Furthermore, since both x and y are lower bounds of P, it follows that $x \leq y$ and $y \leq x$, so indeed $x = y$ holds.

By the observation that the type in Eq. (∗) is a proposition, we may define a map

$$\left\|\sum_{(x:\mathbb{N})} P(x)\right\| \to \sum_{(x:\mathbb{N})} P(x) \times \mathsf{is\text{-}lower\text{-}bound}_P(x)$$

by the universal property of the propositional truncation. A map

$$\sum_{(x:\mathbb{N})} P(x) \to \sum_{(x:\mathbb{N})} P(x) \times \mathsf{is\text{-}lower\text{-}bound}_P(x)$$

was constructed in Theorem 8.3.2 using the decidability of P.

As a corollary of this observation, we observe that there is also a map

$$\left\|\sum_{(x:\mathsf{Fin}_k)} P(x)\right\| \to \sum_{(x:\mathsf{Fin}_k)} P(x)$$

for any decidable subtype P over Fin_k.

Remark 14.4.2 The function of type

$$\left\|\sum_{(x:\mathbb{N})} P(x)\right\| \to \sum_{(x:\mathbb{N})} P(x)$$

we constructed in Example 14.4.1 for decidable subtypes of $\mathbb{N}$ is a rare case in which it is possible to obtain a function

$$\|A\| \to A.$$

We say that the type A satisfies the **principle of global choice** if there is such a function $\|A\| \to A$. Using the univalence axiom, we will see in Corollary 17.4.3 that not every type satisfies the principle of global choice.

More generally, we may wish to define a map $\|A\| \to B$ where the type B is a set. In this situation it is helpful to think of the propositional truncation of A as the quotient of the type A by the equivalence relation that relates every two elements of A with each other. Propositional truncations can therefore also be characterized by the universal property of this quotient, which can be used to extend maps $f : A \to B$ to maps $\|A\| \to B$ when the type B is a set. The idea is that a map $f : A \to B$ into a set B extends to a map $\|A\| \to B$ if it satisfies $f(x) = f(y)$ for all $x, y : A$.

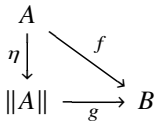
Definition 14.4.3 A map $f : A \to B$ is said to be **weakly constant** if it comes equipped with an element of type

$$\mathsf{is\text{-}weakly\text{-}constant}(f) :\equiv \prod_{(x,y:A)} f(x) = f(y).$$

Remark 14.4.4 A constant map $A \to B$ is a map of the form const_b. A map $f : A \to B$ is therefore constant if comes equipped with an element $b : B$ and a homotopy $f \sim \mathsf{const}_b$. This is a stronger notion than the notion of weakly constant maps, which doesn't require there to be an element in B.

One of the differences between constant maps and weakly constant maps manifests itself as follows: A type A is contractible if and only if the identity map on A is constant, while a type A is a proposition if and only if the identity map on A is weakly constant.

Lemma 14.4.5 *Consider a commuting triangle*



where B is an arbitrary type. Then the map f is weakly constant.

Proof Since f is assumed to be homotopic to $g \circ \eta$, it suffices to show that $g \circ \eta$ is weakly constant. For any $x, y : A$, we have the identification $\alpha(x, y) : \eta(x) = \eta(y)$ in $\|A\|$. Using the action on paths of g, we obtain the identification

$$\mathsf{ap}_g(\alpha(x, y)) : g(\eta(x)) = g(\eta(y))$$

in B. □

We now show, in a theorem due to Kraus [**Kraus**], that any weakly constant map $f : A \to B$ into a set B extends uniquely to a map $\|A\| \to B$. We therefore conclude that, in order to define a map $\|A\| \to B$ into a set B it suffices to define a map $f : A \to B$ and show that it is weakly constant.

Theorem 14.4.6 (Kraus) *Let A be a type and let B be a set. Then the map*

$$(\|A\| \to B) \to \textstyle\sum_{(f:A\to B)} \prod_{(x,y:A)} f(x) = f(y)$$

given by $g \mapsto (g \circ \eta, \lambda x.\, \lambda y.\, \mathsf{ap}_g(\alpha(x, y)))$ is an equivalence.

Proof Consider a map $f : A \to B$ equipped with $H : \prod_{(x,y:A)} f(x) = f(y)$. We first show that f extends in at most one way to a map $\|A\| \to B$. Let $g, h : \|A\| \to B$ be two maps equipped with homotopies $f \sim g \circ \eta$ and $f \sim h \circ \eta$. In order to construct a homotopy $g \sim h$, note that each identity type $g(x) = h(x)$ is a proposition by the assumption that B is a set. We can therefore construct a homotopy $g \sim h$ by the induction principle of propositional truncations. By the induction principle, it suffices to construct a homotopy $g \circ \eta \sim h \circ \eta$, which we obtain from the homotopies $f \sim g \circ \eta$ and $f \sim h \circ \eta$.

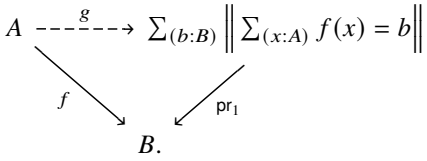
Since we've already proven uniqueness, it remains to construct an extension of the map f. We first claim that the type

$$\sum_{(b:B)} \left\| \sum_{(x:A)} f(x) = b \right\|$$

is a proposition. To see this, consider two elements b and b' in this subtype of B. It suffices to show that $b = b'$. Since B is assumed to be a set, the identity type $b = b'$ is a proposition. Therefore we may assume an element $x : A$ equipped with $p : f(x) = b$ and an element $x' : A$ equipped with $p' : f(x') = b'$. Using the assumption that f is weakly constant, we obtain the identification

$$b \overset{p^{-1}}{=\!=} f(x) \overset{H(x,x')}{=\!=} f(x') \overset{p'}{=\!=} b'.$$

Now we observe that the map $f : A \to B$ factors uniquely as follows



Indeed, the map g is given by $x \mapsto (f(x), \eta(x, \mathsf{refl}))$. Since the codomain of g is a proposition, we obtain via the universal property of the propositional truncation of A a unique map $h : \|A\| \to \sum_{(b:B)} \left\| \sum_{(x:A)} f(x) = b \right\|$ equipped with a homotopy $g \sim h \circ \eta$. Now we obtain the map $\mathsf{pr}_1 \circ h : \|A\| \to B$ equipped with the concatenated homotopy

$$(\mathsf{pr}_1 \circ h) \circ \eta \doteq \mathsf{pr}_1 \circ (h \circ \eta) \sim \mathsf{pr}_1 \circ g \sim f. \qquad \square$$

Exercises

14.1 Let A be a type. Show that

(a) $\|\|A\|\| \leftrightarrow \|A\|$.
(b) $\|\mathsf{is\text{-}decidable}(A)\| \leftrightarrow \mathsf{is\text{-}decidable}\|A\|$.
(c) $\mathsf{is\text{-}decidable}(A) \to (\|A\| \to A)$.
(d) $\neg\neg\|A\| \leftrightarrow \neg\neg A$.
(e) $\|A\| \vee \|B\| \leftrightarrow \|A + B\|$.
(f) $\exists_{(x:A)} \|B(x)\| \leftrightarrow \|\sum_{(x:A)} B(x)\|$.

14.2 Show that the relation $x, y \mapsto \|x = y\|$ is an equivalence relation, on any type.

14.3 Consider two maps $f : A \to P$ and $g : B \to Q$ into propositions P and

Q. Show that if both f and g are propositional truncations then the map $f \times g : A \times B \to P \times Q$ is also a propositional truncation. Conclude that

$$\|A \times B\| \simeq \|A\| \times \|B\|.$$

14.4 Consider a map $f : A \to P$ into a proposition P. We say that f satisfies the **dependent universal property of the propositional truncation** of A, if for any family Q of propositions over P, the precomposition function

$$- \circ f : \left(\prod_{(p:P)} Q(p)\right) \to \left(\prod_{(x:A)} Q(f(x))\right)$$

is an equivalence. Show that the following are equivalent:

(i) The map f is a propositional truncation.
(ii) The map f satisfies the dependent universal property of the propositional truncation.

14.5 Consider a map $f : A \to P$ into a proposition P.

(a) Show that if there is a map $g : P \to A$, then f is a propositional truncation. Conclude that for any type A equipped with a point $a : A$, the constant map

$$\mathsf{const}_\star : A \to \mathbf{1}$$

is a propositional truncation of A.

(b) Show that if A is a proposition, then f is a propositional truncation if and only if f is an equivalence. Conclude that if A is a proposition, then the identity function $\mathsf{id} : A \to A$ is a propositional truncation.

14.6 Consider a type A equipped with an element $d : \mathsf{is\text{-}decidable}(A)$.

(a) Define a function $f : A \to \sum_{(x:A)} \mathsf{inl}(x) = d$ and show that f is a propositional truncation of A.

(b) Consider the function $\pi : \mathsf{is\text{-}decidable}(A) \to \mathsf{Fin}_2$ defined by

$$\pi(\mathsf{inl}(x)) := 1$$
$$\pi(\mathsf{inr}(x)) := 0.$$

Define a function $g : A \to (\pi(d) = 1)$ and show that g is a propositional truncation of A.

14.7 Consider a family B of $(k+1)$-truncated types over the propositional truncation $\|A\|$ of a type A. Show that the map

$$\left(\prod_{(x:\|A\|)} B(x)\right) \to \left(\prod_{(x:A)} B(x)\right)$$

given by $f \mapsto f \circ \eta$ is a k-truncated map.

14.8 Consider a universe $\mathcal{U}$, let P_1 and P_2 be propositions in $\mathcal{U}$, and furthermore, let P be a family of propositions in $\mathcal{U}$ over a type A in $\mathcal{U}$. Construct the following equivalences:

$$\begin{aligned}
\top &\simeq \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} Q \to Q,\\
\bot &\simeq \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} Q,\\
\|A\| &\simeq \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} (A \to Q) \to Q,\\
P_1 \vee P_2 &\simeq \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} (P_1 \to Q) \to ((P_2 \to Q) \to Q),\\
P_1 \wedge P_2 &\simeq \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} (P_1 \to (P_2 \to Q)) \to Q,\\
P_1 \Rightarrow P_2 &\simeq \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} P_1 \to ((P_2 \to Q) \to Q),\\
\neg A &\simeq \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} A \to Q,\\
\exists_{(x:A)} P(x) &\simeq \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} \left(\prod_{(x:A)} P(x) \to Q\right) \to Q,\\
\forall_{(x:A)} P(x) &\simeq \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} \prod_{(x:A)} (P(x) \to Q) \to Q\\
\|a = x\| &\simeq \textstyle\prod_{(Q:A\to\mathsf{Prop}_\mathcal{U})} Q(a) \to Q(x).
\end{aligned}$$

These are the **impredicative encodings** of the logical operators. *Note:* It has the appearence that we could have defined $\|A\|$ by its impredicative encoding. There is, however, a subtle issue if we take this as a definition: The map

$$A \to \textstyle\prod_{(Q:\mathsf{Prop}_\mathcal{U})} (A \to Q) \to Q$$

only satisfies the universal property of the propositional truncation with respect to propositions that are equivalent to propositions in $\mathcal{U}$.

14.9 In this exercise we introduce the **interval** as a higher inductive type $\mathbb{I}$, equipped with two point constructors and one path constructor

$$\begin{aligned}
&\mathsf{source}, \mathsf{target} : \mathbb{I}\\
&\mathsf{path} : \mathsf{source} = \mathsf{target}.
\end{aligned}$$

The induction principle of $\mathbb{I}$ asserts that for any type family P over $\mathbb{I}$, if we have

$$\begin{aligned}
u &: P(\mathsf{source})\\
v &: P(\mathsf{target})\\
p &: \mathsf{tr}_P(\mathsf{path}, u) = v,
\end{aligned}$$

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

then there is a section $f : \prod_{(x:\mathbb{I})} P(x)$ equipped with identifications

$$\alpha : f(\mathsf{source}) = u$$
$$\beta : f(\mathsf{target}) = v$$

and an identification γ witnessing that the square

$$\begin{array}{ccc} \mathsf{tr}_P(\mathsf{path}, f(\mathsf{source})) & \xrightarrow{\mathsf{ap}_{\mathsf{tr}_P(\mathsf{path})}(\alpha)} & \mathsf{tr}_P(\mathsf{path}, u) \\ {\scriptstyle \mathsf{apd}_f(\mathsf{path})}\Big\| & & \Big\|{\scriptstyle p} \\ f(\mathsf{target}) & \xrightarrow[\beta]{} & v \end{array}$$

commutes. Note that the constructors of $\mathbb{I}$ induce a map

$$\varepsilon : \left(\prod_{(x:\mathbb{I})} P(x)\right) \to \left(\sum_{(u:P(\mathsf{source}))} \sum_{(v:P(\mathsf{target}))} \mathsf{tr}_P(\mathsf{path}, u) = v\right).$$

given by $f \mapsto (f(\mathsf{source}), f(\mathsf{target}), \mathsf{apd}_f(\mathsf{path}))$.

(a) Characterize the identity types of the codomain of the map ε in the following way: for any (u, v, q) and (u', v', q'), construct an equivalence

$$((u, v, q) = (u', v', q')) \simeq \sum_{(\alpha:u=u')} \sum_{(\beta:v=v')} q \cdot \beta = \mathsf{ap}_{\mathsf{tr}_P(\mathsf{path})}(\alpha) \cdot q'.$$

(b) Prove the dependent universal property of $\mathbb{I}$, i.e., show that the map ε is an equivalence.

(c) Show that $\mathbb{I}$ is contractible.

15 Image factorizations

The image of a map $f : A \to X$ can be thought of as the least subtype of X that contains all the values of f. More precisely, the image of f is an embedding $i : \mathrm{im}(f) \hookrightarrow X$ that fits in a commuting triangle

$$\begin{array}{ccc} A & \xrightarrow{q} & \mathrm{im}(f) \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle i} & \\ & X & \end{array}$$

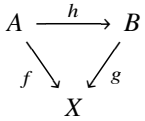
and satisfies the *universal property* of the image of f, which states that if a subtype $B \hookrightarrow X$ contains all the values of f, then it contains the image of f.

15.1 The image of a map

The universal property of the image

Recall from Exercise 13.15 that we made the following definition:

Definition 15.1.1 Let $f : A \to X$ and $g : B \to X$ be maps. A **morphism from** f **to** g **over** X consists of a map $h : A \to B$ equipped with a homotopy $H : f \sim g \circ h$ witnessing that the triangle



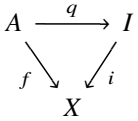
commutes. Thus, we define the type

$$\mathsf{hom}_X(f, g) :\equiv \textstyle\sum_{(h:A\to B)} f \sim g \circ h.$$

Composition of morphisms over X is defined by

$$(k, K) \circ (h, H) :\equiv (k \circ h, H \cdot (K \cdot h)).$$

Definition 15.1.2 Consider a commuting triangle



with $H : f \sim i \circ q$, where i is an embedding. We say that i satisfies the **universal property of the image of** f if the precomposition function

$$- \circ (q, H) : \mathsf{hom}_X(i, m) \to \mathsf{hom}_X(f, m)$$

is an equivalence for every embedding $m : B \hookrightarrow X$.

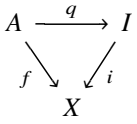
Lemma 15.1.3 *For any $f : A \to X$ and any embedding $m : B \to X$, the type $\mathsf{hom}_X(f, m)$ is a proposition.*

Proof Recall from Exercise 13.15 that the type $\mathsf{hom}_X(f, m)$ is equivalent to the type

$$\textstyle\prod_{(a:A)} \mathsf{fib}_m(f(a)).$$

Furthermore, recall from Theorem 12.2.3 that a map is an embedding if and only if its fibers are propositions. Thus we see that the type $\prod_{(a:A)} \mathsf{fib}_m(f(a))$ is a product of propositions, hence it is a proposition by Theorem 13.1.5. □

Proposition 15.1.4 *Consider a commuting triangle*



with $H : f \sim i \circ q$, where i is an embedding. Then the following are equivalent:

(i) *The embedding i satisfies the universal property of the image inclusion of f.*
(ii) *For every embedding $m : B \to X$ there is a map*

$$\mathsf{hom}_X(f, m) \to \mathsf{hom}_X(i, m).$$

Proof Since $\mathsf{hom}_X(f, m)$ is a proposition for every every embedding $m : B \to X$, the claim follows immediately by the observation made in Remark 14.1.2. □

The existence of the image

The image of a map $f : A \to X$ can be defined using the propositional truncation.

Definition 15.1.5 For any map $f : A \to X$ we define the **image** of f to be the type

$$\mathrm{im}(f) :\equiv \textstyle\sum_{(x:X)} \|\mathsf{fib}_f(x)\|.$$

Furthermore, we define

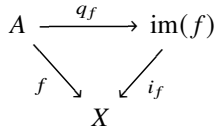
(i) the **image inclusion**

$$i_f : \mathrm{im}(f) \to X$$

to be the projection pr_1,
(ii) the map

$$q_f : A \to \mathrm{im}(f)$$

to be the map given by $q_f(x) :\equiv (f(x), \eta(x, \mathsf{refl}_{f(x)}))$, and
(iii) the homotopy $I_f : f \sim i_f \circ q_f$ witnessing that the triangle



commutes, to be given by $I_f(x) :\equiv \mathsf{refl}_{f(x)}$.

Proposition 15.1.6 *The image inclusion $i_f : \mathrm{im}(f) \to X$ of any map $f : A \to X$ is an embedding.*

Proof The claim follows directly by Corollary 12.2.4, because the type $\|\mathsf{fib}_f(x)\|$ is a proposition for each $x : X$. □

Theorem 15.1.7 *The image inclusion $i_f : \mathrm{im}(f) \to X$ of any map $f : A \to X$ satisfies the universal property of the image inclusion of f.*

Proof Consider an embedding $m : B \hookrightarrow X$. Note that we have a commuting square

$$\begin{array}{ccc} \mathsf{hom}_X(i_f, m) & \longrightarrow & \mathsf{hom}_X(f, m) \\ \downarrow & & \downarrow \\ \left(\prod_{(x:X)} \mathsf{fib}_{i_f}(x) \to \mathsf{fib}_m(x)\right) & \xrightarrow[h \mapsto \lambda x.\, h_x \circ \varphi_x]{} & \left(\prod_{(x:X)} \mathsf{fib}_f(x) \to \mathsf{fib}_m(x)\right) \end{array}$$

in which all four types are propositions, and the vertical maps are equivalences. Therefore it suffices to construct a map

$$\left(\prod_{(x:X)} \mathsf{fib}_f(x) \to \mathsf{fib}_m(x)\right) \to \left(\prod_{(x:X)} \mathsf{fib}_{i_f}(x) \to \mathsf{fib}_m(x)\right)$$

The fiber $\mathsf{fib}_{i_f}(x)$ is equivalent to the propositional truncation $\|\mathsf{fib}_f(x)\|$ and the type $\mathsf{fib}_m(x)$ is a proposition by the assumption that m is an embedding. Therefore we obtain the desired map by the universal property of the propositional truncation. □

The uniqueness of the image

We will now show that the universal property of the image implies that the image is determined uniquely up to equivalence.

Theorem 15.1.8 *Let f be a map, and consider two commuting triangles*

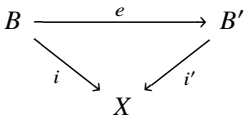
$$\begin{array}{ccc} A & \xrightarrow{q} & B \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle i} & \\ & X & \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{q'} & B' \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle i'} & \\ & X & \end{array}$$

with $I : f \sim i \circ q$ and $I' : f \sim i' \circ q'$, in which i and i' are assumed to be embeddings. Then, if any two of the following three properties hold, so does the third:

(i) *The embedding i satisfies the universal property of the image inclusion of f.*

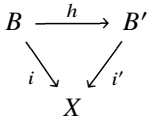
(ii) *The embedding i' satisfies the universal property of the image inclusion of f.*

(iii) *The type of equivalences $e : B \simeq B'$ equipped with a homotopy witnessing that the triangle*



commutes is contractible.

Proof First, we show that if (i) and (ii) hold, then (iii) holds. Note that the type $\mathsf{hom}_X(i, i')$ is a proposition, since i' is assumed to be an embedding. Therefore it suffices to show that the unique map $h : B \to B'$ such that the triangle



commutes, is an equivalence. To see this, note that by Exercise 13.15 it suffices to show that the action on fibers

$$\mathsf{fib}_i(x) \to \mathsf{fib}_{i'}(x)$$

is an equivalence for each $x : X$. This follows from the universal property of i', since we similarly obtain a family of maps

$$\mathsf{fib}_{i'}(x) \to \mathsf{fib}_i(x)$$

indexed by $x : X$, and the types $\mathsf{fib}_i(x)$ and $\mathsf{fib}_{i'}(x)$ are propositions by the assumptions that i and i' are embeddings.

Now we will show that (iii) implies that (i) holds if and only if (ii) holds. We will assume a morphism $(e, H) : \mathsf{hom}_X(i, i')$ such that the map e is an equivalence. Furthermore, consider an embedding $m : C \to X$. Then the fact that (i) holds if and only if (ii) holds follows from the equivalence

$$\big(\mathsf{hom}_X(f, m) \to \mathsf{hom}_X(i, m)\big) \simeq \big(\mathsf{hom}_X(f, m) \to \mathsf{hom}_X(i', m)\big). \qquad \square$$

15.2 Surjective maps

A map $f : A \to B$ is surjective if for every $b : B$ there is an *unspecified* element $a : A$ that maps to b. We define this property using the propositional truncation.

Definition 15.2.1 A map $f : A \to B$ is said to be **surjective** if there is an element of type

$$\mathsf{is\text{-}surj}(f) :\equiv \prod_{(y:B)} \|\mathsf{fib}_f(b)\|.$$

Example 15.2.2 Any equivalence is a surjective map, since its fibers are contractible. More generally, any map that has a section is surjective. Those are sometimes called **split epimorphisms**. Note that having a section is stronger than surjectivity, since in general we don't have a function $\|\mathsf{fib}_f(b)\| \to \mathsf{fib}_f(b)$.

In Exercise 14.4 we showed the dependent universal property of the propositional truncation: a map $f : A \to B$ into a proposition B satisfies the universal property of the propositional truncation if and only if for every family of propositions P over B, the precomposition map

$$- \circ f : \left(\prod_{(b:B)} P(b)\right) \to \left(\prod_{(a:A)} P(f(a))\right)$$

is an equivalence. In the following proposition we show that, if we omit the condition that B is a proposition, then f satisfies this dependent universal property if and only if f is surjective.

Proposition 15.2.3 *Consider a map $f : A \to B$. Then the following are equivalent:*

(i) *The map $f : A \to B$ is surjective.*
(ii) *For any family P of propositions over B, the precomposition map*

$$- \circ f : \left(\prod_{(y:B)} P(y)\right) \to \left(\prod_{(x:A)} P(f(x))\right)$$

is an equivalence.

Proof Suppose first that f is surjective, and consider the commuting square

$$\begin{array}{ccc} \left(\prod_{(y:B)} P(y)\right) & \xrightarrow{\quad - \circ f \quad} & \left(\prod_{(x:A)} P(f(x))\right) \\ {\scriptstyle h \mapsto \lambda y.\, \mathsf{const}_{h(y)}}\downarrow & & \uparrow{\scriptstyle h \mapsto \lambda x.\, h(f(x),(x,\mathsf{refl}))} \\ \left(\prod_{(y:B)} \|\mathsf{fib}_f(y)\| \to P(y)\right) & \xrightarrow[h \mapsto h(-) \circ \eta]{} & \left(\prod_{(y:B)} \mathsf{fib}_f(y) \to P(y)\right) \end{array}$$

In this square, the bottom map is an equivalence by Exercise 13.12 (a) and by the universal property of the propositional truncation of $\mathsf{fib}_f(y)$. The map on the right is an equivalence by Exercise 13.15 (a). Furthermore, the map on the left is an equivalence by Exercise 13.12 (a) and Exercise 13.8, because the type $\|\mathsf{fib}_f(y)\|$ is contractible by the assumption that f is surjective. Therefore it follows that the top map is an equivalence, which completes the proof that (i) implies (ii).

For the converse, it follows immediately from the assumption (ii) that

$$- \circ f : \left(\prod_{(y:B)} \|\mathsf{fib}_f(y)\|\right) \to \left(\prod_{(x:A)} \|\mathsf{fib}_f(f(x))\|\right)$$

is an equivalence. Hence it suffices to construct an element of type $\|\mathsf{fib}_f(f(x))\|$ for each $x : A$. This is easy, because we have

$$\eta(x, \mathsf{refl}_{f(x)}) : \|\mathsf{fib}_f(f(x))\|. \qquad \square$$

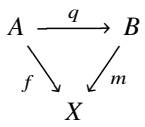
As a corollary we obtain that any surjective map into a proposition satisfies the universal property of the propositional truncation.

Corollary 15.2.4 *For any map $f : A \to P$ into a proposition P, the following are equivalent:*

(i) *The map f satisfies the universal property of the propositional truncation of A.*
(ii) *The map f is surjective.*

Using the characterization of surjective maps of Proposition 15.2.3, we can also give a new characterization of the image of a map.

Theorem 15.2.5 *Consider a commuting triangle*



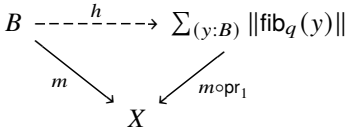
in which m is an embedding. Then the following are equivalent:

(i) *The embedding m satisfies the universal property of the image inclusion of f.*
(ii) *The map q is surjective.*

Proof First assume that m satisfies the universal property of the image inclusion of f, and consider the composite function

$$\left(\sum_{(y:B)} \|\mathsf{fib}_q(y)\|\right) \xrightarrow{\mathsf{pr}_1} B \xrightarrow{m} X.$$

Note that $m \circ \mathsf{pr}_1$ is a composition of embeddings, so it is an embedding. By the universal property of m there is a unique map h for which the triangle



commutes. Now note that $\mathsf{pr}_1 \circ h$ is a map such that $m \circ (\mathsf{pr}_1 \circ h) \sim m$. The identity function is another map for which we have $m \circ \mathsf{id} \sim m$, so it follows by uniqueness that $\mathsf{pr}_1 \circ h \sim \mathsf{id}$. In other words, the map h is a section of the projection map. Therefore we obtain by Corollary 13.2.3 a dependent function

$$\prod_{(b:B)} \|\mathsf{fib}_q(b)\|,$$

showing that q is surjective.

For the converse, suppose that q is surjective. To prove that m satisfies the universal property of the image factorization of f, it suffices to construct a map

$$\mathsf{hom}_X(f, m') \to \mathsf{hom}_X(m, m'),$$

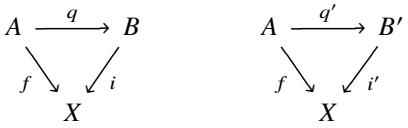
for any embedding $m' : B' \to X$. To see that there is such an equivalence, we make the following calculation

$$\begin{aligned}
\mathsf{hom}_X(m, m') &\simeq \prod_{(b:B)} \mathsf{fib}_{m'}(m(b)) && \text{(By Exercise 13.15)} \\
&\simeq \prod_{(a:A)} \mathsf{fib}_{m'}(m(q(a))) && \text{(By Proposition 15.2.3)} \\
&\simeq \prod_{(a:A)} \mathsf{fib}_{m'}(f(a)) && \text{(By } f \sim m \circ q\text{)} \\
&\simeq \mathsf{hom}_X(f, m'). && \text{(By Exercise 13.15)}
\end{aligned}$$

□

Corollary 15.2.6 *Every map factors uniquely as a surjective map followed by an embedding.*

Proof Consider a map $f : A \to X$, and two factorizations



of f where m and m' are embeddings, and q and q' are surjective. Then both m and m' satisfy the universal property of the image factorization of f by Theorem 15.2.5. Now it follows by Theorem 15.1.8 that the type of $(e, H) : \mathsf{hom}_X(i, i')$ in which e is an equivalence, equipped with an identification

$$(e, H) \circ (q, I) = (q', I')$$

in $\mathsf{hom}_X(f, i')$, is contractible. □

15.3 Cantor's diagonal argument

Now that we have introduced surjective maps, we are in position to give Cantor's famous diagonal argument, which he used to show that there are infinite sets

of different cardinality. The diagonal argument gives a proof that there is no surjective map from X to its power set $\mathcal{P}(X)$. The power set of a type X is of course defined with respect to a universe $\mathcal{U}$, as the type of families of propositions in $\mathcal{U}$ indexed by X.

Definition 15.3.1 Consider a type X, and a universe $\mathcal{U}$. We define the $\mathcal{U}$-power set of X to be

$$\mathcal{P}_{\mathcal{U}}(X) :\equiv X \to \mathsf{Prop}_{\mathcal{U}}.$$

Theorem 15.3.2 *For any type X and any universe $\mathcal{U}$, there is no surjective function*

$$f : X \to \mathcal{P}_{\mathcal{U}}(X)$$

Proof Consider a function $f : X \to (X \to \mathsf{Prop}_{\mathcal{U}})$, and suppose that f is surjective. Following Cantor's diagonalization argument, we define the subset $P : X \to \mathsf{Prop}_{\mathcal{U}}$ by

$$P(x) :\equiv \neg(f(x, x)).$$

Our goal is to reach a contradiction and f is assumed to be surjective. Therefore, it suffices to show that

$$\left\| \textstyle\sum_{(x:X)} f(x) = P \right\| \to \emptyset.$$

The empty type is a proposition, so by the universal property of the propositional truncation it is equivalent to show that

$$\left(\textstyle\sum_{(x:X)} f(x) = P \right) \to \emptyset.$$

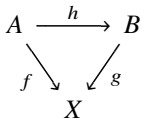
Consider an element $x : X$ equipped with an identification $f(x) = P$. Our goal is to construct an element of the empty type, i.e, to reach a contradiction. By the identification $f(x) = P$ it follows that

$$f(x, y) \leftrightarrow P(y)$$

for all $y : X$. In particular, it follows that $f(x, x) \leftrightarrow P(x)$. However, since $P(x)$ is defined as $\neg(f(x, x))$, we obtain that $f(x, x) \leftrightarrow \neg(f(x, x))$. By Exercise 4.3 (a) this gives us the desired contradiction. □

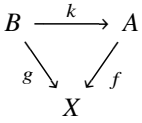
Exercises

15.1 Consider a commuting triangle



where g is an embedding.

(a) Show that if there is a morphism



over X, then g satisfies the universal property of the image of f.

(b) Show that if f is an embedding, then g satisfies the universal property of f if and only if h is an equivalence.

15.2 (a) Show that for any proposition P, the constant map

$$\mathsf{const}_\star : P \to \mathbf{1}$$

is an embedding. Use this fact to construct an equivalence

$$\left(\textstyle\sum_{(A:\mathcal{U})} A \hookrightarrow \mathbf{1}\right) \simeq \mathsf{Prop}_{\mathcal{U}}.$$

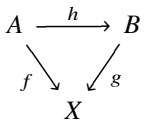
(b) Consider a map $f : A \to P$ into a proposition P. Show that the following are equivalent:

(i) The map f is a propositional truncation of A.
(ii) The constant map $P \to \mathbf{1}$ satisfies the universal property of the image of the constant map $A \to \mathbf{1}$.

15.3 Consider a map $f : A \to B$. Show that the following are equivalent:

(i) f is an equivalence.
(ii) f is both surjective and an embedding.

15.4 Consider a commuting triangle



with $H : f \sim g \circ h$, and assume that h is surjective.

(a) Show that the following are equivalent:

(i) The map f is surjective.
(ii) The map g is surjective.

(b) As a converse to Exercise 12.11, show that if f and h are k-truncated, then g is also k-truncated.

15.5 Prove **Lawvere's fixed point theorem**: For any two types A and B, if there is a surjective map $f : A \to B^A$, then for any $h : B \to B$ there exists an $x : B$ such that $h(x) = x$, i.e., show that

$$\left(\exists_{(f:A\to(A\to B))} \mathsf{is\text{-}surj}(f)\right) \to \left(\forall_{(h:B\to B)} \exists_{(b:B)} h(b) = b\right).$$

16 Finite types

16.1 Counting in type theory

When someone counts the elements of a finite set A, they go through the elements of A one by one, at each stage keeping track of how many elements have been counted so far. This process results in the number $|A|$ of elements of the set A, and moreover it gives a bijection from the standard finite set with $|A|$ elements. In other words, to count the elements of A is to give an equivalence from one of the standard finite sets to the set A. We turn this into a definition.

Definition 16.1.1 For each type A, we define the type

$$\mathsf{count}(A) :\equiv \textstyle\sum_{(k:\mathbb{N})} (\mathsf{Fin}_k \simeq A).$$

The elements of $\mathsf{count}(A)$ are called **countings** of A. When we have $(k, e) : \mathsf{count}(A)$, we also say that A **has** k **elements**.

Note that the type $\mathsf{count}(A)$ is often not a proposition. For instance, different equivalences of type $\mathsf{Fin}_k \simeq \mathsf{Fin}_k$ induce different elements of type $\mathsf{count}(\mathsf{Fin}_k)$.

Example 16.1.2 It follows immediately from the definition of countings that every standard finite type can be counted in a canonical way: For any $k : \mathbb{N}$ we have $(k, \mathsf{id}) : \mathsf{count}(\mathsf{Fin}_k)$. It also follows immediately from the definition of countings that types equipped with countings are closed under equivalences.

Example 16.1.3 Suppose A comes equipped with a counting $(k, e) : \mathsf{count}(A)$. Then $k = 0$ if and only if A is empty. Indeed, the inverse of e is a map $e^{-1} : A \to \emptyset$. Conversely, if we have $f : \mathsf{is\text{-}empty}(A)$, then the map $f : A \to \emptyset$ is automatically an equivalence. This shows that $\mathsf{Fin}_k \simeq \emptyset$, and a short argument by induction on k yields that $k = 0$.

Example 16.1.4 A type A has one element if and only if it is contractible. Indeed, the type Fin_1 is contractible, so it follows from the 3-for-2 property of contractible types (Exercise 10.2) that there is an equivalence $\mathsf{Fin}_1 \simeq A$ if and only if A is contractible.

Example 16.1.5 A proposition P comes equipped with a counting if and only if it is decidable. To see this, note that for any type X, if we have $(k, e) : \mathsf{count}(X)$, then it follows that X is decidable. This is shown by induction on k. In the case where $k = 0$, it follows that X is empty, and hence that X is decidable. In the case where k is a successor, the bijection $e : \mathsf{Fin}_k \simeq X$ gives us the element $e(\star) : X$. Again we conclude that X is decidable.

Conversely, if P is decidable, then we can construct a counting of P by case analysis on $d : P + \neg P$. If P holds, then it is contractible and hence equivalent to Fin_1. If $\neg P$ holds, then P is equivalent to Fin_0.

Remark 16.1.6 We also note that any type A equipped with a counting $e : \mathsf{Fin}_k \simeq A$ has decidable equality. This follows from Proposition 8.1.8, where we showed that Fin_k has decidable equality, for any $k : \mathbb{N}$.

Theorem 16.1.7 *We make the following claims about countings:*

(i) *Consider two types A and B. The following are equivalent:*
   (a) *Both A and B come equipped with a counting.*
   (b) *The coproduct $A + B$ comes equipped with a counting.*

(ii) *Consider a type family B indexed by a type A. Consider the following three conditions:*
   (a) *The type A comes equipped with a counting.*
   (b) *The type $B(x)$ comes equipped with a counting, for each $x : A$.*
   (c) *The type $\sum_{(x:A)} B(x)$ comes equipped with a counting.*

   If (a) holds, then (b) holds if and only if (c) holds. Furthermore, if both (b) and (c) hold and if B comes equipped with a section $f : \prod_{(x:A)} B(x)$, then (a) holds.

   Consequently, if P is a subtype of a type A equipped with a counting, then we have

$$\mathsf{count}\left(\sum_{(x:A)} P(x)\right) \leftrightarrow \prod_{(x:A)} \mathsf{is\text{-}decidable}(P(x)).$$

Proof We will first prove the forward direction of (i). Then we will prove both claims in (ii), and we will prove the reverse direction of claim (i) last.

For the forward direction of claim (i), suppose we have equivalences e :

$\mathsf{Fin}_k \simeq A$ and $f : \mathsf{Fin}_l \simeq B$. The equivalences e and f induce via Exercises 9.6 and 9.8 a composite equivalence

$$A + B \xrightarrow{\simeq} \mathsf{Fin}_k + \mathsf{Fin}_l \xrightarrow{\simeq} \mathsf{Fin}_{k+l},$$

from which we obtain an element of type $\mathsf{count}(A + B)$.

Next, we will prove the forward direction in the first claim of (ii), i.e., we will prove that if A comes equipped with an equivalence $e : \mathsf{Fin}_k \simeq A$, and if B is a family of types over A equipped with

$$f : \textstyle\prod_{(x:A)} \mathsf{count}(B(x)),$$

then the total space $\sum_{(x:A)} B(x)$ also has a counting. The proof is by induction on k. Note that in the base case, where $k = 0$, the type $\sum_{(x:A)} B(x)$ is empty, so it has a counting. For the inductive step, note Σ distributes from the right over coproducts. This gives an equivalence

$$\begin{aligned}\textstyle\sum_{(x:A)} B(x) &\simeq \textstyle\sum_{(x:\mathsf{Fin}_{k+1})} B(e(x)) \\ &\simeq \left(\textstyle\sum_{(x:\mathsf{Fin}_k)} B(e(\mathsf{inl}(x)))\right) + B(e(\mathsf{inr}(\star))).\end{aligned}$$

The type $\sum_{(x:\mathsf{Fin}_k)} B(e(\mathsf{inl}(x)))$ has a counting by the inductive hypothesis, and the type $B(e(\mathsf{inr}(\star)))$ has a counting by assumption. Therefore, it follows that the total space $\sum_{(x:A)} B(x)$ has a counting.

Now we will prove the converse direction of the first claim in (ii). Suppose that A comes equipped with $e : \mathsf{Fin}_k \simeq A$, and that $\sum_{(x:A)} B(x)$ comes equipped with $f : \mathsf{Fin}_l \simeq \sum_{(x:A)} B(x)$. By Example 16.1.5 it suffices to show that, for $a : A$, the type $B(a)$ is a decidable subtype of $\sum_{(x:A)} B(x)$. Consider the map

$$i : B(a) \to \textstyle\sum_{(x:A)} B(x)$$

given by $b \mapsto (a, b)$. For $(x, y) : \sum_{(x:A)} B(x)$, we have the equivalences

$$\begin{aligned}\mathsf{fib}_i(x, y) &\simeq \textstyle\sum_{(b:B(a))} (a, b) = (x, y) \\ &\simeq \textstyle\sum_{(b:B(a))} \sum_{(p:a=x)} \mathsf{tr}_B(p, b) = y \\ &\simeq \textstyle\sum_{(p:a=x)} \mathsf{fib}_{\mathsf{tr}_B(p)}(y) \\ &\simeq a = x.\end{aligned}$$

Here we used that $\mathsf{tr}_B(p)$ is an equivalence, and therefore has contractible fibers. Now note that the type $a = x$ is a decidable proposition by Remark 16.1.6.

Next, we will prove the second claim in (ii). Suppose that B is a family over A that comes equipped with a section $b : \prod_{(x:A)} B(x)$, and suppose that each $B(x)$ has a counting, and that the total space $\sum_{(x:A)} B(x)$ has a counting. Then

we have a map

$$g : A \to \sum_{(x:A)} B(x)$$

given by $a \mapsto (a, b(a))$. The fibers of g can be computed by the following equivalences:

$$\begin{aligned}\mathsf{fib}_g(x, y) &\simeq \sum_{(a:A)} (a, b(a)) = (x, y) \\ &\simeq \sum_{(a:A)} \sum_{(p:a=x)} \mathsf{tr}_B(p, b(a)) = y \\ &\simeq \mathsf{tr}_B(p, b(x)) = y.\end{aligned}$$

Note that the type $\mathsf{tr}_B(p, b(x)) = y$ is a decidable proposition by Remark 16.1.6. Now it follows by the forward direction of the first claim in (ii) that A has a counting.

It remains to prove the converse direction of (i). Note that the forward direction of the first claim in (ii) implies that countings on a type X induce countings on any decidable subtype of X. Note that both A and B are decidable subtypes of the coproduct $A + B$. Any counting of $A + B$ therefore induces countings of A and of B. □

Corollary 16.1.8 *Consider two types A and B. We make two claims:*

(i) *If both A and B come equipped with a counting, then the product $A \times B$ has a counting.*
(ii) *If the product $A \times B$ comes equipped with a counting, then we have two functions*

$$B \to \mathsf{count}(A)$$
$$A \to \mathsf{count}(B).$$

Proof The first claim follows from (ii) (ii)(a) in Theorem 16.1.7, and the second claim follows from (ii) (ii)(b) in Theorem 16.1.7. □

16.2 Double counting in type theory

In combinatorics, counting arguments often proceed by showing that two finite sets are isomorphic—or, in the language of type theory, by showing that two finite types are equivalent. The idea here is, of course, that when we count the elements of a type twice correctly, then both countings must result in the same number. However, this is something that we must prove before we can use it. In other words, we must show that

$$(\mathsf{Fin}_k \simeq \mathsf{Fin}_l) \to (k = l)$$

for any two natural numbers k and l. We will prove this claim as a consequence of the following general fact.

Proposition 16.2.1 *For any two types X and Y, there is a map*

$$(X + \mathbf{1} \simeq Y + \mathbf{1}) \to (X \simeq Y).$$

Proof We prove the claim in four steps. We will write i for $\mathsf{inl} : X \to X + \mathbf{1}$ and also for $\mathsf{inl} : Y \to Y + \mathbf{1}$, and we will write $\star$ for $\mathsf{inr}(\star) : X + \mathbf{1}$ and also for $\mathsf{inr}(\star) : Y + \mathbf{1}$.

(i) We first show that for any equivalence $e : X + \mathbf{1} \simeq Y + \mathbf{1}$ and any $x : X$ equipped with an identification $p : e(i(x)) = \star$, that there is an element

$$\mathsf{star\text{-}value}(e, x, p) : Y$$

equipped with an identification

$$\alpha : i(\mathsf{star\text{-}value}(e, x, p)) = e(\star).$$

To see this, note that the map e is injective. The elements $i(x)$ and $\star$ are distinct, so it follows that the elements $e(i(x))$ and $e(\star)$ are distinct. In particular, we have $e(\star) \neq \star$. Therefore it follows that there is an element $y : Y$ equipped with an identification $i(y) = e(\star)$.

(ii) Next, we construct for every equivalence $e : X + \mathbf{1} \simeq Y + \mathbf{1}$ a map $f : X \to Y$ equipped with identifications

$$\beta : \textstyle\prod_{(y:Y)} (e(i(x)) = i(y)) \to (f(x) = y)$$
$$\gamma : \textstyle\prod_{(p:e(i(x))=\star)} f(x) = \mathsf{star\text{-}value}(e, x, p).$$

In order to construct the map $f : X \to Y$, we first construct a dependent function

$$f' : \textstyle\prod_{(x:X)} \prod_{(u:Y+\mathbf{1})} ((e(i(x)) = u) \to Y).$$

This function is defined by pattern matching on u, by

$$f'(x, i(y), p) :\equiv y$$
$$f'(x, \star, p) :\equiv \mathsf{star\text{-}value}(e, x, p)$$

Then we define $f(x) :\equiv f'(x, e(i(x)), \mathsf{refl})$. By the definition of f' it then follows that we have an identification

$$\begin{aligned} f(x) &\doteq f'(x, e(i(x)), \mathsf{refl}) \\ &= f'(x, i(y), p) \\ &\doteq y \end{aligned}$$

for any $y : Y$ and $p : e(i(x)) = i(y)$, and that we have an identification

$$\begin{aligned} f(x) &\doteq f'(x, e(i(x)), \mathsf{refl}) \\ &= f'(x, \star, p) \\ &\doteq \mathsf{star\text{-}value}(e, x, p) \end{aligned}$$

for any $p : e(i(x)) = \star$.

(iii) The inverse function $g : Y \to X$ is constructed in the same way as the function $f : X \to Y$, using the equivalence $e^{-1} : Y + \mathbf{1} \simeq X + \mathbf{1}$. This function comes equipped with

$$\begin{aligned} \delta &: \textstyle\prod_{(x:X)} (e^{-1}(i(y)) = i(x)) \to (g(y) = x) \\ \varepsilon &: \textstyle\prod_{(p:e^{-1}(i(y))=\star)} g(y) = \mathsf{star\text{-}value}(e^{-1}, y, p). \end{aligned}$$

(iv) It remains to show that f and g are inverse to each other. The proof that g is a retraction of f is similar to the proof that g is a section of f, so we will only prove the latter. In other words, we will construct an identification

$$f(g(y)) = y$$

for any $y : Y$. The proof is by case analysis on $(e^{-1}(i(y)) = \star) + (e^{-1}(i(y)) \neq \star)$. In the case where $p : e^{-1}(i(y)) = \star$, we have the identification

$$\varepsilon(p) : g(y) = \mathsf{star\text{-}value}(e^{-1}, y, p).$$

Furthermore, we have the identification

$$\gamma(q) : f(g(y)) = \mathsf{star\text{-}value}(e, g(y), q),$$

where $q : e(i(g(y))) = \star$ is the composite of the identifications

$$\begin{aligned} e(i(g(y))) &= e(i(\mathsf{star\text{-}value}(e^{-1}, y, p))) \\ &= e(e^{-1}(\star)) \\ &= \star. \end{aligned}$$

Using the identification $\gamma(q)$, we obtain

$$\begin{aligned} i(f(g(y))) &= i(\mathsf{star\text{-}value}(e, g(y), q)) \\ &= e(\star) \\ &= e(e^{-1}(i(y))) \\ &= i(y). \end{aligned}$$

Since $i : Y \to Y + \mathbf{1}$ is injective, it follows that $f(g(y)) = y$. □

Theorem 16.2.2 *For any two natural numbers k and l, there is a map*

$$(\mathsf{Fin}_k \simeq \mathsf{Fin}_l) \to (k = l).$$

Proof The proof is by induction on k and l. In the base case, where both k and l are zero, the claim is obvious. If k is zero and l is a successor, then we have $0 : \mathsf{Fin}_l$. Any equivalence $e : \mathsf{Fin}_k \simeq \mathsf{Fin}_l$ now gives us the element

$$e^{-1}(0) : \emptyset,$$

which is of course absurd. Similarly, if k is a successor and l is zero, we obtain $e(0) : \emptyset$, which is again absurd. If both k and l are a successor, then we have by Proposition 16.2.1 the composite

$$(\mathsf{Fin}_{k+1} \simeq \mathsf{Fin}_{l+1}) \longrightarrow (\mathsf{Fin}_k \simeq \mathsf{Fin}_l) \longrightarrow (k = l) \longrightarrow (k + 1 = l + 1). \qquad \square$$

16.3 Finite types

The type of all finite types is the subtype of the base universe $\mathcal{U}_0$ consisting of all types X for which there exists an unspecified equivalence $\mathsf{Fin}_k \simeq X$ for some $k : \mathbb{N}$.

Definition 16.3.1 A type X is said to be **finite** if it comes equipped with an element of type

$$\mathsf{is\text{-}finite}(X) :\equiv \left\| \textstyle\sum_{(k:\mathbb{N})} \mathsf{Fin}_k \simeq X \right\|$$

The type $\mathbb{F}$ of all finite types is defined to be

$$\mathbb{F} :\equiv \textstyle\sum_{(X:\mathcal{U}_0)} \mathsf{is\text{-}finite}(X).$$

In other words, the type $\mathbb{F}$ of finite types is the image of the map $\mathsf{Fin} : \mathbb{N} \to \mathcal{U}_0$. We also define the type $\mathbb{F}_k$ of **k-element types** by

$$\mathbb{F}_k :\equiv \textstyle\sum_{(X:\mathcal{U}_0)} \|\mathsf{Fin}_k \simeq X\|.$$

Remark 16.3.2 It follows directly from the definition of finiteness that any type X equipped with a counting is finite. In particular, any Fin_k is finite. Furthermore, it follows that if X is equivalent to a finite type Y, then X is also finite. Indeed, we can use the functoriality of the propositional truncation to obtain a function

$$\left\| \textstyle\sum_{(k:\mathbb{N})} \mathsf{Fin}_k \simeq Y \right\| \to \left\| \textstyle\sum_{(k:\mathbb{N})} \mathsf{Fin}_k \simeq X \right\|$$

from a map $\left(\sum_{(k:\mathbb{N})} \mathsf{Fin}_k \simeq Y\right) \to \left(\sum_{(k:\mathbb{N})} \mathsf{Fin}_k \simeq X\right.$. Given an equivalence $e : X \simeq Y$, such a map is given as the map induced on total spaces from the family of maps $f \mapsto e^{-1} \circ f$.

Similarly, it follows that any finite type has decidable equality, and that every finite type is a set.

In the following proposition we will show that each finite type can be assigned a unique cardinality.

Theorem 16.3.3 *For any type X, consider the type* $\mathsf{is\text{-}finite}'(X)$ *defined by*

$$\mathsf{is\text{-}finite}'(X) :\equiv \textstyle\sum_{(k:\mathbb{N})} \|\mathsf{Fin}_k \simeq X\|.$$

Then the type $\mathsf{is\text{-}finite}'(X)$ *is a proposition, and there is an equivalence*

$$\mathsf{is\text{-}finite}(X) \leftrightarrow \mathsf{is\text{-}finite}'(X).$$

If X is a finite type, then the unique number k such that $\|\mathsf{Fin}_k \simeq X\|$ is the ***cardinality*** *of X. We write $|X|$ for the cardinality of X.*

Proof We first prove the claim that the type $\mathsf{is\text{-}finite}'(X)$ is a proposition. In other words, we need to show that any two natural numbers k and k' for which there are respective elements of the types $\|\mathsf{Fin}_k \simeq X\|$ and $\|\mathsf{Fin}_{k'} \simeq X\|$, can be identified.

Since the type of natural numbers is a set, the type $k = k'$ is a proposition. Therefore, we may assume that we have equivalences $\mathsf{Fin}_k \simeq X$ and $\mathsf{Fin}_{k'} \simeq X$. Consequently, we have an equivalence $\mathsf{Fin}_k \simeq \mathsf{Fin}_{k'}$. Now it follows from Theorem 16.2.2 that $k = k'$.

The second claim is that the propositions $\mathsf{is\text{-}finite}(X)$ and $\mathsf{is\text{-}finite}'(X)$ are equivalent, which we will show by constructing functions back and forth. Since we have shown that the type $\mathsf{is\text{-}finite}'(X)$ is a proposition, we obtain a map $\mathsf{is\text{-}finite}(X) \to \mathsf{is\text{-}finite}'(X)$ via the universal property of the propositional truncation, from the map

$$\left(\textstyle\sum_{(k:\mathbb{N})} \mathsf{Fin}_k \simeq X\right) \to \textstyle\sum_{(k:\mathbb{N})} \|\mathsf{Fin}_k \simeq X\|$$

given by $(k, e) \mapsto (k, \eta(e))$.

To construct a map $\mathsf{is\text{-}finite}'(X) \to \mathsf{is\text{-}finite}(X)$, it suffices to construct a map

$$\|\mathsf{Fin}_{k'} \simeq X\| \to \left\|\textstyle\sum_{(k:\mathbb{N})} \mathsf{Fin}_k \simeq X\right\|$$

for each $k' : \mathbb{N}$. Again by the universal property of the propositional truncation, we obtain this map from the function

$$(\mathsf{Fin}_{k'} \simeq X) \to \left\|\textstyle\sum_{(k:\mathbb{N})} \mathsf{Fin}_k \simeq X\right\|$$

given by $e \mapsto \eta(k', e)$. □

Corollary 16.3.4 *There is an equivalence*

$$\mathbb{F} \simeq \textstyle\sum_{(k:\mathbb{N})} \mathbb{F}_k.$$

Proof This equivalence can be obtained by composing the equivalences

$$\begin{aligned}\textstyle\sum_{(X:\mathcal{U}_0)} \mathsf{is\text{-}finite}(X) &\simeq \textstyle\sum_{(X:\mathcal{U}_0)} \sum_{(k:\mathbb{N})} \|\mathsf{Fin}_k \simeq X\| \\ &\simeq \textstyle\sum_{(k:\mathbb{N})} \sum_{(X:\mathcal{U}_0)} \|\mathsf{Fin}_k \simeq X\|.\end{aligned}$$

□

We now aim to extend Theorem 16.1.7 to obtain some closure properties of finite types. Before we do so, we prove the finite principle of choice.

Proposition 16.3.5 *Consider a type family B over a finite type A. Then there is a map*

$$\left(\textstyle\prod_{(x:A)} \|B(x)\|\right) \to \left\|\textstyle\prod_{(x:A)} B(x)\right\|$$

Proof Note that the type $\left\|\prod_{(x:A)} B(x)\right\|$ is a proposition. Therefore we may assume that the type A comes equipped with a counting $e : \mathsf{Fin}_k \simeq A$. By this equivalence, it suffices to show that for every type family B over Fin_k, there is a map

$$\left(\textstyle\prod_{(x:\mathsf{Fin}_k)} \|B(x)\|\right) \to \left\|\textstyle\prod_{(x:\mathsf{Fin}_k)} B(x)\right\|.$$

We proceed by induction on k. In the base case, Fin_k is empty and therefore the type $\prod_{(x:Fink)} B(x)$ is contractible. The asserted function therefore exists.

For the inductive step, note that by the dependent universal property of coproducts (Exercise 13.9) we have the equivalences

$$\begin{aligned}\left(\textstyle\prod_{(x:\mathsf{Fin}_{k+1})} \|B(x)\|\right) &\simeq \left(\textstyle\prod_{(x:\mathsf{Fin}_k)} \|B(i(x))\|\right) \times \|B(\star)\| \\ \left\|\textstyle\prod_{(x:\mathsf{Fin}_k)} B(x)\right\| &\simeq \left\|\left(\textstyle\prod_{(x:\mathsf{Fin}_k)} B(i(x))\right) \times B(\star)\right\|.\end{aligned}$$

Recall from Exercise 14.3 that $\|X \times Y\| \simeq \|X\| \times \|Y\|$ for any two types X and Y. We use this fact, along with the inductive hypothesis, to finish the proof. □

Theorem 16.3.6

(i) *For any two types X and Y, the following are equivalent:*
   (a) *Both X and Y are finite.*
   (b) *The coproduct $X + Y$ is finite.*

(ii) *For any two types X and Y, we make two claims:*
   (a) *If both X and Y are finite, then the cartesian product $X \times Y$ is finite.*

(b) *If the type $X \times Y$ is finite, then we have two functions*

$$Y \to \mathsf{is\text{-}finite}(X)$$
$$X \to \mathsf{is\text{-}finite}(Y).$$

(iii) *Consider a type family B over A, and consider the following three conditions:*

(a) *The type A is finite.*
(b) *The type $B(x)$ is finite for each $x : A$.*
(c) *The type $\sum_{(x:A)} B(x)$ is finite.*

If (a) holds, then (b) is equivalent to (c). Moreover, if (b) and (c) hold, then (a) holds if and only if A is a set and the type $\sum_{(x:A)} \neg B(x)$ is finite. Furthermore, if (b) and (c) hold and B has a section, then (a) holds.

Proof To prove claim (i), first suppose that both X and Y are finite. Since the type $\mathsf{is\text{-}finite}(X + Y)$ is a proposition, we may assume that X and Y come equipped with countings. It follows from Theorem 16.1.7 that $X + Y$ has a counting, so it is finite. Conversely, suppose that the type $X + Y$ is finite. Since the types $\mathsf{is\text{-}finite}(X)$ and $\mathsf{is\text{-}finite}(Y)$ are both propositions, we may assume that the coproduct $X + Y$ comes equipped with a counting. Again it follows from Theorem 16.1.7 that the types X and Y have countings, so they are finite.

The proof of claim (ii) is similar to the proof of claim (i), hence we omit it.

It remains to prove claim (iii). First, suppose that the type A is finite, and that each $B(x)$ is finite. By Proposition 16.3.5 we have a map

$$\left(\prod_{(x:A)} \mathsf{is\text{-}finite}(B(x))\right) \to \left\|\prod_{(x:A)} \mathsf{count}(B(x))\right\|.$$

Since our goal is to construct an element of a proposition, we may therefore assume that each $B(x)$ comes equipped with a counting. We may also assume that A comes equipped with a counting. It follows from Theorem 16.1.7 that the type $\sum_{(x:A)} B(x)$ has a counting, so it is finite.

Next, assume that A is finite and that the type $\sum_{(x:A)} B(x)$ is finite, and let $a : A$. The type $\mathsf{is\text{-}finite}(B(a))$ is a proposition, so we may assume that the types A and $\sum_{(x:A)} B(x)$ come equipped with countings. It follows from Theorem 16.1.7 that $B(a)$ has a counting, so it is finite.

The final claim has two parts. First, assume that each $B(x)$ is finite, that the type $\sum_{(x:A)} B(x)$ is finite, and that the type family B has a section $f : \prod_{(x:A)} B(x)$. It follows that the map

$$A \to \sum_{(x:A)} B(x)$$

given by $x \mapsto (x, f(x))$ is a decidable embedding, because the fiber at (x, y) of

this map is equivalent to the identity type $f(x) = y$ in $B(x)$, which is a decidable proposition. It follows from the fact that (a) and (b) together imply (c) that A is finite.

For the remaining part of the final claim, assume that A is a set. Note that the assumption that each $B(x)$ is finite implies that each $B(x)$ is either inhabited or empty. It follows that we have an equivalence

$$A \simeq \left(\textstyle\sum_{(x:A)} \|B(x)\|\right) + \left(\textstyle\sum_{(x:A)} \neg B(x)\right).$$

We assume that the type $\sum_{(x:A)} \neg B(x)$ is finite. In order to show that A is finite, it therefore suffices to show that the type $\sum_{(x:A)} \|B(x)\|$ is finite. Without loss of generality, we assume that each $B(x)$ is inhabited. To finish the proof, it suffices to show that there is an element of type

$$\left\|\textstyle\prod_{(x:A)} B(x)\right\|$$

using the assumption that $\prod_{(x:A)} \|B(x)\|$. To construct such an element, we may assume a counting $e : \mathsf{Fin}_k \simeq \sum_{(x:A)} B(x)$. We claim that there is a function

$$\|B(a)\| \to B(a),$$

i.e., that the type $B(a)$ satisfies the principle of global choice of Remark 14.4.2 for each $a : A$. Recall from Example 14.4.1 that the decidable subtypes of Fin_k satisfy global choice. Therefore it also follows that the decidable subtypes of $\sum_{(x:A)} B(x)$ satisfy global choice. Thus, it suffices to show that $B(x)$ is a decidable subtype of $\sum_{(x:A)} B(x)$.

The assumption that A is a set implies by Exercise 12.13 that the fiber inclusion $i_a : B(a) \to \sum_{(x:A)} B(x)$ is an embedding for each $a : A$. Furthermore, we note that we have the following equivalence computing the fibers of i_a at (x, y):

$$\left(\textstyle\sum_{(z:B(a))} (a, z) = (x, y)\right) \simeq (a = x).$$

The type on the left hand side is decidable, so it follows that the type A has decidable equality. We conclude that each $B(a)$ is a decidable subtype of $\sum_{(x:A)} B(x)$. □

Theorem 16.3.7 *Consider a universe $\mathcal{U}$. The subuniverse*

$$\textstyle\sum_{(X:\mathcal{U})} \mathsf{is\text{-}finite}(X)$$

of finite types in $\mathcal{U}$ is the least subuniverse that contains $\emptyset$, $\mathbf{1}$, and is closed under coproducts.

Exercises

16.1 Suppose that A is a retract of B. Show that $\mathsf{count}(B) \to \mathsf{count}(A)$. Conclude that $\mathsf{is\text{-}finite}(B) \to \mathsf{is\text{-}finite}(A)$.

16.2 (a) Consider a family of decidable types A_i indexed by a finite type I. Show that the dependent product

$$\prod_{(i:I)} A_i$$

is decidable.

(b) Show that $\mathsf{is\text{-}emb}(f)$ is decidable, for any map $f : I \to J$ between finite types.

(c) Show that $\mathsf{is\text{-}surj}(f)$ is decidable, for any map $f : I \to J$ between finite types.

(d) Show that $\mathsf{is\text{-}equiv}(f)$ is decidable, for any map $f : I \to J$ between finite types.

16.3 Consider a finite type X.

(a) Show that any embedding $f : X \to X$ is an equivalence.

(b) Show that any surjective map $f : X \to X$ is an equivalence.

16.4 (a) Construct an equivalence $\mathsf{Fin}_{n^m} \simeq (\mathsf{Fin}_m \to \mathsf{Fin}_n)$. Conclude that if A and B are finite types, then $A \to B$ is finite.

(b) Construct an equivalence $\mathsf{Fin}_{n!} \simeq (\mathsf{Fin}_n \simeq \mathsf{Fin}_n)$. Conclude that if A is finite, then $A \simeq A$ is finite.

16.5 (a) Consider a set A and an arbitrary type B. Show that any embedding $A \hookrightarrow B$ factors uniquely through the embedding $(\mathbf{1} \hookrightarrow B) \hookrightarrow B$ given by $e \mapsto e(\star)$.

(b) A map $f : A \to B$ is said to be **decidable** if the type $\mathsf{fib}_f(b)$ is decidable for all $b : B$. Write $A \hookrightarrow_{\mathsf{d}} B$ for the type of decidable embeddings from A to B. Show that for any type A with decidable equality and an arbitrary type B, any embedding $A \hookrightarrow B$ factors uniquely through the embedding $(\mathbf{1} \hookrightarrow_{\mathsf{d}} B) \hookrightarrow B$.

(c) (Escardó) For any two types A and B, construct an equivalence

$$((A + \mathbf{1}) \simeq (B + \mathbf{1})) \simeq (\mathbf{1} \hookrightarrow_{\mathsf{d}} (B + \mathbf{1})) \times (A \simeq B).$$

16.6 (a) For any two types A and B, construct an equivalence

$$((A + \mathbf{1}) \hookrightarrow_{\mathsf{d}} (B + \mathbf{1})) \simeq (\mathbf{1} \hookrightarrow_{\mathsf{d}} (B + \mathbf{1})) \times (A \hookrightarrow_{\mathsf{d}} B).$$

(b) Construct an equivalence $\mathsf{Fin}_{(n)_m} \simeq (\mathsf{Fin}_m \hookrightarrow \mathsf{Fin}_n)$, where $(n)_m$ is the ***m*-th falling factorial** of n, which is defined recursively by

$$(0)_0 :\equiv 1 \qquad\qquad (0)_{m+1} :\equiv 0$$

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

$$(n+1)_0 :\equiv 1 \qquad (n+1)_{m+1} :\equiv (n+1)(n)_m.$$

Conclude that if A and B are finite with cardinality m and n, then the type $A \hookrightarrow B$ is finite with candinality $(n)_m$.

16.7 (a) Consider an arbitrary type A and a type B with decidable equality. Construct an equivalence

$$((A + \mathbf{1}) \twoheadrightarrow (B + \mathbf{1})) \simeq (B + \mathbf{1}) \times (A \twoheadrightarrow B) + (A \to B + \mathbf{1}).$$

(b) Construct an equivalence $\mathsf{Fin}_{\mathsf{S}(m,n)} \simeq (\mathsf{Fin}_m \twoheadrightarrow \mathsf{Fin}_n)$, where $\mathsf{S}(m, n)$ is defined recursively by

$$\begin{aligned} \mathsf{S}(0,0) &:\equiv 1 \\ \mathsf{S}(0,n+1) &:\equiv 0 \\ \mathsf{S}(m+1,0) &:\equiv 0 \\ \mathsf{S}(m+1,n+1) &:\equiv (n+1)\mathsf{S}(m,n) + \mathsf{S}(m,n+1). \end{aligned}$$

Conclude that if A and B are finite with cardinality m and n, then the type $A \twoheadrightarrow B$ is finite with cardinality $\mathsf{S}(m, n)$. Note: the number $\mathsf{S}(m, n)$ is $n!\left\{{m \atop n}\right\}$, where $\left\{{m \atop n}\right\}$ is the **Stirling number of the second kind** at (m, n).

16.8 Consider a surjective map $f : X \to Y$, and suppose that X is finite. Show that the following are equivalent:

(i) The type Y has decidable equality.
(ii) The type Y is finite.

16.9 Consider a family B of types over A.

(a) Show that if A is finite and if each $B(x)$ is finite, then the type

$$\textstyle\prod_{(x:A)} B(x)$$

is finite.

(b) Show that if A is finite and if $\prod_{(x:A)} B(x)$ is finite, then we have

$$\left(\textstyle\prod_{(x:A)} \|B(x)\|\right) \to \left(\prod_{(x:A)} \mathsf{is\text{-}finite}(B(x))\right).$$

(c) Show that if $\prod_{(x:A)} B(x)$ is finite and if each $B(x)$ is finite, then A is finite if and only if the following three conditions hold:

(i) A has decidable equality.
(ii) The type

$$\textstyle\sum_{(x:A)} |B(x)| \leq 1$$

is finite.

(iii) The type

$$\prod_{(x:A)} (2 \leq |B(x)|) \to B(x)$$

is finite.

16.10 Consider two arbitrary types A and B. For any 2-element type X, construct an equivalence

$$(A + B)^X \simeq A^X + X \times (A \times B) + B^X.$$

16.11 Consider two finite types X and Y with m and n elements, respectively, and let $f : X \to Y$ be a map.

(a) Show that

$$\mathsf{is\text{-}inj}(f) \to (m \leq n).$$

(b) Prove the pigeonhole principle, i.e., show that

$$(n > m) \to \exists_{(x,x':X)} (x \neq x') \times (f(x) = f(x')).$$

(c) Show that there is no embedding $\mathbb{N} \hookrightarrow \mathsf{Fin}_k$, for any $k : \mathbb{N}$.

16.12 A **unital magma** consists of a type A equipped with an element $e : A$ and a binary operation $\mu : A \to (A \to A)$ such that $\mu(e, y) = y$ and $\mu(x, e) = x$ for each $x, y : A$.

(a) Let $(A, e, \mu, \alpha, \beta)$ be a unital magma. Show that for any type X equipped with a counting $f : \mathsf{Fin}_k \simeq X$, the binary operation μ can be extended to an operation

$$\bar{\mu}_X : (X \to A) \to A.$$

(b) Show that if the unital magma $(A, e, \mu, \alpha, \beta)$ is associative, then the family of operations $\bar{\mu}$ is associative in the sense that there is an identification

$$\bar{\mu}_X (\lambda x.\, \bar{\mu}_{Y(x)} (f(x))) = \bar{\mu}_{\sum_{(x:X)} Y(x)} (\mathsf{ind}_\Sigma(f)).$$

for any family Y of types equipped with countings indexed by a type X equiped with a counting, and any $f : \prod_{(x:X)} Y(x) \to A$.

17 The univalence axiom

The univalence axiom characterizes the identity type of a universe. Roughly speaking, it asserts that equivalent types are equal. By the fundamental theorem of identity types, Theorem 11.2.2, it is immediate that the univalence axiom comes in three equivalent forms:

(i) The univalence axiom itself.
(ii) The total space of equivalences from A into any type in $\mathcal{U}$ is contractible.
(iii) The type family $A \simeq X$ indexed by $X : \mathcal{U}$ is an identity system. The induction principle that we obtain is called *equivalence induction*.

The univalence axiom is due to Voevodsky, who also showed that it is modeled in the simplicial sets. He also showed, in one of his first applications, that the univalence axiom implies function extensionality.

17.1 Equivalent forms of the univalence axiom

Theorem 17.1.1 *Consider a universe $\mathcal{U}$. The following are equivalent:*

(i) *The universe $\mathcal{U}$ is **univalent**: For any two types $A, B : \mathcal{U}$, the map*

$$\mathsf{equiv\text{-}eq} : (A = B) \to (A \simeq B)$$

given by $\mathsf{equiv\text{-}eq}(\mathsf{refl}) :\equiv \mathsf{id}$, *is an equivalence.*
(ii) *The type*

$$\textstyle\sum_{(B:\mathcal{U})} A \simeq B$$

is contractible for each $A : \mathcal{U}$.
(iii) *For any type $A : \mathcal{U}$, the family of types $A \simeq X$ indexed by $X : \mathcal{U}$ is an identity system on $\mathcal{U}$. In other words, the universe $\mathcal{U}$ satisfies the principle of **equivalence induction**: For every $A : \mathcal{U}$ and for every type family of types $P(X, e)$ indexed by $X : \mathcal{U}$ and $e : A \simeq X$, the map*

$$\left(\textstyle\prod_{(X:\mathcal{U})} \prod_{(e:A\simeq X)} P(X, e)\right) \to P(A, \mathsf{id})$$

given by $f \mapsto f(A, \mathsf{id})$ has a section.

Proof The claim is a special case of Theorem 11.2.2, the fundamental theorem of identity types. □

Axiom 17.1.2 (The univalence axiom) We will assume that all the universes generated by Postulate 6.2.1 are univalent. Given a univalent universe $\mathcal{U}$, we will write $\mathsf{eq\text{-}equiv}$ for the inverse of $\mathsf{equiv\text{-}eq}$.[1]

An important direct consequence of the univalence axiom is the principle of propositional extensionality. This principle asserts that any two logically

[1] The naming of the functions $\mathsf{equiv\text{-}eq}$ and $\mathsf{eq\text{-}equiv}$ is in the order of function application. The element $\mathsf{equiv\text{-}eq}(p)$ is the equivalence obtained from the identification p, whereas $\mathsf{eq\text{-}equiv}(e)$ is the identification (equality) obtained from the equivalence e. The type of the output comes first in the name, and the type of the input comes closest to the input itself.

equivalent propositions P and Q can be identified. Propositional extensionality is sometimes assumed in formal systems without the univalence axiom.

In order to prove propositional extensionality, we first observe that the univalence axiom also characterises the identity type of any subuniverse.

Proposition 17.1.3 *Consider a universe $\mathcal{U}$, and let P be a family of propositions over $\mathcal{U}$. Then the family of maps*

$$\mathsf{equiv\text{-}eq} : (A = B) \to (\mathsf{pr}_1(A) \simeq \mathsf{pr}_1(B))$$

indexed by $A, B : \sum_{(X:\mathcal{U})} P(X)$, given by $\mathsf{equiv\text{-}eq}(\mathsf{refl}) :\equiv \mathsf{id}$ *is an equivalence.*

Proof Since P is a subuniverse, it follows from Corollary 12.2.4 that the projection map is an embedding. Therefore we see that the asserted map is the composite of the equivalences

$$(A = B) \xrightarrow{\mathsf{ap}_{\mathsf{pr}_1}} (\mathsf{pr}_1(A) = \mathsf{pr}_1(B)) \xrightarrow{\mathsf{equiv\text{-}eq}} (\mathsf{pr}_1(A) \simeq \mathsf{pr}_1(B)). \qquad \square$$

Remark 17.1.4 Often, when P is a subuniverse and $A : \sum_{(X:\mathcal{U})} P(X)$, we will also write A for the type $\mathsf{pr}_1(A)$. Using this convention, the equivalence in Proposition 17.1.3 is displayed as

$$(A = B) \simeq (A \simeq B).$$

Important examples of subuniverses include the subuniverse $\mathsf{Prop}_{\mathcal{U}}$ of propositions in $\mathcal{U}$, the subuniverse $\mathsf{Set}_{\mathcal{U}}$ of sets in $\mathcal{U}$, and the subuniverse $\mathcal{U}^{\leq k}$ of k-truncated types in $\mathcal{U}$. The subuniverse $\mathbb{F}$ of finite types in $\mathcal{U}_0$, and the subuniverses $\mathbb{F}_k$ of k-element types are further important subuniverses to which Proposition 17.1.3 applies. Note that by the univalence axiom, any subuniverse is automatically closed under equivalences. Indeed, if we have $X \simeq Y$, then we have $P(X) \to P(Y)$ by transporting along the equality $X = Y$ induced by univalence.

Theorem 17.1.5 *Propositions satisfy* ***propositional extensionality****: for any two propositions P and Q, the canonical map*

$$\mathsf{iff\text{-}eq} : (P = Q) \to (P \leftrightarrow Q)$$

defined by $\mathsf{iff\text{-}eq}(\mathsf{refl}) :\equiv (\mathsf{id}, \mathsf{id})$ *is an equivalence. It follows that the type $\mathsf{Prop}_{\mathcal{U}}$ of propositions in $\mathcal{U}$ is a set.*

Proof Recall from Exercise 13.3 that $\mathsf{is\text{-}prop}(X)$ is a proposition for any type X. Proposition 17.1.3 therefore applies, which gives

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

The last equivalence follows from Proposition 12.1.4, using the fact that $(P \simeq Q)$ is a proposition by Exercise 13.4. □

Corollary 17.1.6 *The type*

$$\mathsf{dec\text{-}Prop}_{\mathcal{U}} :\equiv \sum_{(P:\mathsf{Prop}_{\mathcal{U}})} \mathsf{is\text{-}decidable}(P)$$

of decidable propositions in any universe $\mathcal{U}$ is equivalent to bool.

Proof Note that Σ distributes from the left over coproducts, so we have an equivalence

$$\left(\sum_{(P:\mathsf{Prop}_{\mathcal{U}})} P + \neg P\right) \simeq \left(\sum_{(P:\mathsf{Prop}_{\mathcal{U}})} P\right) + \left(\sum_{(Q:\mathsf{Prop}_{\mathcal{U}})} \neg Q\right).$$

Therefore it suffices to show that both $\sum_{(P:\mathsf{Prop}_{\mathcal{U}})} P$ and $\sum_{(Q:\mathsf{Prop}_{\mathcal{U}})} \neg Q$ are contractible. At the centers of contraction we have $(\mathbf{1}, \star)$ and $(\emptyset, \mathsf{id})$, respectively. For the contractions, note that both types are subtypes of the types of propositions. Therefore it suffices to show that $\mathbf{1} = P$ for any proposition P equipped with $p : P$, and that $\emptyset = Q$ for any proposition Q equipped with $q : \neg Q$. Both identifications are obtained immediately from propositional extensionality. □

17.2 Univalence implies function extensionality

One of the first applications of the univalence axiom was Voevodsky's theorem that the univalence axiom on a universe $\mathcal{U}$ implies function extensionality for types in $\mathcal{U}$. The proof uses the fact that weak function extensionality implies function extensionality. We will also make use of the following lemma.

Lemma 17.2.1 *For any equivalence $e : X \simeq Y$ in a univalent universe $\mathcal{U}$, and any type A, the post-composition map*

$$e \circ - : (A \to X) \to (A \to Y)$$

is an equivalence.

Note that this statement was also part of Exercise 13.12 (d). That exercise is solved using function extensionality. However, since our present goal is to derive function extensionality from the univalence axiom, we cannot make use of that exercise. Therefore we give a new proof, using the univalence axiom.

Proof Since $\mathcal{U}$ is assumed to be a univalent universe, it satisfies by Theorem 17.1.1 the principle of equivalence induction. Therefore, it suffices to show that the post-composition map

$$\mathsf{id} \circ - : (A \to X) \to (A \to X)$$

is an equivalence. This post-composition map is of course just the identity map on $A \to X$, so it is indeed an equivalence. □

Theorem 17.2.2 *For any universe $\mathcal{U}$, the univalence axiom on $\mathcal{U}$ implies function extensionality on $\mathcal{U}$.*

Proof Note that by Theorem 13.1.2 it suffices to show that univalence implies weak function extensionality. We note that the proof of Theorem 13.1.2 also goes through when it is restricted to types in $\mathcal{U}$.

Suppose that $B : A \to \mathcal{U}$ is a family of contractible types. Our goal is to show that the product $\prod_{(x:A)} B(x)$ is contractible. Since each $B(x)$ is contractible, the projection map $\mathsf{pr}_1 : \left(\sum_{(x:A)} B(x)\right) \to A$ is an equivalence by Exercise 10.7.

Now it follows by Lemma 17.2.1 that $\mathsf{pr}_1 \circ -$ is an equivalence. Consequently, it follows from Theorem 10.4.6 that the fibers of

$$\mathsf{pr}_1 \circ - : \left(A \to \sum_{(x:A)} B(x)\right) \to (A \to A)$$

are contractible. In particular, the fiber at id_A is contractible. Therefore it suffices to show that $\prod_{(x:A)} B(x)$ is a retract of $\sum_{(f : A \to \sum_{(x:A)} B(x))} \mathsf{pr}_1 \circ f = \mathsf{id}_A$. In other words, we will construct a section-retraction pair

$$\left(\prod_{(x:A)} B(x)\right) \xrightarrow{i} \left(\sum_{(f : A \to \sum_{(x:A)} B(x))} \mathsf{pr}_1 \circ f = \mathsf{id}_A\right) \xrightarrow{r} \left(\prod_{(x:A)} B(x)\right),$$

with $H : r \circ i \sim \mathsf{id}$.

We define the function i by

$$i(f) :\equiv (\lambda x.\, (x, f(x)), \mathsf{refl}_{\mathsf{id}}).$$

To see that this definition is correct, we need to know that

$$\lambda x.\, \mathsf{pr}_1(x, f(x)) \doteq \mathsf{id}.$$

This is indeed the case, by the rule λ-eq for Π-types, on Page 12.

Next, we define the function r. Consider a function $h : A \to \sum_{(x:A)} B(x)$ equipped with an identification $p : \mathsf{pr}_1 \circ h = \mathsf{id}$. Then we have the homotopy $\mathsf{htpy\text{-}eq}(p) : \mathsf{pr}_1 \circ h \sim \mathsf{id}$. Furthermore, we obtain $\mathsf{pr}_2(h(x)) : B(\mathsf{pr}_1(h(x)))$. Using these ingredients, we define r by

$$r((h, p), x) :\equiv \mathsf{tr}_B(\mathsf{htpy\text{-}eq}(p, x), \mathsf{pr}_2(h(x))).$$

It remains to construct a homotopy $H : r \circ i \sim \mathsf{id}$. We simply compute

$$\begin{aligned} r(i(f)) &\doteq r(\lambda x.\, (x, f(x)), \mathsf{refl}) \\ &\doteq \mathsf{tr}_B(\mathsf{htpy\text{-}eq}(\mathsf{refl}, x), \mathsf{pr}_2(x, f(x))) \\ &\doteq \mathsf{tr}_B(\mathsf{refl}, f(x)) \end{aligned}$$

$$\doteq f(x).$$

Thus we see that $r \circ i \doteq \mathsf{id}$ by an application of the η-rule for Π-types. Therefore we simply define $H(f) :\equiv \mathsf{refl}$. □

17.3 Maps and families of types

Using the univalence axiom, we can establish a fundamental relation between maps into a type A, and families of types indexed by A. A special case of this relation asserts that the type of all pairs (X, e) consisting of a type X and an embedding $e : X \hookrightarrow A$ is equivalent to the type of all subtypes of A, i.e., the type of all families P of propositions in $\mathcal{U}$ indexed by A.

Theorem 17.3.1 *For any type A and any univalent universe $\mathcal{U}$ containing A, the map*

$$\left(\textstyle\sum_{(X:\mathcal{U})} X \to A\right) \to (A \to \mathcal{U})$$

given by $(X, f) \mapsto \mathsf{fib}_f$ is an equivalence.

Proof The map in the converse direction is given by

$$B \mapsto \left(\textstyle\sum_{(x:A)} B(x), \mathsf{pr}_1\right).$$

To verify that this map is a section of the asserted map, we have to prove that

$$\mathsf{fib}_{\mathsf{pr}_1} = B$$

for any $B : A \to \mathcal{U}$. By function extensionality and the univalence axiom, this is equivalent to

$$\textstyle\prod_{(x:A)} \mathsf{fib}_{\mathsf{pr}_1}(x) \simeq B(x).$$

Such a family of equivalences was constructed in Exercise 10.7.

It remains to verify that

$$(X, f) = \left(\textstyle\sum_{(x:A)} \mathsf{fib}_f(x), \mathsf{pr}_1\right).$$

Before we do this, we claim that the identity type

$$(X, f) = (Y, g)$$

in the type $\sum_{(X:\mathcal{U})} X \to A$ is equivalent to the type of pairs (e, f) consisting of an equivalence $e : X \simeq Y$ equipped with a homotopy $f \sim g \circ e$. This fact follows from Theorem 11.2.2, because the type

$$\textstyle\sum_{(Y:\mathcal{U})} \sum_{(g:Y\to A)} \sum_{(e:X\simeq Y)} f \sim g \circ e$$

is contractible by the structure identity principle, Theorem 11.6.2.

To finish the proof, it therefore suffices to construct an equivalence

$$e : X \simeq \textstyle\sum_{(a:A)} \mathsf{fib}_f\,(a)$$

equipped with a homotopy $f \sim \mathsf{pr}_1 \circ e$. Such an equivalence e equipped with a homotopy was constructed in Exercise 10.8. □

The following corollary is so important, that we call it a theorem.

Theorem 17.3.2 *Consider a type A and a univalent universe $\mathcal{U}$ containing A. Furthermore, let P be a family of types indexed by $\mathcal{U}$, and write*

$$\mathcal{U}_P :\equiv \textstyle\sum_{(X:\mathcal{U})} P(X).$$

Then the map

$$\left(\textstyle\sum_{(X:\mathcal{U})} \sum_{(f:X\to A)} \prod_{(a:A)} P(\mathsf{fib}_f\,(a))\right) \to (A \to \mathcal{U}_P)$$

given by $(X, f, p) \mapsto \lambda a.\, (\mathsf{fib}_f\,(a), p(a))$ is an equivalence.

Proof The asserted map is homotopic to the composition of the equivalences

$$\begin{aligned}
&\textstyle\sum_{(X:\mathcal{U})} \sum_{(f:X\to A)} \prod_{(a:A)} P(\mathsf{fib}_f\,(a)) \\
&\simeq \textstyle\sum_{((X,f):\sum_{(X:\mathcal{U})} X\to A)} \prod_{(a:A)} P(\mathsf{fib}_f\,(a)) \\
&\simeq \textstyle\sum_{(B:A\to\mathcal{U})} \prod_{(a:A)} P(B(a)) \\
&\simeq A \to \textstyle\sum_{(X:\mathcal{U})} P(X).
\end{aligned}$$

□

Theorem 17.3.2 applies to any subuniverse. Examples include the subuniverse of k-types, for any truncation level k, the subuniverse of decidable propositions, the subuniverse of finite types, the subuniverse of inhabited types, and so on. It also applies to type families over $\mathcal{U}$ that aren't families of propositions. The families $P :\equiv \mathsf{is\text{-}decidable}$ and $P :\equiv \mathsf{count}$ are examples.

Corollary 17.3.3 *Consider a type A and a univalent universe $\mathcal{U}$ containing A. Then the map*

$$\left(\textstyle\sum_{(X:\mathcal{U})} X \hookrightarrow A\right) \to (A \to \mathsf{Prop}_{\mathcal{U}})$$

given by $(X, f) \mapsto \mathsf{fib}_f$ is an equivalence.

In other words, a subtype of a type A is the same thing as a type X equipped with an embedding $e : X \hookrightarrow A$. This brings us to an important point about equality of subtypes.

Remark 17.3.4 By function extensionality and propositional extensionality, it follows that two subtypes $P, Q : A \to \mathsf{Prop}_{\mathcal{U}}$ are the same if and only if

$$P(a) \leftrightarrow Q(a)$$

holds for all $a : A$. In other words, two subtypes of A are the same if and only if they contain the same elements of A.

On the other hand, by Corollary 17.3.3 we can also consider two types X and Y equipped with embeddings $f : X \hookrightarrow A$ and $g : Y \hookrightarrow A$ as subtypes of A. Now one might be inclined to think that the subtypes X and Y are equal if and only if X and Y are equivalent. However, this is not the case.

Using the structure identity principle, Theorem 11.6.2, we see that the identity type $(X, f) = (Y, g)$ in the type $\sum_{(X:\mathcal{U})} X \hookrightarrow A$ is equivalent to the type

$$\sum_{(e:X\simeq Y)} f \sim g \circ e.$$

In other words, two subtypes (X, f) and (Y, g) of A are equal if and only if there is an equivalence $X \simeq Y$ that is compatible with the embeddings $f : X \hookrightarrow A$ and $g : Y \hookrightarrow X$. Indeed, this condition is equivalent to the previous condition that two subtypes are the same if and only if they have the same elements.

We see that the combination of the structure identity principle and the univalence axiom automatically characterizes equality of subtypes in the most natural way, and we will see similar natural characerizations of identity types throughout this book.

17.4 Classical mathematics with the univalence axiom

In classical mathematics, the axiom of choice asserts that for any collection X of nonempty sets, there is a choice function f such that $f(x) \in x$ for each $x \in X$. The univalence axiom is consistent with the axiom of choice, but we have to be careful in our formulation of the axiom of choice to make it about sets. A naive interpretation that would be applicable to all types, such as the assertion that every family B of inhabited types has a section, is not consistent with univalence. We will use the type $\mathbb{F}_2$ of 2-element types for a counterexample.

Proposition 17.4.1 *The type*

$$\sum_{(X:\mathbb{F}_2)} X$$

of pointed 2-element types is contractible. Consequently, the canonical family of maps

$$(\mathsf{Fin}_2 = X) \to X$$

indexed by $X : \mathbb{F}_2$, is a family of equivalences.

Proof By the univalence axiom it follows that the type $\sum_{(X:\mathbb{F}_2)} \mathsf{Fin}_2 \simeq X$ is contractible. In order to show that $\sum_{(X:\mathbb{F}_2)} X$ is contractible, it therefore suffices to show that the map

$$f : (\mathsf{Fin}_2 \simeq X) \to X$$

given by $f(e) :\equiv e(\star)$, is an equivalence. Since being an equivalence is a proposition by Exercise 13.4, we may assume an equivalence $\alpha : \mathsf{Fin}_2 \simeq X$, and we proceed by equivalence induction on α. Therefore, it suffices to show that the map

$$f : (\mathsf{Fin}_2 \simeq \mathsf{Fin}_2) \to \mathsf{Fin}_2$$

give by $f(e) :\equiv e(\star)$ is an equivalence. Using the notation from Section 7.3, we define the inverse map g by

$$\begin{aligned} g(\star) &:\equiv \mathsf{id} \\ g(i(\star)) &:\equiv \mathsf{succ}_2, \end{aligned}$$

and it is a straightforward verification that f and g are inverse to each other. □

Corollary 17.4.2 *There is no dependent function*

$$\prod_{(X:\mathbb{F}_2)} X.$$

Proof By Proposition 17.4.1 and Exercise 13.12 (a), we have an equivalence

$$\left(\prod_{(X:\mathbb{F}_2)} \mathsf{Fin}_2 = X\right) \simeq \left(\prod_{(X:\mathbb{F}_2)} X\right).$$

Note that $\prod_{(X:\mathbb{F}_2)} \mathsf{Fin}_2 = X$ is the type of contractions of $\mathbb{F}_2$, using the center of contraction Fin_2. Therefore it suffices to show that $\mathbb{F}_2$ is not contractible. Recall from Exercise 10.1 that the identity types of contractible types are contractible. On the other hand, it follows from Proposition 17.4.1 that the identity type $\mathsf{Fin}_2 = \mathsf{Fin}_2$ in $\mathbb{F}_2$ is equivalent to Fin_2. This type isn't contractible by Exercise 10.4. We conclude that $\mathbb{F}_2$ is not contractible. □

The family $X \mapsto X$ over $\mathbb{F}_2$ is therefore an example of a family of nonempty types for which there are provably no sections. In the following corollary we conclude more generally that there is no way to construct an element of an arbitrary inhabited type.

Corollary 17.4.3 *If $\mathcal{U}$ is a univalent universe, then there is no function*

$$\prod_{(A:\mathcal{U})} \|A\| \to A.$$

Proof Suppose $f : \prod_{(A:\mathcal{U})} \|A\| \to A$. By restricting f to the type of 2-element types in $\mathcal{U}$, we obtain a function

$$\prod_{(A:\mathbb{F}_2)} \|A\| \to A.$$

Note that every 2-element type is inhabited, i.e., there is an element of type $\|A\|$ for every 2-element type A. To see this, consider a type $A : \mathcal{U}$ such that $\|\mathsf{Fin}_2 \simeq A\|$. To obtain an element of type $\|A\|$, we may assume an equivalence $e : \mathsf{Fin}_2 \simeq A$. Then we have $\eta(e(0)) : \|A\|$.

Since every 2-element type is inhabited, we obtain a function $\prod_{(A:\mathbb{F}_2)} A$, which is impossible by Corollary 17.4.2. □

Corollary 17.4.3 is of philosophical importance. It shows that the **principle of global choice** is incompatible with the univalence axiom, i.e., that there is no way to obtain construct a function $\|A\| \to A$ for all types A. In other words, we cannot obtain an element of A merely from the assumption that the type A is inhabited. What is the obstruction? It is the fact that no such choice of an element of A can be invariant under the automorphisms on A, i.e., under the self-equivalences on A. Indeed, in the example where A is the 2-element type Fin_2 there are no fixed point of the equivalence $\mathsf{succ}_2 : \mathsf{Fin}_2 \simeq \mathsf{Fin}_2$. By the univalence axiom, there is an identification $p : \mathsf{Fin}_2 = \mathsf{Fin}_2$ in $\mathcal{U}$, such that $\mathsf{tr}(p, x) = \mathsf{succ}_2(x)$. If we'd have a function

$$f : \prod_{(X:\mathcal{U})} \|X\| \to X,$$

the dependent action on paths of f would give an identification

$$\mathsf{apd}_f(p) : \mathsf{succ}_2(f(\mathsf{Fin}_2, p, H)) = f(\mathsf{Fin}_2, p, \eta(0)).$$

In other words, it would give us a fixed point for the successor function on Fin_2.

This is perhaps a good moment to stress that the axiom of choice is really an axiom about *sets*, not about more general types. And indeed, when we restrict the axiom of choice to sets, it turns out to be consistent with the univalence axiom and therefore safe to assume. In this book, however, we will not have many applications for the axiom of choice and therefore we will not assume it, unless we explicitly say otherwise.

Definition 17.4.4 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, i.e., it asserts that there is an element of type

$$\mathsf{AC}_{\mathcal{U}}(A, B) :\equiv \left(\prod_{(x:A)} \|B(x)\|\right) \to \left\|\prod_{(x:A)} B(x)\right\|,$$

for every $A : \mathsf{Set}_{\mathcal{U}}$ and $B : A \to \mathsf{Set}_{\mathcal{U}}$.

Similar care has to be taken with the type theoretic formulation of the law of excluded middle. It is again inconsistent to assume that every type is decidable.

Theorem 17.4.5 *There is no dependent function*

$$\prod_{(X:\mathcal{U})} \mathsf{is\text{-}decidable}(X).$$

Proof Suppose there is such a dependent function d. By restricting d to the subuniverse of 2-element types, we obtain a dependent function

$$d : \prod_{(X:\mathbb{F}_2)} \mathsf{is\text{-}decidable}(X).$$

However, each 2-element type X is inhabited. By Exercise 14.1 we obtain a function

$$\mathsf{is\text{-}decidable}(X) \to X$$

for each 2-element type X. Therefore, we obtain from d a dependent function $\prod_{(X:\mathbb{F}_2)} X$, which does not exist by Corollary 17.4.2. □

The law of excluded middle is really an axiom of propositional logic, and it is indeed consistent with the univalence axiom that every *proposition* is decidable.

Definition 17.4.6 The **law of excluded middle** asserts that every proposition is decidable, i.e.,

$$\mathsf{LEM}_{\mathcal{U}} :\equiv \prod_{(P:\mathsf{Prop}_{\mathcal{U}})} \mathsf{is\text{-}decidable}(P).$$

We will again not assume the law of excluded middle, unless we say otherwise. Nevertheless, we have seen in Section 8 that many propositions are already decidable without assuming the law of excluded middle, and decidability remains an important concept in type theory and mathematics.

17.5 The binomial types

Definition 17.5.1 A map $f : A \to B$ is said to be **decidable** if it comes equipped with an element of type

$$\mathsf{is\text{-}decidable}(f) :\equiv \prod_{(b:B)} \mathsf{is\text{-}decidable}(\mathsf{fib}_f(b)).$$

We will write $A \hookrightarrow_{\mathsf{d}} B$ for the type of decidable embeddings from A to B.

Definition 17.5.2 Consider a type A, and a universe $\mathcal{U}$. We define the **connected component** of $\mathcal{U}$ at A by

$$\mathcal{U}_A :\equiv \sum_{(X:\mathcal{U})} \|A \simeq X\|.$$

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

Example 17.5.3 Note that type $\mathcal{U}_{\mathsf{Fin}_k}$ is the type $\mathbb{F}_k$ of all k-element types. Note also that if $A \simeq B$, then $\mathcal{U}_A \simeq \mathcal{U}_B$.

Definition 17.5.4 Consider two types A and B and a universe $\mathcal{U}$ containing both A and B. We define the **binomial type** $\binom{A}{B}_{\mathcal{U}}$ by

$$\binom{A}{B}_{\mathcal{U}} :\equiv \textstyle\sum_{(X:\mathcal{U}_B)} X \hookrightarrow_{\mathsf{d}} A.$$

Remark 17.5.5 We define the binomial types using decidable embeddings because the usual properties of binomial coefficients generalise most naturally under the extra assumption of decidability. In particular the binomial theorem, which is stated as Exercise 17.11 and generalised in Exercise 18.15, rely on the use of decidable embeddings.

Proposition 17.5.6 *Consider two types A and B, and a universe $\mathcal{U}$ containing both A and B. Then we have an equivalence*

$$\binom{A}{B}_{\mathcal{U}} \simeq \sum_{(P:A\to \mathsf{dec\text{-}Prop}_{\mathcal{U}})} \left\| B \simeq \textstyle\sum_{(a:A)} P(a) \right\|.$$

Proof This equivalence follows from Theorem 17.3.2, by which we have

$$\left(\textstyle\sum_{(X:\mathcal{U})} X \hookrightarrow_{\mathsf{d}} A\right) \simeq (A \to \mathsf{dec\text{-}Prop}_{\mathcal{U}}).$$ □

Remark 17.5.7 Combining Corollary 17.1.6 and Proposition 17.5.6, we obtain an equivalence

$$\binom{A}{B}_{\mathcal{U}} \simeq \sum_{(f:A\to \mathsf{bool})} \left\| B \simeq \textstyle\sum_{(a:A)} f(a) = \mathsf{true} \right\|.$$

for any universe $\mathcal{U}$ that contains both A and B. This equivalence is important, because the right hand side doesn't depend on the universe $\mathcal{U}$. Therefore we will simply write $\binom{A}{B}$ for $\binom{A}{B}_{\mathcal{U}}$, if the universe $\mathcal{U}$ contains both A and B.

Lemma 17.5.8 *For any two types A and B, we have equivalences*

$$\binom{\emptyset}{\emptyset} \simeq \mathbf{1} \qquad \binom{A+\mathbf{1}}{\emptyset} \simeq \mathbf{1}$$

$$\binom{\emptyset}{B+\mathbf{1}} \simeq \emptyset \qquad \binom{A+\mathbf{1}}{B+\mathbf{1}} \simeq \binom{A}{B} + \binom{A}{B+\mathbf{1}}.$$

Proof For the first two equivalences, we prove that $\binom{X}{\emptyset}$ is contractible for any type X. To see this, we first note that the type $\mathcal{U}_\emptyset$ is contractible. Indeed, at the center of contraction we have the empty type, and any two types that are

merely equivalent to the empty type are empty and hence equivalent. Therefore it follows that

$$\binom{X}{\emptyset} \simeq \emptyset \hookrightarrow_{\mathsf{d}} X.$$

The type of decidable embeddings $\emptyset \hookrightarrow_{\mathsf{d}} X$ is contractible, because the type $\emptyset \to X$ is contractible with the map ex-falso : $\emptyset \to X$ at the center of contraction, which is of course a decidable embedding.

Next, the fact that the binomial type $\binom{\emptyset}{B+\mathbf{1}}$ is empty follows from the fact that the type of maps $X \to \emptyset$ is empty for any type X merely equivalent to $B + \mathbf{1}$.

For the last equivalence we will use Proposition 17.5.6. Using the universal property of $A + \mathbf{1}$, we see that

$$\binom{A+\mathbf{1}}{B+\mathbf{1}} \simeq \textstyle\sum_{(P:A\to \mathsf{dec\text{-}Prop}_{\mathcal{U}})} \sum_{(Q:\mathsf{dec\text{-}Prop}_{\mathcal{U}})} \|(B+\mathbf{1}) \simeq \sum_{(a:A)} P(a) + Q\|.$$

Using the fact that $\mathsf{dec\text{-}Prop}_{\mathcal{U}} \simeq \mathsf{Fin}_2$, observe that we have an equivalence

$$\begin{aligned}\textstyle\sum_{(Q:\mathsf{dec\text{-}Prop}_{\mathcal{U}})} \|(B+\mathbf{1}) \simeq (\textstyle\sum_{(a:A)} P(a) + Q)\| \\ \simeq \|(B+\mathbf{1}) \simeq (\textstyle\sum_{(a:A)} P(a) + \mathbf{1})\| + \|(B+\mathbf{1}) \simeq \textstyle\sum_{(a:A)} P(a)\|.\end{aligned}$$

Furthermore, note that it follows from Proposition 16.2.1 that

$$\|(B+\mathbf{1}) \simeq (\textstyle\sum_{(a:A)} P(a) + \mathbf{1})\| \simeq \|B \simeq \textstyle\sum_{(a:A)} P(a)\|.$$

Thus we see that

$$\begin{aligned}\binom{A+\mathbf{1}}{B+\mathbf{1}} \simeq \left(\textstyle\sum_{(P:A\to \mathsf{dec\text{-}Prop}_{\mathcal{U}})} \|B \simeq \textstyle\sum_{(a:A)} P(a)\|\right) \\ + \left(\textstyle\sum_{(P:A\to \mathsf{dec\text{-}Prop}_{\mathcal{U}})} \|(B+\mathbf{1}) \simeq \textstyle\sum_{(a:A)} P(a)\|\right).\end{aligned}$$

□

Theorem 17.5.9 *If A and B are finite types of cardinality n and k, respectively, then the type $\binom{A}{B}$ is finite of cardinality $\binom{n}{k}$.*

Proof The claim that the type $\binom{A}{B}$ is finite of cardinality $\binom{n}{k}$ is a proposition, so we may assume $e : \mathsf{Fin}_n \simeq A$ and $f : \mathsf{Fin}_k \simeq B$. The claim now follows by induction on n and k, using Lemma 17.5.8. □

Remark 17.5.10 It is perhaps remarkable that the type $\sum_{(X:\mathcal{U}_B)} X \hookrightarrow_{\mathsf{d}} A$ is a good generalisation of the binomial coefficients to types. When A and B are finite types of cardinality n and k, respectively, then the type $B \hookrightarrow_{\mathsf{d}} A$ has a factor $k!$ too many elements. When we seemingly enlarge it by the type $\mathcal{U}_B$ of all types merely equivalent to B, it turns out that we obtain the correct generalisation of the binomial coefficients.

One reason why it works is that the identity type of $\sum_{(X:\mathcal{U}_B)} X \hookrightarrow_{\mathsf{d}} A$ is characterised, via the univalence axiom, by

$$((X, f) = (Y, g)) \simeq \sum_{(e:X\simeq Y)} f \sim g \circ e.$$

Therefore it follows that for any two decidable embeddings $f, g : B \hookrightarrow_{\mathsf{d}} A$, if f and g are the same up to a permutation on B, then we get an identification $(B, f) = (B, g)$ in the type $\sum_{(X:\mathcal{U}_B)} X \hookrightarrow_{\mathsf{d}} A$.

From a group theoretic perspective we may observe that the automorphism group $B \simeq B$ acts freely on the set of decidable embeddings $\mathrm{B} \hookrightarrow_{\mathsf{d}} A$, and the type $\sum_{(X:\mathcal{U}_B)} X \hookrightarrow A$ can be viewed as the type of orbits of that action. Since this action of $\mathrm{Aut}(B)$ on $B \hookrightarrow_{\mathsf{d}} A$ is free, we see that the number of orbits is $\frac{1}{k!}$ times the number of elements in $B \hookrightarrow_{\mathsf{d}} A$. We will study groups and group actions in more detail in Section 28.

Exercises

17.1 (a) Use the univalence axiom to show that the type $\sum_{(A:\mathcal{U})} \mathsf{is\text{-}contr}(A)$ of all contractible types in $\mathcal{U}$ is contractible.

(b) Use the univalence axiom and Exercises 13.3 and 13.4 to show that the universe of k-types

$$\mathcal{U}^{\leq k} :\equiv \sum_{(X:\mathcal{U})} \mathsf{is\text{-}trunc}_k(X)$$

is a $(k+1)$-type, for any $k \geq -2$.

(c) Show that $\mathsf{Prop}_{\mathcal{U}}$ is not a proposition.

(d) Show that the universe $\mathsf{Set}_{\mathcal{U}}$ of sets in $\mathcal{U}$ is not a set.

17.2 Give an example of a type family B over a type A for which the implication

$$\neg\left(\prod_{(x:A)} B(x)\right) \to \left(\sum_{(x:A)} \neg B(x)\right)$$

is false.

17.3 Consider a type A and a univalent universe $\mathcal{U}$ containing A. Construct an equivalence

$$A \simeq \sum_{(B:A\to\mathcal{U})} \mathsf{is\text{-}contr}\left(\sum_{(a:A)} B(a)\right).$$

17.4 Consider a map $f : A \to B$. Show that the following are equivalent:

(i) The map f is surjective.

(ii) For every set C, the precomposition function

$$- \circ f : (B \to C) \to (A \to C)$$

is an embedding.

Hint: To show that (ii) implies (i), use the assumption with the set $C \coloneqq \mathsf{Prop}_{\mathcal{U}}$, where $\mathcal{U}$ is a univalent universe containing both A and B.

17.5 (Escardó) Consider a type A in $\mathcal{U}$. Show that the identity type, seen as a function

$$\mathsf{Id} : A \to (A \to \mathcal{U}),$$

is an embedding.

17.6 (a) For any type A in $\mathcal{U}$, consider the function

$$\Sigma_A : (A \to \mathcal{U}) \to \mathcal{U},$$

which takes a family B of $\mathcal{U}$-small types to its Σ-type. Show that the following are equivalent:

(i) The type A is k-truncated.
(ii) The map Σ_A is k-truncated.

Hint: Construct an equivalence $\mathsf{fib}_{\Sigma_A}(X) \simeq (X \to A)$.

(b) Show that the map $+ : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$, which takes (A, B) to the coproduct $A + B$, is 0-truncated.

17.7 (Escardó) Consider a proposition P and a universe $\mathcal{U}$ containing P. Show that the map

$$\Pi_P : (P \to \mathcal{U}) \to \mathcal{U},$$

given by $A \mapsto \prod_{(p:P)} A(p)$, is an embedding.

17.8 For any $k : \mathbb{N}$, show that the type

$$\sum_{(X:\mathbb{F}_{k+1})} \mathsf{Fin}_k \hookrightarrow X$$

is contractible.

17.9 Consider a type A.

(a) Recall from Exercise 12.14 that an element $a : A$ is isolated if and only if the map $\mathsf{const}_a : \mathbf{1} \to A$ is a decidable embedding. Construct an equivalence

$$\binom{A}{\mathbf{1}} \simeq \sum_{(a:A)} \mathsf{is\text{-}isolated}(a).$$

(b) Construct an equivalence

$$\binom{A}{\mathbf{1}} \simeq \left(\sum_{(X:\mathcal{U})} (X + \mathbf{1}) \simeq A\right).$$

Conclude that the map $X \mapsto X + \mathbf{1}$ on a univalent universe $\mathcal{U}$ is 0-truncated.

There are 306 exercises.

Ljubljana, November 10, 2021

(c) More generally, construct an equivalence

$$\binom{A}{B} \simeq \sum_{(X:\mathcal{U}_B)} \sum_{(Y:\mathcal{U})} (X + Y \simeq A).$$

17.10 (a) For any type A, construct an equivalence

$$\mathcal{U}_A \simeq \sum_{(X:\mathcal{U}_{A+1})} \binom{X}{\mathbf{1}}.$$

(b) For any $k : \mathbb{N}$, construct an equivalence

$$\left(\sum_{(X:\mathbb{F}_{k+1})} X\right) \simeq \mathbb{F}_k.$$

In other words, show that the type of $(k+1)$-element types equipped with a point is equivalent to the type of k-element types. Conclude that the type of pointed finite types is equivalent to the type of finite types, i.e., conclude that we have an equivalence

$$\left(\sum_{(X:\mathbb{F})} X\right) \simeq \mathbb{F}.$$

17.11 For any $(X, i) : \binom{A}{B}$, we define $A \setminus (X, i) :\equiv (A \setminus X, A \setminus i) : \binom{A}{B}$, where

$$A \setminus X :\equiv \sum_{(a:A)} \neg(\mathsf{fib}_i(a))$$
$$A \setminus i :\equiv \mathsf{pr}_1.$$

Now consider a finite type X and two arbitrary types A and B. Construct an equivalence

$$(A + B)^X \simeq \sum_{(k:\mathbb{N})} \sum_{((Y,i):\binom{X}{\mathsf{Fin}_k})} A^Y \times B^{X \setminus Y}.$$

17.12 Consider two types A and B. The **Stirling type of the second kind** is the type

$$\left\{\begin{matrix} A \\ B \end{matrix}\right\} :\equiv \sum_{(X:\mathcal{U}_B)} A \twoheadrightarrow X.$$

(a) Suppose that B has decidable equality. Construct an equivalence

$$\left\{\begin{matrix} A + \mathbf{1} \\ B + \mathbf{1} \end{matrix}\right\} \simeq (B + \mathbf{1}) \left\{\begin{matrix} A \\ B + \mathbf{1} \end{matrix}\right\} + \left\{\begin{matrix} A \\ B \end{matrix}\right\}$$

(b) Suppose that A and B are finite types of cardinality n and k. Show that the Stirling type $\left\{\begin{matrix} A \\ B \end{matrix}\right\}$ of the second kind is a finite type of cardinality $\left\{\begin{matrix} n \\ k \end{matrix}\right\}$, where $\left\{\begin{matrix} n \\ k \end{matrix}\right\}$ is the **Stirling number of the second kind**.

17.13 (a) Show that for $k \neq 2$, the type

$$\prod_{(X:\mathbb{F}_k)} X \to X$$

is contractible. Conclude that the type $\prod_{(X:\mathbb{F}_k)} X \simeq X$ is also contractible. Hint: Use Exercise 17.10.

(b) Show that the type

$$\prod_{(X:\mathbb{F}_2)} X \to X$$

is equivalent to Fin_2. Conclude that the type $\prod_{(X:\mathbb{F}_2)} X \simeq X$ is also equivalent to Fin_2.

17.14 (a) Show that for $k \geq 3$, the type

$$\prod_{(X:\mathbb{F}_k)} (X + X) \hookrightarrow (X \times X) + \mathbf{1}$$

is empty, even though the inequality $2k \leq k^2 + 1$ holds for all $k : \mathbb{N}$.

(b) Show that the type

$$\prod_{(X:\mathbb{F}_2)} (X + X) \hookrightarrow (X \times X) + \mathbf{1}$$

is equivalent to Fin_8.

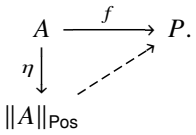
17.15 Consider a preorder $(A, \leq)$, and define for any $a : A$ the order preserving map

$$y_a : \mathsf{PreOrd}((A, \leq)^{\mathsf{op}}, (\mathsf{Prop}_{\mathcal{U}}, \to))$$

by $y_a(x) :\equiv (x \leq a)$. Furthermore, define the **poset reflection** $\|A\|_{\mathsf{Pos}}$ to be the image of the map

$$a \mapsto y_a : A \to \mathsf{PreOrd}((A, \leq)^{\mathsf{op}}, (\mathsf{Prop}_{\mathcal{U}}, \to)).$$

Equip the type $\|A\|_{\mathsf{Pos}}$ with the structure of a poset and construct an order preserving map $\eta : A \to \|A\|_{\mathsf{Pos}}$ that satisfies the following universal property: For any poset P, any order preserving map $f : A \to P$ extends uniquely along η to an order preserving map $g : \|A\|_{\mathsf{Pos}} \to P$, as indicated in the following diagram:



17.16 Given a type A, the type of **unordered pairs** in A is defined to be

$$\mathsf{unordered\text{-}pairs}(A) :\equiv \textstyle\sum_{(X:\mathbb{F}_2)} A^X.$$

The type of homotopy commutative binary operations from A to B is defined using the type of unordered pairs as

$$\mathsf{unordered\text{-}pairs}(A) \to B.$$

(a) Show that if B is a set, then this type is equivalent to the type

$$\textstyle\sum_{(m:A\to(A\to B))} \prod_{(x,y:A)} m(x,y) = m(y,x).$$

(b) Show that the type of undirected graphs in $\mathcal{U}$ with at most one edge between any two points

$$\textstyle\sum_{(V:\mathcal{U})} \Big(\mathsf{unordered\text{-}pairs}(V) \to \mathsf{Prop}_{\mathcal{U}}\Big).$$

is equivalent to the type

$$\textstyle\sum_{(V:\mathcal{U})} \sum_{(E:V\to(V\to\mathsf{Prop}_{\mathcal{U}}))} \prod_{(x,y:V)} E(x,y) \to E(y,x).$$

Note: More generally, the type of **(undirected) graphs** in a univalent universe $\mathcal{U}$ is defined to be

$$\mathsf{Graph}_{\mathcal{U}} :\equiv \textstyle\sum_{(V:\mathcal{U})} \Big(\mathsf{unordered\text{-}pairs}(V) \to \mathcal{U}\Big).$$

18 Set quotients

In this section we construct the quotient of a type by an equivalence relation. By an equivalence relation we understand a binary relation R which is reflexive, symmetric, and transitive. Moreover, we require that the type $R(x,y)$ relating x and y is a proposition. Therefore, if $\mathcal{U}$ is a universe that contains the types A and $R(x,y)$ for each $x, y : A$, then we can view R as a map

$$A \to (A \to \mathsf{Prop}_{\mathcal{U}}).$$

The quotient A/R is constructed as the type of equivalence classes, which is just the image of the map $R : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$. Thus, our construction of the quotient by an equivalence relation is very much like the classical construction of a quotient set. Examples of set quotients are abundant. We cover two of them: the type of rational numbers and the set truncation of a type.

There is, however, a subtle issue with our construction of the set quotient as the image of the map $R : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$. What universe is the quotient A/R in? Note that $\mathsf{Prop}_{\mathcal{U}}$ is a type in the successor universe $\mathcal{U}^+$, constructed in Definition 6.2.3. Therefore the function type $A \to \mathsf{Prop}_{\mathcal{U}}$ as well as the quotient A/R are also types in $\mathcal{U}^+$. That seems unfortunate, because in Zermelo-Fraenkel

set theory the quotient of a set by an equivalence relation is an ordinary set, and not a more general class.

In Zermelo-Fraenkel set theory quotients are are sets because of the axiom schema of replacement. The replacement axioms assert that the image of any function is again a set. This leads us to wonder about a type theoretical variant of the replacement axioms. Indeed, there is such a variant. The type theoretic replacement property asserts that for any map $f : A \to B$ from a type A in $\mathcal{U}$ to a type B of which the *identity types* are equivalent to types in $\mathcal{U}$, the image of f is also equivalent to a type in $\mathcal{U}$, and in fact this property is a theorem. We prove it in Theorem 26.6.11, using the univalence axiom and a new construction of the image of a map.

18.1 Equivalence relations and the replacement axiom

Definition 18.1.1 Let $R : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$ be a binary relation valued in the propositions. We say that R is an **equivalence relation** if R comes equipped with

$$\begin{aligned}\rho &: \textstyle\prod_{(x:A)} R(x,x)\\ \sigma &: \textstyle\prod_{(x,y:A)} R(x,y) \to R(y,x)\\ \tau &: \textstyle\prod_{(x,y,z:A)} R(x,y) \to (R(y,z) \to R(x,z)),\end{aligned}$$

witnessing that R is reflexive, symmetric, and transitive.

Example 18.1.2 Consider the type

$$Q :\equiv \mathbb{Z} \times \left(\textstyle\sum_{(x:\mathbb{N})} 0 < x\right).$$

We define $((x,(y,p)) \sim (x',(y',p'))) :\equiv (xy' = x'y)$. Then the relation $\sim$ is an equivalence relation on Q. Indeed, we have

$$\mathsf{refl} : (x,(y,p)) \sim (x,(y,p))$$

for any $(x,(y,p)) : Q$, and

$$r^{-1} : (x',(y',p')) \sim (x,(y,p))$$

for any $r : (x,(y,p)) \sim (x',(y',p'))$.

Definition 18.1.3 Let $R : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$ be an equivalence relation. The **equivalence class** of $x : A$ is defined to be

$$[x]_R :\equiv R(x).$$

More generally, a subtype $P : A \to \mathsf{Prop}_{\mathcal{U}}$ is said to be an **equivalence class** if it satisfies

$$\mathsf{is\text{-}equivalence\text{-}class}(P) :\equiv \exists_{(x:A)} P = R(x).$$

Furthermore, we define A/R to be the type of equivalence classes, i.e., we define

$$A/R :\equiv \textstyle\sum_{(P:A\to\mathsf{Prop}_{\mathcal{U}})} \mathsf{is\text{-}equivalence\text{-}class}(P).$$

In other words, A/R is the image of the map $[-]_R : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$. In the following proposition we characterize the identity type of A/R. As a corollary, we obtain equivalences

$$([x]_R = [y]_R) \simeq R(x, y),$$

justifying that the quotient A/R is defined to be the type of equivalence classes. Note that in our characterization of the identity type of A/R we make use of the univalence axiom.

Proposition 18.1.4 *Let $R : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$ be an equivalence relation. Furthermore, consider $x : A$ and an equivalence class P. Then the canonical map*

$$([x]_R = P) \to P(x)$$

is an equivalence.

Proof By Theorem 11.2.2 it suffices to show that the total space

$$\textstyle\sum_{(P:A/R)} P(x)$$

is contractible. The center of contraction is of course $[x]_R$, which satisfies $[x]_R(x)$ by reflexivity of R. It remains to construct a contraction. Since $\sum_{(P:A/R)} P(x)$ is a subtype of A/R, we construct a contraction by showing that

$$[x]_R = P$$

whenever $P(x)$ holds. Recall that P is an equivalence class, i.e., that there exists an element $y : A$ such that $P = [y]_R$. Note that our goal is a proposition, so we may assume that we have such a y. Then we obtain that $R(x, y)$ holds from the assumption that $P(x)$ holds. Thus, we have to show that

$$[x]_R = [y]_R$$

given that $R(x, y)$ holds. By function extensionality and propositional extensionality, it is equivalent to show that

$$\textstyle\prod_{(z:A)} R(x, z) \leftrightarrow R(y, z)$$

We get a function $R(x,z) \to R(y,z)$ by transitivity, since $R(y,x)$ holds by symmetry. Conversely, we get a function $R(y,z) \to R(x,z)$ directly by transitivity. □

Corollary 18.1.5 *Consider an equivalence relation R on a type A, and let $x, y : A$. Then there is an equivalence*

$$([x]_R = [y]_R) \simeq R(x,y).$$

Proof By Proposition 18.1.4 we have an equivalence

$$([x]_R = [y]_R) \simeq R(y,x).$$

Moreover, $R(y,x)$ is equivalent to $R(x,y)$ by symmetry of R. □

A question that sometimes comes up in mathematics, is whether a certain type or object is small with respect to a universe. Notice that in type theory, the type of equivalence classes of an equivalence relation in $\mathcal{U}$ is only a type in $\mathcal{U}^+$, the universe that contains $\mathcal{U}$ and every type in $\mathcal{U}$. Indeed, the type $\mathsf{Prop}_\mathcal{U}$ of propositions in $\mathcal{U}$ is only a type in $\mathcal{U}^+$, from which it follows that the type $A \to \mathsf{Prop}_\mathcal{U}$ is only a type in $\mathcal{U}^+$. The type of equivalence classes of an equivalence relation R on A in $\mathcal{U}$ is a subtype of $A \to \mathsf{Prop}_\mathcal{U}$, so we conclude that A/R is a type in $\mathcal{U}^+$. In classical mathematics, on the other hand, we consider the class of equivalence classes of an equivalence relation to be a (small) set. We introduce the notion of smallness to type theory.

Definition 18.1.6 Consider a universe $\mathcal{U}$.

(i) A type A is said to be **$\mathcal{U}$-small** if there is a type $X : \mathcal{U}$ and an equivalence $A \simeq X$. We write

$$\mathsf{is\text{-}small}_\mathcal{U}(A) :\equiv \textstyle\sum_{(X:\mathcal{U})} A \simeq X.$$

Similarly, a map $f : A \to B$ is said to be **$\mathcal{U}$-small** if all of its fibers are $\mathcal{U}$-small.

(ii) A type A is said to be **locally $\mathcal{U}$-small** if for every $x, y : A$ the identity type $x = y$ is $\mathcal{U}$-small. We write

$$\mathsf{is\text{-}locally\text{-}small}_\mathcal{U}(A) :\equiv \textstyle\prod_{(x,y:A)} \mathsf{is\text{-}small}_\mathcal{U}(x = y).$$

Similarly, a map $f : A \to B$ is said to be **locally $\mathcal{U}$-small** if all of its fibers are locally $\mathcal{U}$-small.

Example 18.1.7

(i) Any $\mathcal{U}$-small type is also locally $\mathcal{U}$-small.

(ii) Any contractible type is $\mathcal{U}$-small with respect to any universe, and any proposition is locally small with respect to any universe $\mathcal{U}$.

(iii) Any univalent universe $\mathcal{U}$ is locally $\mathcal{U}$-small, because by the univalence axiom we have an equivalence

$$(A = B) \simeq (A \simeq B)$$

for each $A, B : \mathcal{U}$, and the type $A \simeq B$ is in $\mathcal{U}$.

(iv) For any family P of $\mathcal{U}$-small types over a $\mathcal{U}$-small type A, the dependent product $\prod_{(x:A)} B(x)$ is $\mathcal{U}$-small. Furthermore, for any family P of locally $\mathcal{U}$-small types over a $\mathcal{U}$-small type A, the dependent product $\prod_{(x:A)} P(x)$ is locally $\mathcal{U}$-small. In particular, any type $A \to B$ of functions from a small type into a small or locally small type is again small or locally small, respectively.

(v) The type of $\mathcal{U}$-small types in $\mathcal{V}$ is equivalent to the type of $\mathcal{V}$-small types in $\mathcal{U}$. This follows from the equivalence

$$\left(\sum_{(Y:\mathcal{V})} \sum_{(X:\mathcal{U})} Y \simeq X\right) \simeq \left(\sum_{(X:\mathcal{U})} \sum_{(Y:\mathcal{V})} X \simeq Y\right).$$

(vi) If the law of excluded middle holds for propositions in $\mathcal{U}$, then we obtain from Corollary 17.1.6 an equivalence

$$\mathsf{Prop}_{\mathcal{U}} \simeq \mathsf{bool}.$$

This implies that the type $\mathsf{Prop}_{\mathcal{U}}$ is $\mathcal{U}_0$-small, and that any set is locally $\mathcal{U}_0$-small.

(vii) In Theorem B.5.10 we will show that $\mathcal{U}$ cannot be $\mathcal{U}$-small, i.e., that there can be no type $U : \mathcal{U}$ equipped with an equivalence $U \simeq \mathcal{U}$.

By the following proposition there is at most one way in which a type can be $\mathcal{U}$-small.

Proposition 18.1.8 *The type* $\mathsf{is\text{-}small}_{\mathcal{U}}(A)$ *is a proposition for any type* A.

Proof Let A be an arbitrary type. In order to show that $\mathsf{is\text{-}small}_{\mathcal{U}}(A)$ is a proposition, we will use Proposition 12.1.3 and show that for any $X : \mathcal{U}$ and any equivalence $e : A \simeq X$, the type

$$\sum_{(Y:\mathcal{U})} A \simeq Y$$

is contractible. Note that we have an equivalence

$$\left(\sum_{(Y:\mathcal{U})} X \simeq Y\right) \simeq \left(\sum_{(Y:\mathcal{U})} A \simeq Y\right)$$

because precomposing with the equivalence $e : A \simeq X$ is an equivalence. However, the type $\sum_{(Y:\mathcal{U})} X \simeq Y$ is contractible by Theorem 17.1.1. □

The following is an immediate corollary.

Corollary 18.1.9 *For each function $f : A \to B$, the type* $\mathsf{is\text{-}small}_{\mathcal{U}}(f)$ *is a proposition, and for each type X the type* $\mathsf{is\text{-}locally\text{-}small}_{\mathcal{U}}(X)$ *is a proposition.*

Since the type of $\mathcal{U}$-small types in a universe $\mathcal{V}$ is a subuniverse of $\mathcal{V}$, we also obtain the following corollary.

Corollary 18.1.10 *Consider a type A in $\mathcal{U}$ and an arbitrary type B. If $\|A \simeq B\|$, then B is $\mathcal{U}$-small.*

Proof Since $\mathsf{is\text{-}small}_{\mathcal{U}}(B)$ is a proposition, we may assume $A \simeq X$ and the claim follows immediately. □

Corollary 18.1.11 *Any finite type is $\mathcal{U}$-small for any universe $\mathcal{U}$. Consequently, we get equivalences*

$$\left(\textstyle\sum_{(X:\mathcal{U})} \mathsf{is\text{-}finite}(X)\right) \simeq \left(\textstyle\sum_{(Y:\mathcal{V})} \mathsf{is\text{-}finite}(Y)\right)$$

for any two univalent universes $\mathcal{U}$ and $\mathcal{V}$.

Using Proposition 18.1.8 we can show that the universe inclusions

$$\mathcal{U} \to \mathcal{U}^+$$
$$\mathcal{U} \to \mathcal{U} \sqcup \mathcal{V}$$
$$\mathcal{V} \to \mathcal{U} \sqcup \mathcal{V}.$$

constructed in Remarks 6.2.4 and 6.2.6 are embeddings. These claims follow from the following, slightly more general theorem.

Theorem 18.1.12 *Consider two universes $\mathcal{U}$ and $\mathcal{V}$, and suppose that $\mathcal{V}$ contains every type in $\mathcal{U}$. Then the universe inclusion*

$$i :\equiv \lambda X.\, \check{\mathcal{T}}(X) : \mathcal{U} \to \mathcal{V}$$

is an embedding.

Proof By univalence it follows that $\mathsf{fib}_i(Y) \simeq \mathsf{is\text{-}small}_{\mathcal{U}}(Y)$ for any $Y : \mathcal{V}$, which is a proposition by Proposition 18.1.8. □

Recall that in set theory, the replacement axiom asserts that for any family of sets $\{X_i\}_{i \in I}$ indexed by a set I, there is a set $X[I]$ consisting of precisely those sets x for which there exists an $i \in I$ such that $x \in X_i$. In other words: the image of a set-indexed family of sets is again a set. Without the replacement axiom, $X[I]$ would be a class.

In type theory, we may similarly wonder about the universe level of the image of a map $X : I \to \mathcal{U}$, given that I is in $\mathcal{U}$. Recall from Definition 6.1.1

that a type A is in $\mathcal{U}$ if $\mathcal{U}$ comes equipped with an element $\check{A} : \mathcal{U}$ such that $\mathcal{T}(\check{A}) \doteq A$. We make the following generalization of this concept.

Axiom 18.1.13 For any map $f : A \to B$ from a small type A into a locally small type B, the image of f is again small.

Example 18.1.14 For any type $A : \mathcal{U}$, the type $\mathcal{U}_A$ of all types in UU merely equivalent to A is is equivalent to the image of the constant map $\mathsf{const}_A : \mathbf{1} \to \mathcal{U}$ is small. Since $\mathbf{1}$ is small and $\mathcal{U}$ is locally $\mathcal{U}$-small, it follows from the replacement axiom that $\mathcal{U}_A$ is $\mathcal{U}$-small.

Example 18.1.15 The type $\mathbb{F}$ of all finite types is equivalent to be the image of the map

$$\mathsf{Fin} : \mathbb{N} \to \mathcal{U}_0.$$

Since $\mathbb{N}$ is $\mathcal{U}$-small and $\mathcal{U}_0$ is locally $\mathcal{U}$-small, it follows from the replacement axiom that $\mathbb{F}$ is $\mathcal{U}$-small.

18.2 The universal property of set quotients

The quotient A/R is constructed as the image of R, so we obtain a commuting triangle

$$\begin{array}{ccc} A & \xrightarrow{q_R} & A/R \\ & {\scriptstyle R}\searrow \quad \swarrow{\scriptstyle i_R} & \\ & \mathsf{Prop}_{\mathcal{U}}^A, & \end{array}$$

and the embedding $i_R : A/R \to \mathsf{Prop}_{\mathcal{U}}^A$ satisfies the universal property of the image of R. This universal property is, however, not the usual universal property of the quotient.

Definition 18.2.1 Consider a map $q : A \to B$ into a set B satisfying the property that

$$R(x, y) \to (f(x) = f(y))$$

for all $x, y : A$. We say that $q : A \to B$ is a **set quotient** of R, or that q satisfies the **universal property of the set quotient by** R, if for every map $f : A \to X$ into a set X such that $f(x) = f(y)$ whenever $R(x, y)$ holds, there is a unique extension

$$\begin{array}{ccc} A & & \\ {\scriptstyle q}\downarrow & \searrow{\scriptstyle f} & \\ B & \dashrightarrow & X. \end{array}$$

Remark 18.2.2 Formally, we express the universal property of the quotient by R as follows. Consider a map $q : A \to B$ that satisfies the property that

$$H : \prod_{(x,y:A)} R(x,y) \to (f(x) = f(y)).$$

Then there is for any set X a map

$$q^* : (B \to X) \to \left(\sum_{(f:A\to X)} \prod_{(x,y:A)} R(x,y) \to (f(x) = f(y))\right).$$

This map takes a function $h : B \to X$ to the pair

$$q^*(h) :\equiv (h \circ q, \lambda x.\, \lambda y.\, \lambda r.\, \mathsf{ap}_h(H_{x,y}(r))).$$

The universal property of the set quotient of R asserts that the map q^* is an equivalence for every set X. It is important to note that the universal property of set quotients is formulated with respect to sets.

Theorem 18.2.3 *Consider a type A and a universe $\mathcal{U}$ containing A. Furthermore, let $R : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$ be an equivalence relation, and consider a map $q : A \to B$ into a set B, not necessarily in $\mathcal{U}$. Then the following are equivalent.*

(i) *The map q satisfies the property that*

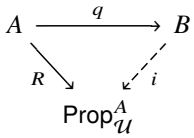
$$q(x) = q(y)$$

for every $x, y : A$ for which $R(x,y)$ holds, and moreover q satisfies the universal property of the set quotient of R.

(ii) *The map q is surjective and **effective**, which means that for each $x, y : A$ we have an equivalence*

$$(q(x) = q(y)) \simeq R(x,y).$$

(iii) *The map $R : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$ extends along q to an embedding*



and the embedding i satisfies the universal property of the image inclusion of R.

In the formulation of Theorem 18.2.3, we don't assume that B is in the same universe as A and R, because we want to apply it to $B :\equiv \mathrm{im}(R)$. As we will see below, this extra generality only affects the proof that (ii) implies (iii).

Proof We first show that (ii) is equivalent to (iii), since this is the easiest part. After that, we will show that (i) is equivalent to (ii).

Assume that (iii) holds. Then q is surjective by Theorem 15.2.5. Moreover, we have

$$\begin{aligned} R(x,y) &\simeq R(x) = R(y) \\ &\simeq i(q(x)) = i(q(y)) \\ &\simeq q(x) = q(y) \end{aligned}$$

In this calculation, the first equivalence holds by Corollary 18.1.5; the second equivalence holds since we have a homotopy $R \sim i \circ q$; and the third equivalence holds since i is an embedding. This completes the proof that (iii) implies (ii).

Next, we show that (ii) implies (iii). First, we want to define a map

$$i : B \to \mathsf{Prop}_{\mathcal{U}}^{A}.$$

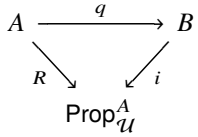
We would like to define $i(b,a) :\equiv (b = q(a))$. This direct definition does not go through, however, because the type B is not assumed to be in $\mathcal{U}$. Nevertheless, observe that by the assumption that q is surjective and effective, the type B is locally $\mathcal{U}$-small. Indeed, $\mathsf{is\text{-}small}_U(X)$ is a proposition for any type X. By surjectiveness of q, it therefore suffices to show that $q(a) = q(a')$ is $\mathcal{U}$-small for each $a, a' : A$. This follows by the assumption that q is effective. In particular, the identity type type $b = q(a)$ is a $\mathcal{U}$-small proposition, for every $b : B$ and $a : A$. Let us write $s(b,a)$ for the element of type $\mathsf{is\text{-}small}_{\mathcal{U}}(b = q(a))$. Consider a universe $\mathcal{V}$ containing B. Then we can define a map

$$j : B \to (A \to \textstyle\sum_{(P:\mathsf{Prop}_{\mathcal{V}})} \mathsf{is\text{-}small}_{\mathcal{U}}(P))$$

by $j(b,a) :\equiv (b = q(a), s(b,a))$, and now we obtain i from j by defining

$$i(b,a) :\equiv \mathsf{pr}_1(\mathsf{pr}_2(j(b,a))).$$

Note that $i(b,a) \simeq (b = q(a))$ for all $b : B$ and $a : A$. Then the triangle



commutes, since we have an equivalence

$$i(q(a), a') \simeq (q(a) = q(a')) \simeq R(a, a')$$

for each $a, a' : A$. To show that i is an embedding, it suffices to show that i is

injective, i.e., that

$$\prod_{(b,b':B)} (i(b) = i(b')) \to (b = b')$$

Note that this is a property, and that q is assumed to be surjective. Hence by Proposition 15.2.3 it is equivalent to show that

$$\prod_{(a,a':A)} (i(q(a)) = i(q(a'))) \to (q(a) = q(a')).$$

Since $R \sim i \circ q$, and $q(a) = q(a')$ is assumed to be equivalent to $R(a, a')$, it suffices to show that

$$\prod_{(a,a':A)} (R(a) = R(a')) \to R(a, a'),$$

which follows directly from Corollary 18.1.5. Thus we have shown that the factorization $R \sim i \circ q$ factors R as a surjective map followed by an injective map. We conclude by Theorem 15.2.5 that the embedding i satisfies the universal property of the image factorization of R, which finishes the proof that (ii) implies (iii).

Now we show that (i) implies (ii). To see that q is surjective if it satisfies the assumptions in (i), consider the image factorization

$$\begin{array}{ccc} A & \xrightarrow{q_q} & \mathrm{im}(q) \\ & {\scriptstyle q}\searrow \quad \swarrow{\scriptstyle i_q} & \\ & B. & \end{array}$$

We claim that the map i_q has a section. To see this, we first note that we have

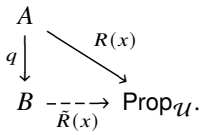
$$q_q(x) = q_q(y)$$

for any $x, y : A$ satisfying $R(x, y)$, because if $R(x, y)$ holds, then $q(x) = q(y)$ and hence $i_q(q_q(x)) = i_q(q_q(y))$ holds and i_q is an embedding. Since $\mathrm{im}(q)$ is a set, we may apply the universal property of q and we obtain a unique extension of q_q along q

$$\begin{array}{ccc} A & & \\ {\scriptstyle q}\downarrow & \searrow^{q_q} & \\ B & \xrightarrow[h]{} & \mathrm{im}(q). \end{array}$$

Now we observe that the composite $i_q \circ h$ is an extension of q along q, so it must be the identity function by uniqueness. Thus we have established that h is a section of i_q. Now it follows from the fact that i_q is an embedding with a section, that i_q is an equivalence. We conclude that q is surjective, because q is the composite $i_q \circ q_q$ of a surjective map followed by an equivalence.

Now we have to show that $q(x) = q(y)$ is equivalent to $R(x, y)$. We first apply the universal property of q to obtain for each $x : A$ an extension of $R(x)$ along q



Since the triangle commutes, we have an equivalence $\tilde{R}(x, q(x')) \simeq R(x, x')$ for each $x' : A$. Now we apply Theorem 11.2.2 to see that the canonical family of maps

$$\prod_{(y:B)} (q(x) = y) \to \tilde{R}(x, y)$$

is a family of equivalences. Thus, we need to show that the type $\sum_{(y:B)} \tilde{R}(x, y)$ is contractible. For the center of contraction, note that we have $q(x) : B$, and the type $\tilde{R}(x, q(x))$ is equivalent to the type $R(x, x)$, which is inhabited by reflexivity of R. To construct the contraction, it suffices to show that

$$\prod_{(y:B)} \tilde{R}(x, y) \to (q(x) = y).$$

Since this is a property, and since we have already shown that q is a surjective map, we may apply Proposition 15.2.3, by which it suffices to show that

$$\prod_{(x':A)} \tilde{R}(x, q(x')) \to (q(x) = q(x')).$$

Since $\tilde{R}(x, q(x')) \simeq R(x, x')$, this is immediate from our assumption on q. Thus we obtain the contraction, and we conclude that we have an equivalence $\tilde{R}(x, y) \simeq (q(x) = y)$ for each $y : B$. It follows that we have an equivalence

$$R(x, y) \simeq (q(x) = q(y))$$

for each $x, y : A$, which completes the proof that (i) implies (ii).

It remains to show that (ii) implies (i). Assume (ii), and let $f : A \to X$ be a map into a set X, satisfying the property that

$$\prod_{(a,a':A)} R(a, a') \to (f(a) = f(a')).$$

Our goal is to show that the type of extensions of f along q is contractible. By Exercise 17.4 it follows that there is at most one such an extension, so it suffices to construct one.

In order to construct an extension, we will construct for every $b : B$ a term $x : X$ satisfying the property

$$P(x) :\equiv \exists_{(a:A)} (f(a) = x) \wedge (q(a) = b).$$

Before we make this construction, we first observe that there is at most one such

x, i.e., that the type of $x : X$ satisfying $P(x)$ is in fact a proposition. To see this, we need to show that $x = x'$ for any $x, x' : X$ satisfying $P(x)$ and $P(x')$. Since X is assumed to be a set, our goal of showing that $x = x'$ is a property. Therefore we may assume that we have $a, a' : A$ satisfying

$$\begin{aligned} f(a) &= x & \qquad q(a) &= b \\ f(a') &= x' & \qquad q(a') &= b. \end{aligned}$$

It follows from these assumptions that $q(a) = q(a')$, and hence that $R(a, a')$ holds. This in turn implies that $f(a) = f(a')$, and hence that $x = x'$.

Now let $b : B$. Our goal is to construct an $x : X$ that satisfies the property

$$\exists_{(a:A)} (f(a) = x) \wedge (q(a) = b).$$

Since q is assumed to be surjective, we have $\|\mathsf{fib}_q(b)\|$. Moreover, since we have shown that at most one $x : X$ exists with the asserted property, we get to assume that we have $a : A$ satisfying $q(a) = b$. Now we see that $x :\equiv f(a)$ satisfies the desired property.

Thus, we obtain a function $h : B \to X$ satisfying the property that for all $b : B$ there exists an $a : A$ such that

$$f(a) = h(b) \qquad \text{and} \qquad q(a) = b.$$

In particular, it follows that $h(q(a)) = f(a)$ for all $a : A$, which completes the proof that (ii) implies (i). □

Corollary 18.2.4 *Consider an equivalence relation R over a type A. Then the quotient map*

$$q : A \to A/R$$

is surjective and effective, and it satisfies the universal property of the set quotient.

18.3 Set truncations

An instance of set quotients is the notion of set truncation. Analogous to the propositional truncation, the set truncation of a type A is a map $\eta : A \to \|A\|_0$ into a set $\|A\|_0$ such that any map $f : A \to X$ into a set X extends uniquely along η:

$$\begin{array}{ccc} A & & \\ {\scriptstyle\eta}\downarrow & \searrow^{f} & \\ \|A\|_0 & \dashrightarrow & X. \end{array}$$

In other words, the set truncation $\eta : A \to \|A\|_0$ is the universal way of mapping A into a set. We first specify what it means for a map $f : A \to B$ into a set B to be a set truncation of A.

Definition 18.3.1 We say that a map $f : A \to B$ into a set B is a **set truncation** if the precomposition function

$$- \circ f : (B \to X) \to (A \to X)$$

is an equivalence for every set X.

In the following theorem we prove several conditions that are equivalent to being a set truncation.

Theorem 18.3.2 *Consider a map $f : A \to B$ into a set B. Then the following are equivalent:*

(i) *The map f is a set truncation.*
(ii) *The map f satisfies the dependent universal property of the set truncation: For every family X of sets over B, the precomposition function*

$$- \circ f : \left(\prod_{(b:B)} X(b)\right) \to \left(\prod_{(a:A)} X(f(a))\right)$$

is an equivalence.
(iii) *The map f is a set quotient with respect to the equivalence relation $x, y \mapsto \|x = y\|$.*

Proof The fact that (ii) implies (i) is immediate. Moreover, the fact that (i) is equivalent to (iii) follows from the fact that any map $h : A \to X$ into a set X comes equipped with a function

$$\|x = y\| \to (h(x) = h(y))$$

for every $x, y : A$.

It remains to prove that (i) implies (ii). Consider a family X of sets over B, and consider the commuting square

$$\begin{array}{ccc} \sum_{(g:B\to B)} \prod_{(b:B)} X(g(b)) & \xrightarrow{(g,s)\mapsto(g\circ f, s\circ f)} & \sum_{(h:A\to B)} \prod_{(a:A)} X(h(a)) \\ \simeq\downarrow & & \downarrow\simeq \\ (B \to \sum_{(b:B)} X(b)) & \xrightarrow[-\circ f]{} & (A \to \sum_{(b:B)} X(b)) \end{array}$$

The side maps are equivalences by the distributivity of Π over Σ, and the bottom map is an equivalence by the assumption that f is a set truncation. Therefore it follows that the top map is an equivalence. Furthermore, note that the map

$$- \circ f : (B \to B) \to (A \to B)$$

is an equivalence by the assumption that f is a set truncation. Therefore it follows from Theorem 11.1.6 that the map

$$- \circ f : \left(\prod_{(b:B)} X(g(b))\right) \to \left(\prod_{(a:A)} X(g(f(a)))\right)$$

is an equivalence for every $g : B \to B$. Now we take $g :\equiv \mathsf{id}$ to complete the proof that (i) implies (ii). □

Corollary 18.3.3 *On any universe $\mathcal{U}$, there is an operation $\|-\|_0 : \mathcal{U} \to \sum_{(X:\mathcal{U})} \mathsf{is\text{-}set}(X)$ such that every type A in $\mathcal{U}$ comes equipped with a map*

$$\eta : A \to \|A\|_0$$

*that satisfies the universal property of the set truncation. The set $\|A\|_0$ is called the **set truncation** of A.*

Proof By Theorem 18.3.2 it follows that a map $f : A \to B$ into a set B is a set truncation if and only if it is a quotient map with respect to the equivalence relation $x, y \mapsto \|x = y\|$. Given a type A in $\mathcal{U}$, the quotient of A by $x, y \mapsto \|x = y\|$ is again a type in $\mathcal{U}$ by Axiom 18.1.13. □

Corollary 18.3.4 *The set truncation $\eta : A \to \|A\|_0$ is surjective and effective with respect to the equivalence relation $x, y \mapsto \|x = y\|$, i.e., we have an equivalence*

$$(\eta(x) = \eta(y)) \simeq \|x = y\|$$

for each $x, y : A$.

By this corollary, we may think of the set truncation $\|A\|_0$ of A as the set of connected components of A. Indeed, if we have an unspecified identification $\|x = y\|$ in A, then we think of x and y as being in the same connected component. For example, any k-element set is a type that is in the same connected component of $\mathcal{U}$ as the type Fin_k.

There are truncation operations for every truncation level. That is, we can define for every type A a map $\eta : A \to \|A\|_k$ such that the map

$$- \circ \eta : (\|A\|_k \to X) \to (A \to X)$$

is an equivalence for every k-truncated type X. We will study k-truncations in more detail in Section 31, when we have more types of higher truncation levels available and when we have more things to say about them.

18.4 Unique representatives of equivalence classes

Definition 18.4.1 Consider an equivalence relation R on a type A, and let P be a family of types over A. We say that P is a **choice of (unique) representatives** for R if P comes equipped with an element of type

$$\mathsf{is\text{-}choice\text{-}of\text{-}reps}(P) :\equiv \prod_{(x:A)} \mathsf{is\text{-}contr}\left(\sum_{(y:A)} P(y) \times R(x,y)\right).$$

Theorem 18.4.2 *Consider an equivalence relation R on a type A, and let P be a family of types over A. If P is a choice of unique representatives, then the map*

$$A \to \sum_{(x:A)} P(x)$$

obtained from $\mathsf{is\text{-}choice\text{-}of\text{-}reps}(P)$ *satisfies the universal property of the set quotient of A by R.*

Exercises

18.1 Consider a map $f : A \to B$ into a set B, and let $R : A \to (A \to \mathsf{Prop}_{\mathcal{U}})$ be the equivalence relation given by

$$R(x,y) :\equiv f(x) = f(y).$$

Show that the map $q_f : A \to \mathrm{im}(f)$ satisfies the universal property of the set quotient of R.

18.2 Show that the set truncation of a loop space is a group.

18.3 Recall that a normal subgroup H of a group G is a subgroup of G such that ghg^{-1} is in H for every $h : H$ and $g : G$. Given a normal subgroup H of G, we write G/H for the quotient of G by the equivalence relation where $g \sim g'$ if and only if there is a $h : H$ such that $gh = g'$. Show that G/H is again a group.

18.4 (a) Show that any proposition is locally small.
(b) Show that any essentially small type is locally small.
(c) Show that the function type $A \to X$ is locally small whenever A is essentially small and X is locally small.

18.5 Let $f : A \to B$ be a map. Show that the following are equivalent:

(i) The map f is **locally small** in the sense that for every $x, y : A$, the action on paths of f

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

is an essentially small map.

(ii) The diagonal δ_f of f as defined in Exercise 22.2 is classified by the universal fibration.

18.6 Consider an abelian group A. Define an operation

$$\prod_{(X:\mathbb{F})} A^X \to A$$

that extends the (binary) group operation to the finite un-ordered n-tuples of elements in A.

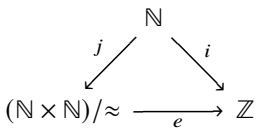
18.7 Consider the relation

$$(m, n) \approx (m', n')$$

on the type $\mathbb{N} \times \mathbb{N}$.

(a) Show that this relation is an equivalence relation.

(b) Construct an equivalence $e : ((\mathbb{N} \times \mathbb{N})/\approx) \simeq \mathbb{Z}$, such that the triangle



commutes, where the map j is given by $j(n) :\equiv [(n, 0_{\mathbb{N}})]_{\approx}$.

18.8 Recall that the group $\mathbb{Z}_n$ was constructed in **??**. Construct a group isomorphism

$$\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}/n,$$

where the group on the left-hand side is defined as the quotient of $\mathbb{Z}$ by the normal subgroup $n\mathbb{Z}$.

18.9 Consider a type A, and let $\mathcal{U}$ be an arbitrary universe. Define the type of $\mathcal{U}$-small **partitions** on A to be the type of $Q : (A \to \mathsf{Prop}_{\mathcal{U}}) \to \mathsf{Prop}_{\mathcal{U}}$ equipped with a term of type

$$\mathsf{is\text{-}partition}(Q) :\equiv \prod_{(a:A)} \mathsf{is\text{-}contr}\left(\sum_{(P:A\to\mathsf{Prop}_{\mathcal{U}})} Q(P) \times P(a)\right).$$

We will write $\mathsf{Partition}_{\mathcal{U}}(A)$ for the type of all $\mathcal{U}$-small partitions on A. When Q is a partition on A, then we write Q_x for the unique subset $P : A \to \mathsf{Prop}_{\mathcal{U}}$ such that $Q(P)$ and $P(x)$ both hold. Furthermore, given a partition Q on A, define the type

$$A/Q :\equiv \sum_{(P:A\to\mathsf{Prop}_{\mathcal{U}})} Q(P),$$

and define the map $q : A \to A/Q$ by $q(x) :\equiv Q_x$.

(a) For any $\mathcal{U}$-small equivalence relation R on A, define $Q_R : (A \to \mathsf{Prop}_{\mathcal{U}}) \to \mathsf{Prop}_{\mathcal{U}}$ by

$$Q_R(P) :\equiv \exists_{(x:A)} \forall_{(y:A)} P(y) \leftrightarrow R(x, y).$$

As a special case, we define $K_f :\equiv Q_{f(-)=f(-)}$ for any map $f : A \to B$ into a set B. Show that Q_R is a partition on A, and that we therefore obtain a map

$$\mathsf{Eq\text{-}Rel}_{\mathcal{U}}(A) \to \mathsf{Partition}_{\mathcal{U}}(A).$$

(b) Show that the map

$$\mathsf{Eq\text{-}Rel}_{\mathcal{U}}(A) \to \mathsf{Partition}_{\mathcal{U}}(A)$$

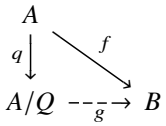
is an equivalence.

(c) Define the ordering relation $\leq$ on the partitions on A by

$$(Q \leq Q') :\equiv \forall_{(x:A)} Q_x \subseteq Q'_x$$

Show that this relation satisfies the axioms of a poset.

(d) Show that the map $q : A \to A/Q$ satisfies the following universal property: For any map $f : A \to B$ into a set B such that $Q \leq K_f$, there is a unique map $g : A/Q \to B$ such that the triangle



commutes.

(e) Show that the map $q : A \to A/Q$ satisfies the universal property of the quotient A/R_Q, where R_Q is the equivalence relation on A given by

$$R_Q(x, y) :\equiv \exists_{(P:A\to\mathsf{Prop}_{\mathcal{U}})} Q(P) \times P(x) \times P(y).$$

18.10 (a) Consider a proposition P, and define the relation $\sim_P$ on bool by

$$(\mathsf{true} \sim_P \mathsf{true}) :\equiv \mathbf{1} \qquad (\mathsf{true} \sim_P \mathsf{false}) :\equiv P$$
$$(\mathsf{false} \sim_P (\mathsf{true}) :\equiv P \qquad (\mathsf{false} \sim_P \mathsf{false}) :\equiv \mathbf{1}$$

Show that $\sim_P$ is an equivalence relation.

(b) Consider a universe $\mathcal{U}$ containing the proposition P. Construct an embedding $\mathsf{bool}/\sim_P \hookrightarrow \mathsf{Prop}_{\mathcal{U}}$.

(c) Show that the law of excluded middle holds if and only if every set has decidable equality.

(d) Use the quotient $\mathsf{bool}/\sim_P$ to show that the axiom of choice implies the law of excluded middle.

18.11 A map $f : A \to B$ is called **weakly path-constant** if it comes equipped with an element of type

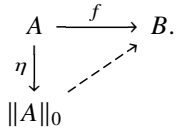
$$\mathsf{is\text{-}weakly\text{-}path\text{-}constant}(f) : \prod_{(x,y:A)} \prod_{(p,q:x=y)} \mathsf{ap}_f(p) = \mathsf{ap}_f(q).$$

In other words, f is weakly path-constant if for each $x, y : A$ the map $\mathsf{ap}_f : (x = y) \to (f(x) = f(y))$ is weakly constant in the sense of Definition 14.4.3.

(a) Show that every map $\|A\|_0 \to B$ is weakly path-constant. Use this to obtain a map

$$\alpha : \Big(\|A\|_0 \to B\Big) \to \Big(\textstyle\sum_{(f:A\to B)} \mathsf{is\text{-}weakly\text{-}path\text{-}constant}(f)\Big).$$

(b) Show that if B is a 1-type, then the map α is an equivalence. In other words, show that every weakly path-constant map $f : A \to B$ into a 1-type B has a unique extension



18.12 For any natural number k, define

$$Q_k :\equiv \mathsf{list}(\mathsf{Fin}_k) \times \mathsf{list}(\mathsf{Fin}_k),$$

and let $\sim_k : Q_k \to (Q_k \to \mathsf{Prop}_{\mathcal{U}})$ be the equivalence relation generated by

$$(\mathsf{cons}(0, x), y) \sim_{k+1} (x, y)$$
$$(x, (\mathsf{concat\text{-}list}(y, 0))) \sim_{k+1} (x, y).$$

Show that $\mathbb{Q}_{\geq 0} \simeq Q_{k+1}/{\sim_{k+1}}$.

18.13 Consider two (unrelated) univalent universes $\mathcal{U}$ and $\mathcal{V}$.

(a) Show that the type of essentially $\mathcal{V}$-small types in $\mathcal{U}$ is equivalent to the type of essentially $\mathcal{U}$-small types in $\mathcal{V}$.

(b) Suppose that each type in $\mathcal{U}$ is essentially $\mathcal{V}$-small. Show that the type $\mathcal{U}$ itself is $\mathcal{V}^+$-small.

18.14 Consider a type A and a universe $\mathcal{U}$ containing A. Let

$$\tau : A \to ((A \to \mathsf{Prop}_{\mathcal{U}}) \to \mathsf{Prop}_{\mathcal{U}})$$

be the map defined by $\tau(a) :\equiv \lambda f.\, f(a)$. Show that the map

$$q_\tau : A \to \mathrm{im}(\tau)$$

obtained from the image factorization of A is a set truncation of A.

18.15 In this exercise we extend the definition of the binomial types to $\|\mathcal{U}\|_0$ as follows: For a type $A : \mathcal{U}$ and $k : \|\mathcal{U}\|_0$, we define

$$\binom{A}{k} :\equiv \sum_{((X,p):\mathsf{fib}_\eta(k))} X \hookrightarrow_{\mathsf{d}} A.$$

Furthermore, for $(X, i) : \binom{A}{k}$, define

$$A \setminus X :\equiv \sum_{(a:A)} \neg(\mathsf{fib}_i(a)).$$
$$i :\equiv \mathsf{pr}_1.$$

Now consider a type X and two type families A and B over X, and let $\mathcal{U}$ be a universe containing X, A, and B. Show that the type $\prod_{(x:X)} A(x) + B(x)$ is equivalent to the type

$$\sum_{(k:\|\mathcal{U}\|_0)} \sum_{((Y,i):\binom{X}{k})} \left(\prod_{(y:Y)} A(i(y))\right) \times \left(\prod_{(y:Y^{\mathsf{c}})} B(i^{\mathsf{c}}(y))\right).$$

18.16 For any type X, construct an equivalence

$$\|\mathsf{list}(X)\|_0 \simeq \mathsf{list}\|X\|_0.$$

18.17 Consider a type A equipped with an element $a : A$. Show that the following are equivalent:

(i) The type A is **connected** in the sense that $\|A\|_0$ is contractible.
(ii) There is an element of type $\|a = x\|$ for any $x : A$.
(iii) For any family B over A, the fiber inclusion

$$i_a : B(a) \to \sum_{(x:A)} B(x)$$

defined in Exercise 12.13 is surjective.

18.18 Consider a type A, and suppose that $\|A\|_0$ is a finite type with k elements. Show that there exists a map $f : \mathsf{Fin}_k \to A$ such that $\eta \circ f$ is an equivalence, i.e., prove the proposition

$$\exists_{(f:\mathsf{Fin}_k \to A)} \mathsf{is\text{-}equiv}(\eta \circ f).$$

18.19 Consider a map $f : A \to B$.

(a) Show that if f is injective, then $\|f\|_0 : \|A\|_0 \to \|B\|_0$ is injective.
(b) Show that the following are equivalent

(i) The map f is surjective.
(ii) the map $\|f\|_0 : \|A\|_0 \to \|B\|_0$ is surjective.

(c) Construct a map $h : \mathrm{im}(f) \to \mathrm{im}\,\|f\|_0$ such that the squares

$$\begin{array}{ccccc} A & \xrightarrow{q_f} & \mathrm{im}(f) & \xrightarrow{i_f} & B \\ {\scriptstyle\eta}\downarrow & & {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle\eta} \\ \|A\|_0 & \xrightarrow[q_{\|f\|_0}]{} & \mathrm{im}\,\|f\|_0 & \xrightarrow[i_{\|f\|_0}]{} & \|B\|_0 \end{array}$$

commute, and show that h is a set truncation of $\mathrm{im}(f)$.

19 Groups in univalent mathematics

In this section we demonstrate a typical way to use the univalence axiom, showing that isomorphic groups can be identified. For example, by the third isomorphism theorem, which gives an isomorphism

$$(G/N)/(K/N) \cong (G/K)$$

for any sequence $N \trianglelefteq K \trianglelefteq G$ of normal subgroups of G, we will be able to identify $(G/N)/(K/N)$ with G/K. Identifying such "canonically" isomorphic groups is common practice. This is an instance of the *structure identity principle*, which is described in more detail in section 9.8 of [5].

In order to show that isomorphic groups can be identified, it has to be part of the definition of a group that its underlying type is a set. This is an important observation: in many branches of algebra the objects of study are *set-level* structures[2].

19.1 The type of all groups

We introduce the type of groups in two stages: first we introduce the type of *semi-groups*, and then we introduce groups as semi-groups that possess further structure. It will turn out that this further structure is in fact a property, and this fact will help us to prove that isomorphic groups are equal.

Definition 19.1.1 A **semi-group** consists of a set G equipped with a term of type $\mathsf{has\text{-}associative\text{-}mul}(G)$, which is the type of pairs $(\mu_G, \mathsf{assoc}_G)$ consisting of a binary operation

$$\mu_G : G \to (G \to G)$$

[2] A notable exception is that of categories, which are objects at truncation level 1, i.e., at the level of *groupoids*. We will briefly introduce categories in **??**. For more about categories we recommend Chapter 9 of [5].

and a homotopy

$$\mathsf{assoc}_G : \prod_{(x,y,z:G)} \mu_G(\mu_G(x,y),z) = \mu_G(x,\mu_G(y,z)).$$

We write Semi-Group for the type of all semi-groups in $\mathcal{U}$.

Definition 19.1.2 A semi-group G is said to be **unital** if it comes equipped with a **unit** $e_G : G$ that satisfies the left and right unit laws

$$\mathsf{left\text{-}unit}_G : \prod_{(y:G)} \mu_G(e_G, y) = y$$
$$\mathsf{right\text{-}unit}_G : \prod_{(x:G)} \mu_G(x, e_G) = x.$$

We write $\mathsf{is\text{-}unital}(G)$ for the type of such triples $(e_G, \mathsf{left\text{-}unit}_G, \mathsf{right\text{-}unit}_G)$. Unital semi-groups are also called **monoids**.

The unit of a semi-group is of course unique once it exists. In univalent mathematics we express this fact by asserting that the type $\mathsf{is\text{-}unital}(G)$ is a proposition for each semi-group G. In other words, being unital is a *property* of semi-groups rather than structure on it. This is typical for univalent mathematics: we express that a structure is a property by proving that this structure is a proposition.

Lemma 19.1.3 *For a semi-group G the type $\mathsf{is\text{-}unital}(G)$ is a proposition.*

Proof Let G be a semi-group. Note that since G is a set, it follows that 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:

$$e = \mu_G(e, e') = e'. \qquad \square$$

Definition 19.1.4 Let G be a unital semi-group. We say that G **has inverses** if it comes equipped with an operation $x \mapsto x^{-1}$ of type $G \to G$, satisfying the left and right inverse laws

$$\mathsf{left\text{-}inv}_G : \prod_{(x:G)} \mu_G(x^{-1}, x) = e_G$$
$$\mathsf{right\text{-}inv}_G : \prod_{(x:G)} \mu_G(x, x^{-1}) = e_G.$$

We write $\mathsf{is\text{-}group'}(G, e)$ for the type of such triples $((-)^{-1}, \mathsf{left\text{-}inv}_G, \mathsf{right\text{-}inv}_G)$, and we write

$$\mathsf{is\text{-}group}(G) :\equiv \sum_{(e:\mathsf{is\text{-}unital}(G))} \mathsf{is\text{-}group'}(G, e)$$

A **group** is a unital semi-group with inverses. We write Group for the type of all groups in $\mathcal{U}$.

Lemma 19.1.5 *For any semi-group G the type $\mathsf{is\text{-}group}(G)$ is a proposition.*

Proof We have already seen that the type $\mathsf{is\text{-}unital}(G)$ is a proposition. Therefore it suffices to show that the type $\mathsf{is\text{-}group}'(G, e)$ is a proposition for any $e : \mathsf{is\text{-}unital}(G)$.

Since a semi-group G is assumed to be a set, we note that the types of the inverse laws are propositions. Therefore it suffices to show that any two inverse operations satisfying the inverse laws are homotopic.

Let $x \mapsto x^{-1}$ and $x \mapsto \bar{x}^{-1}$ be two inverse operations on a unital semi-group G, both satisfying the inverse laws. Then we have the following identifications

$$\begin{aligned} x^{-1} &= \mu_G(e_G, x^{-1}) \\ &= \mu_G(\mu_G(\bar{x}^{-1}, x), x^{-1}) \\ &= \mu_G(\bar{x}^{-1}, \mu_G(x, x^{-1})) \\ &= \mu_G(\bar{x}^{-1}, e_G) \\ &= \bar{x}^{-1} \end{aligned}$$

for any $x : G$. Thus the two inverses of x are the same, so the claim follows. □

Example 19.1.6 An important class of examples consists of **loop spaces** $x = x$ of a 1-type X, for any $x : X$. We will write $\Omega(X, x)$ for the loop space of X at x. Since X is assumed to be a 1-type, it follows that the type $\Omega(X, x)$ is a set. Then we have

$$\begin{aligned} \mathsf{refl} &: \Omega(X, x) \\ \mathsf{inv} &: \Omega(X, x) \to \Omega(X, x) \\ \mathsf{concat} &: \Omega(X, x) \to (\Omega(X, x) \to \Omega(X, x)), \end{aligned}$$

and these operations satisfy the group laws, since the group laws are just a special case of the groupoid laws for identity types, constructed in Section 5.2.

Example 19.1.7 The type $\mathbb{Z}$ of integers can be given the structure of a group, with the group operation being addition. The fact that $\mathbb{Z}$ is a set follows from Theorem 11.3.1 and Exercise 12.4. The group laws were shown in Exercise 5.7.

Example 19.1.8 Our last class of examples consists of the **automorphism groups** on sets. Given a set X, we define

$$\mathrm{Aut}(X) :\equiv (X \simeq X).$$

The group operation of $\mathrm{Aut}(X)$ is just composition of equivalences, and the unit of the group is the identity function. Note however, that although function composition is strictly associative and satisfies the unit laws strictly, composition of equivalences only satisfies the group laws up to identification because the proof that composites are equivalences is carried along.

Important special cases of the automorphism groups are the **symmetric groups**

$$\mathcal{S}_n :\equiv \mathsf{Aut}(\mathsf{Fin}_n).$$

19.2 Group homomorphisms

Definition 19.2.1 Let G and H be semi-groups. A **homomorphism** of semi-groups from G to H is a pair (f, μ_f) consisting of a function $f : G \to H$ 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 operation of G. We will write

$$\mathsf{hom}(G, H)$$

for the type of all semi-group homomorphisms from G to H.

Remark 19.2.2 Since it is a property for a function to preserve the multiplication of a semi-group, it follows easily that equality of semi-group homomorphisms is equivalent to the type of homotopies between their underlying functions. In particular, it follows that the type of homomorphisms of semi-groups is a set.

Remark 19.2.3 The **identity homomorphism** on a semi-group G is defined to be the pair consisting of

$$\begin{aligned} \mathsf{id} &: G \to G \\ \lambda x.\, \lambda y.\, \mathsf{refl} &: \prod_{(x,y:G)} \mu_G(x,y) = \mu_G(x,y). \end{aligned}$$

Let $f : G \to H$ and $g : H \to K$ be semi-group homomorphisms. Then the composite function $g \circ f : G \to K$ is also a semi-group homomorphism, since we have the identifications

$$g(f(\mu_G(x,y))) \jdeq g(\mu_H(f(x), f(y))) \jdeq \mu_K(g(f(x)), g(f(y))).$$

Since the identity type of semi-group homomorphisms is equivalent to the type of homotopies between semi-group homomorphisms it is easy to see that semi-group homomorphisms satisfy the laws of a category, i.e., that we have the identifications

$$\begin{aligned} \mathsf{id} \circ f &= f \\ g \circ \mathsf{id} &= g \\ (h \circ g) \circ f &= h \circ (g \circ f) \end{aligned}$$

for any composable semi-group homomorphisms f, g, and h. Note, however

that these equalities are not expected to hold judgmentally, since preservation of the semi-group operation is part of the data of a semi-group homomorphism.

Definition 19.2.4 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

$$\mathsf{hom}(G, H)$$

for the type of all group homomorphisms from G to H.

Remark 19.2.5 Since a group homomorphism is just a semi-group homomorphism between the underlying semi-groups, we immediately obtain the identity homomorphism, composition, and the category laws are satisfied.

19.3 Isomorphic groups are equal

Definition 19.3.1 Let $h : \mathsf{hom}(G, H)$ be a homomorphism of semi-groups. Then h is said to be an **isomorphism** if it comes equipped with a term of type $\mathsf{is\text{-}iso}(h)$, consisting of triples (h^{-1}, p, q) consisting of a homomorphism $h^{-1} : \mathsf{hom}(H, G)$ of semi-groups and identifications

$$p : h^{-1} \circ h = \mathsf{id}_G \qquad \text{and} \qquad q : h \circ h^{-1} = \mathsf{id}_H$$

witnessing that h^{-1} satisfies the inverse lawsWe write $G \cong H$ for the type of all isomorphisms of semi-groups from G to H, i.e.,

$$G \cong H :\equiv \textstyle\sum_{(h:\mathsf{hom}(G,H))} \sum_{(k:\mathsf{hom}(H,G))} (k \circ h = \mathsf{id}_G) \times (h \circ k = \mathsf{id}_H).$$

If f is an isomorphism, then its inverse is unique. In other words, being an isomorphism is a property.

Lemma 19.3.2 *For any semi-group homomorphism $h : \mathsf{hom}(G, H)$, the type*

$$\mathsf{is\text{-}iso}(h)$$

is a proposition. It follows that the type $G \cong H$ is a set for any two semi-groups G and H.

Proof Let k and k' be two inverses of h. In Remark 19.2.2 we have observed that the type of semi-group homomorphisms between any two semi-groups is a set. Therefore it follows that the types $h \circ k = \mathsf{id}$ and $k \circ h = \mathsf{id}$ are propositions, so it suffices to check that $k = k'$. In Remark 19.2.2 we also observed that the equality type $k = k'$ is equivalent to the type of homotopies $k \sim k'$ between their underlying functions. We construct a homotopy $k \sim k'$ by the usual argument:

$$k(y) \jdeq k(h(k'(y)) \jdeq k'(y). \qquad \square$$

Lemma 19.3.3 *A semi-group homomorphism $h : \mathsf{hom}(G, H)$ is an isomorphism if and only if its underlying map is an equivalence. Consequently, there is an equivalence*

$$(G \cong H) \simeq \textstyle\sum_{(e:G\simeq H)} \prod_{(x,y:G)} e(\mu_G(x,y)) = \mu_H(e(x), e(y))$$

Proof If $h : \mathsf{hom}(G, H)$ is an isomorphism, then the inverse semi-group homomorphism also provides an inverse of the underlying map of h. Thus we obtain that h is an equivalence. The standard proof showing that if the underlying map $f : G \to H$ of a group homomorphism is invertible then its inverse is again a group homomorphism also works in type theory. □

Definition 19.3.4 Let G and H be a semi-groups. We define the map

$$\mathsf{iso\text{-}eq} : (G = H) \to (G \cong H)$$

by path induction, taking refl to isomorphism id_G.

Theorem 19.3.5 *The map*

$$\mathsf{iso\text{-}eq} : (G = H) \to (G \cong H)$$

is an equivalence for any two semi-groups G and H.

Proof By the fundamental theorem of identity types Theorem 11.2.2 it suffices to show that the total space

$$\textstyle\sum_{(G':\mathsf{Semi\text{-}Group})} G \cong G'$$

is contractible. Since the type of isomorphisms from G to G' is equivalent to the type of equivalences from G to G' it suffices to show that the type

$$\textstyle\sum_{(G':\mathsf{Semi\text{-}Group})} \sum_{(e:G\simeq G')} \prod_{(x,y:G)} e(\mu_G(x,y)) = \mu_{G'}(e(x), e(y)))$$

is contractible[3]. Since $\mathsf{Semi\text{-}Group}$ is the Σ-type

$$\textstyle\sum_{(G':\mathsf{Set})} \mathsf{has\text{-}associative\text{-}mul}(G'),$$

it suffices to show that the types

$$\textstyle\sum_{(G':\mathsf{Set})} G \simeq G'$$
$$\textstyle\sum_{(\mu':\mathsf{has\text{-}associative\text{-}mul}(G))} \prod_{(x,y:G)} \mu_G(x,y) = \mu'(x,y)$$

[3] In order to show that a type of the form

$$\textstyle\sum_{((x,y):\sum_{(x:A)} B(x))} \sum_{(z:C(x))} D(x,y,z)$$

is contractible, a useful strategy is to first show that the type $\sum_{(x:A)} C(x)$ is contractible. Once this is established, say with center of contraction (x_0, z_0), it suffices to show that the type $\sum_{(y:B(x_0))} D(x_0, y, z_0)$ is contractible.

is contractible. The first type is contractible by the univalence axiom. The second type is contractible by function extensionality. □

Corollary 19.3.6 *The type* Semi-Group *is a* 1*-type.*

Proof It is straightforward to see that the type of group isomorphisms $G \cong H$ is a set, for any two groups G and H. □

We now turn to the proof that isomorphic groups are equal. Analogously to the map iso-eq of semi-groups, we have a map iso-eq of groups. Note, however, that the domain of this map is now the identity type $G = H$ of the *groups* G and H, so the maps iso-eq of semi-groups and groups are not exactly the same maps.

Definition 19.3.7 Let G and H be groups. We define the map

$$\mathsf{iso\text{-}eq} : (G = H) \to (G \cong H)$$

by path induction, taking refl to the identity isomorphism $\mathsf{id} : G \cong G$.

Theorem 19.3.8 *For any two groups* G *and* H*, the map*

$$\mathsf{iso\text{-}eq} : (G = H) \to (G \cong H)$$

is an equivalence.

Proof Let G and H be groups, and write UG and UH for their underlying semi-groups, respectively. Then we have a commuting triangle

$$\begin{array}{ccc} (G = H) & \xrightarrow{\mathsf{ap}_{\mathsf{pr}_1}} & (UG = UH) \\ & {\scriptstyle \mathsf{iso\text{-}eq}} \searrow \quad \swarrow {\scriptstyle \mathsf{iso\text{-}eq}} & \\ & (G \cong H) & \end{array}$$

Since being a group is a property of semi-groups it follows that the projection map Group → Semi-Group forgetting the unit and inverses, is an embedding. Thus the top map in this triangle is an equivalence. The map on the right is an equivalence by Theorem 19.3.5, so the claim follows by the 3-for-2 property. □

Corollary 19.3.9 *The type of groups is a* 1*-type.*

19.4 Quotient groups

19.5 Finite groups

Definition 19.5.1 Let H be a subgroup of G, and let $x : G$. Then the left coset Hx at x is the type

$$\textstyle\sum_{(y:G)} \sum_{(z:H)} zx = y$$

Theorem 19.5.2 (Lagrange's theorem) *Consider a decidable subgroup H of a finite group G. Then the order of H divides the order of G.*

Proof Our goal—to show that the order of H divides the order of G—is a proposition, so by the universal property of propositional truncations we may assume that we have $k : \mathbb{N}$ equipped with an equivalence $e : \mathsf{Fin}_k \simeq G$ into the underlying set of the group G. Furthermore, the underlying set of the subgroup H is assumed to be a decidable subset of G, so it follows from ?? that there is a natural number $l : \mathbb{N}$ equipped with an equivalence $f : \mathsf{Fin}_l \simeq H$.

Now consider the type □

19.6 Group actions

19.7 Fermat's little theorem

By Theorem 16.2.2 we can start doing combinatorics in type theory. To get things off the ground, we will give the combinatorial proof of Fermat's little theorem. Fermat's little theorem is the statement that

$$a^p \equiv a \mod p$$

for any prime number p. We will give the combinatorial proof, making repeated use of the notion of equivalences.

The idea of the proof is as follows: The type $(\mathsf{Fin}_a)^p$ is the p-fold product

$$\mathsf{Fin}_a \times \cdots \times \mathsf{Fin}_a.$$

There's an action of the cyclic group on this type, which takes a tuple $(x_1, \ldots, x_p)$ to the tuple $(x_2, \ldots, x_p, x_1)$. The length of the orbit of $(x_1, \ldots, x_p)$ then divides p, and since p is assumed to be prime the length of each orbit is either 1 or p. Note that the length is 1 if and only if the sequence is constant, and there are a such sequences. Since each non-constant sequence has an orbit of length p, it follows that the number of non-constant sequences is p times the number of non-trivial orbits in Fin_a^p. In other words $a^p = pk + a$ for some $k : \mathbb{N}$. There are, however, quite a few facts that we will have to check in order to get this proof to acceptable form in type theory.

Definition 19.7.1 For any $k : \mathbb{N}$ and any type A, we define the type A^k recursively by

$$A^0 :\equiv \mathbf{1}$$
$$A^{k+1} :\equiv A \times A^k.$$

Exercises

19.1 Let X be a set. Show that the map

$$\mathsf{equiv\text{-}eq} : (X = X) \to (X \simeq X)$$

is a group isomorphism.

19.2 Consider a group G. Show that the function

$$\mu_G : G \to (G \simeq G)$$

is an injective group homomorphism.

19.3 Let $f : \mathsf{hom}(G, H)$ be a group homomorphism. Show that f preserves units and inverses, i.e., show that

$$f(e_G) = e_H$$
$$f(x^{-1}) = f(x)^{-1}.$$

19.4 Give a direct proof and a proof using the univalence axiom of the fact that all semi-group isomorphisms between unital semi-groups preserve the unit. Conclude that isomorphic monoids are equal.

19.5 Consider a monoid M with multiplication $\mu : M \to (M \to M)$ and unit e. Write

$$\bar{\mu} :\equiv \mathsf{fold\text{-}list}(e, \mu) : \mathsf{list}(M) \to M$$

for the iterated multiplication operation (see Exercise 4.4). Show that the square

$$\begin{array}{ccc} \mathsf{list}(\mathsf{list}(M)) & \xrightarrow{\mathsf{flatten\text{-}list}(M)} & \mathsf{list}(M) \\ {\scriptstyle \mathsf{list}(\bar{\mu})}\downarrow & & \downarrow{\scriptstyle \bar{\mu}} \\ \mathsf{list}(M) & \xrightarrow[\bar{\mu}]{} & M \end{array}$$

commutes.

19.6 Show that the number of connected components in the type of all groups of order n is as follows, for $n \leq 8$:

order:	1	2	3	4	5	6	7	8
number of groups:	1	1	1	2	1	2	1	5

19.7 Consider a group G. Show that the map

$$\mathsf{Grp}(\mathbb{Z}, G) \to G$$

given by $h \mapsto h(1_{\mathbb{Z}})$, is an equivalence. In other words, the group $\mathbb{Z}$ satisfies the universal property of the **free group on one generator**.

19.8 Consider a set X equipped with an associative binary operation $\mu : X \to (X \to X)$, and suppose that

(i) The type X is inhabited, i.e., $\|X\|$ holds.
(ii) The maps $\mu(x, -)$ and $\mu(-, y)$ are equivalences, for each $x, y : X$.

Show that X is a group.

19.9 Show that the type of 3-element groups is equivalent to the type of 2-element types.

19.10 Consider a proposition P, and let N_P be the subtype of $\mathbb{Z}_2$ given by

$$\mathbb{N}_P(x) :\equiv (x = 0) \vee P.$$

Show that $\mathbb{N}_P$ is a subgroup of $\mathbb{Z}_2$.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

III

Synthetic Homotopy Theory

20 The circle

We have seen inductive types, in which we describe a type by its constructors and an induction principle that allows us to construct sections of dependent types. Inductive types are freely generated by their constructors, which describe how we can construct their elements.

However, many familiar constructions in algebra involve the construction of algebras by generators and relations. For example, the free abelian group with two generators is described as the group with generators x and y, and the relation $xy = yx$.

In this chapter we introduce higher inductive types, where we follow a similar idea: to allow in the specification of inductive types not only *point constructors*, but also *path constructors* that give us relations between the point constructors. The ideas behind the definition of higher inductive types are introduced by studying the simplest non-trivial example: the *circle*.

20.1 The induction principle of the circle

The *circle* is defined as a higher inductive type $\mathbf{S}^1$ that comes equipped with

$$\mathsf{base} : \mathbf{S}^1$$
$$\mathsf{loop} : \mathsf{base} = \mathsf{base}.$$

Just like for ordinary inductive types, the induction principle for higher inductive types provides us with a way of constructing sections of dependent types. However, we need to take the *path constructor* loop into account in the induction principle.

By applying a section $f : \prod_{(x:\mathbf{S}^1)} P(x)$ to the base point of the circle, we obtain an element $f(\mathsf{base}) : P(\mathsf{base})$. Moreover, using the dependent action

on paths of f of Definition 5.4.2 we also obtain for any dependent function $f : \prod_{(x:\mathbf{S}^1)} P(x)$ a path

$$\mathsf{apd}_f(\mathsf{loop}) : \mathsf{tr}_P(\mathsf{loop}, f(\mathsf{base})) = f(\mathsf{base})$$

in the fiber $P(\mathsf{base})$.

Definition 20.1.1 Let P be a type family over the circle. The **dependent action on generators** is the map

$$\mathsf{dgen}_{\mathbf{S}^1} : \left(\prod_{(x:\mathbf{S}^1)} P(x)\right) \to \left(\sum_{(y:P(\mathsf{base}))} \mathsf{tr}_P(\mathsf{loop}, y) = y\right) \tag{20.1.1}$$

given by $\mathsf{dgen}_{\mathbf{S}^1}(f) :\equiv (f(\mathsf{base}), \mathsf{apd}_f(\mathsf{loop}))$.

We now give the full specification of the circle.

Definition 20.1.2 The **circle** is a type $\mathbf{S}^1$ that comes equipped with

$$\begin{aligned} \mathsf{base} &: \mathbf{S}^1 \\ \mathsf{loop} &: \mathsf{base} = \mathsf{base}, \end{aligned}$$

and satisfies the **induction principle of the circle**, which provides for each type family P over $\mathbf{S}^1$ a map

$$\mathsf{ind}_{\mathbf{S}^1} : \left(\sum_{(y:P(\mathsf{base}))} \mathsf{tr}_P(\mathsf{loop}, y) = y\right) \to \left(\prod_{(x:\mathbf{S}^1)} P(x)\right),$$

and a homotopy witnessing that $\mathsf{ind}_{\mathbf{S}^1}$ is a section of $\mathsf{dgen}_{\mathbf{S}^1}$

$$\mathsf{comp}_{\mathbf{S}^1} : \mathsf{dgen}_{\mathbf{S}^1} \circ \mathsf{ind}_{\mathbf{S}^1} \sim \mathsf{id}$$

for the computation rule.

Remark 20.1.3 The type of identifications $(y, p) = (y', p')$ in the type

$$\sum_{(y:P(\mathsf{base}))} \mathsf{tr}_P(\mathsf{loop}, y) = y$$

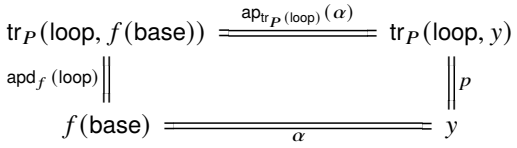
is equivalent to the type of pairs (α, β) consisting of an identification $\alpha : y = y'$, and an identification β witnessing that the square

$$\begin{array}{ccc} \mathsf{tr}_P(\mathsf{loop}, y) & \xrightarrow{\mathsf{ap}_{\mathsf{tr}_P(\mathsf{loop})}(\alpha)} & \mathsf{tr}_P(\mathsf{loop}, y') \\ {\scriptstyle P}\Big\| & & \Big\|{\scriptstyle P'} \\ y & \xrightarrow[\alpha]{} & y' \end{array}$$

commutes. Therefore it follows from the induction principle of the circle that for any $(y, p) : \sum_{(y:P(\mathsf{base}))} \mathsf{tr}_P(\mathsf{loop}, y) = y$, there is a dependent function $f : \prod_{(x:\mathbf{S}^1)} P(x)$ equipped with an identification

$$\alpha : f(\mathsf{base}) = y,$$

and an identification β witnessing that the square



commutes.

20.2 The (dependent) universal property of the circle

Our goal is now to use the induction principle of the circle to derive the **universal property** of the circle. This universal property states that, for any type X the canonical map

$$\left(\mathbf{S}^1 \to X\right) \to \left(\textstyle\sum_{(x:X)} x = x\right)$$

given by $f \mapsto (f(\mathsf{base}), \mathsf{ap}_f(\mathsf{loop}))$ is an equivalence. It turns out that it is easier to prove the **dependent universal property** first. The dependent universal property states that for any type family P over the circle, the canonical map

$$\left(\textstyle\prod_{(x:\mathbf{S}^1)} P(x)\right) \to \left(\textstyle\sum_{(y:P(\mathsf{base}))} \mathsf{tr}_P(\mathsf{loop}, y) = y\right)$$

given by $f \mapsto (f(\mathsf{base}), \mathsf{apd}_f(\mathsf{loop}))$ is an equivalence.

Theorem 20.2.1 *For any type family P over the circle, the map*

$$\left(\textstyle\prod_{(x:\mathbf{S}^1)} P(x)\right) \to \left(\textstyle\sum_{(y:P(\mathsf{base}))} \mathsf{tr}_P(\mathsf{loop}, y) = y\right)$$

given by $f \mapsto (f(\mathsf{base}), \mathsf{apd}_f(\mathsf{loop}))$ is an equivalence.

Proof By the induction principle of the circle we know that the map has a section, i.e., we have

$$\mathsf{ind}_{\mathbf{S}^1} : \left(\textstyle\sum_{(y:P(\mathsf{base}))} \mathsf{tr}_P(\mathsf{loop}, y) = y\right) \to \left(\textstyle\prod_{(x:\mathbf{S}^1)} P(x)\right)$$
$$\mathsf{comp}_{\mathbf{S}^1} : \mathsf{dgen}_{\mathbf{S}^1} \circ \mathsf{ind}_{\mathbf{S}^1} \sim \mathsf{id}$$

Therefore it remains to construct a homotopy

$$\mathsf{ind}_{\mathbf{S}^1} \circ \mathsf{dgen}_{\mathbf{S}^1} \sim \mathsf{id}.$$

Thus, for any $f : \prod_{(x:\mathbf{S}^1)} P(x)$ our task is to construct an identification

$$\mathsf{ind}_{\mathbf{S}^1}(\mathsf{dgen}_{\mathbf{S}^1}(f)) = f.$$

By function extensionality it suffices to construct a homotopy

$$\textstyle\prod_{(x:\mathbf{S}^1)} \mathsf{ind}_{\mathbf{S}^1}(\mathsf{dgen}_{\mathbf{S}^1}(f))(x) = f(x).$$

We proceed by the induction principle of the circle using the family of types $E_{g,f}(x) :\equiv g(x) = f(x)$ indexed by $x : \mathbf{S}^1$, where g is the function

$$g :\equiv \mathsf{ind}_{\mathbf{S}^1}(\mathsf{dgen}_{\mathbf{S}^1}(f)).$$

Thus, it suffices to construct

$$\alpha : g(\mathsf{base}) = f(\mathsf{base})$$
$$\beta : \mathsf{tr}_{E_{g,f}}(\mathsf{loop}, \alpha) = \alpha.$$

An argument by path induction on p yields that

$$\Big(\mathsf{apd}_g(p) \cdot r = \mathsf{ap}_{\mathsf{tr}_P(p)}(q) \cdot \mathsf{apd}_f(p)\Big) \to \Big(\mathsf{tr}_{E_{g,f}}(p, q) = r\Big),$$

for any $f, g : \prod_{(x:X)} P(x)$ and any $p : x = x'$, $q : g(x) = f(x)$ and $r : g(x') = f(x')$. Therefore it suffices to construct an identification $\alpha : g(\mathsf{base}) = f(\mathsf{base})$ equipped with an identification β witnessing that the square

$$\begin{array}{ccc}
\mathsf{tr}_P(\mathsf{loop}, g(\mathsf{base})) & \xrightarrow{\mathsf{ap}_{\mathsf{tr}_P(\mathsf{loop})}(\alpha)} & \mathsf{tr}_P(\mathsf{loop}, f(\mathsf{base})) \\
{\scriptstyle \mathsf{apd}_g(\mathsf{loop})}\Big\| & & \Big\|{\scriptstyle \mathsf{apd}_f(\mathsf{loop})} \\
g(\mathsf{base}) & \xrightarrow[\alpha]{} & f(\mathsf{base})\text{''}
\end{array}$$

commutes. Notice that we get exactly such a pair (α, β) from the computation rule of the circle, by Remark 20.1.3. □

As a corollary we obtain the following uniqueness principle for dependent functions defined by the induction principle of the circle.

Corollary 20.2.2 *Consider a type family P over the circle, and let*

$$y : P(\mathsf{base})$$
$$p : \mathsf{tr}_P(\mathsf{loop}, y) = y.$$

Then the type of functions $f : \prod_{(x:\mathbf{S}^1)} P(x)$ equipped with an identification

$$\alpha : f(\mathsf{base}) = y$$

and an identification β witnessing that the square

$$\begin{array}{ccc}
\mathsf{tr}_P(\mathsf{loop}, f(\mathsf{base})) & \xrightarrow{\mathsf{ap}_{\mathsf{tr}_P(\mathsf{loop})}(\alpha)} & \mathsf{tr}_P(\mathsf{loop}, y) \\
{\scriptstyle \mathsf{apd}_f(\mathsf{loop})}\Big\| & & \Big\|{\scriptstyle p} \\
f(\mathsf{base}) & \xrightarrow[\alpha]{} & y
\end{array}$$

commutes, is contractible.

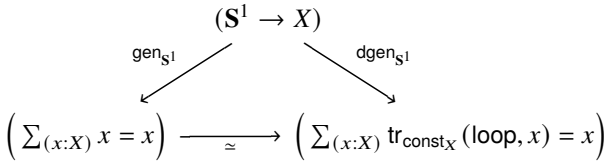
Now we use the dependent universal property to derive the ordinary universal property of the circle. It would be tempting to say that it is a direct corollary, but we need to address the transport that occurs in the dependent universal property.

Theorem 20.2.3 *For each type X, the* ***action on generators***

$$\mathsf{gen}_{\mathbf{S}^1} : (\mathbf{S}^1 \to X) \to \textstyle\sum_{(x:X)} x = x$$

given by $f \mapsto (f(\mathsf{base}), \mathsf{ap}_f(\mathsf{loop}))$ is an equivalence.

Proof We prove the claim by constructing a commuting triangle



in which the bottom map is an equivalence. Indeed, once we have such a triangle, we use the fact from Theorem 20.2.1 that $\mathsf{dgen}_{\mathbf{S}^1}$ is an equivalence to conclude that $\mathsf{gen}_{\mathbf{S}^1}$ is an equivalence.

To construct the bottom map, we first observe that for any constant type family const_B over a type A, any $p : a = a'$ in A, and any $b : B$, there is an identification

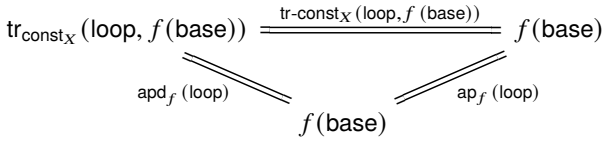
$$\mathsf{tr\text{-}const}_B(p, b) = b.$$

This identification is easily constructed by path induction on p. Now we construct the bottom map as the induced map on total spaces of the family of maps

$$l \mapsto \mathsf{tr\text{-}const}_X(\mathsf{loop}, x) \cdot l,$$

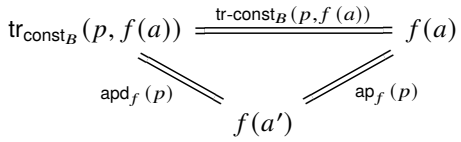
indexed by $x : X$. Since concatenating by a path is an equivalence, it follows by Theorem 11.1.3 that the induced map on total spaces is indeed an equivalence.

To show that the triangle commutes, it suffices to construct for any $f : \mathbf{S}^1 \to X$ an identification witnessing that the triangle



commutes. This again follows from general considerations: for any $f : A \to B$

and any $p : a = a'$ in A, the triangle

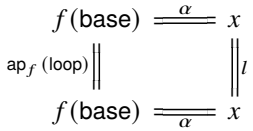


commutes by path induction on p. □

Corollary 20.2.4 *For any loop $l : x = x$ in a type X, the type of maps $f : \mathbf{S}^1 \to X$ equipped with an identification*

$$\alpha : f(\mathsf{base}) = x$$

and an identification β witnessing that the square



commutes, is contractible.

20.3 Multiplication on the circle

One way the circle arises classically, is as the set of complex numbers at distance 1 from the origin. It is an elementary fact that $|xy| = |x||y|$ for any two complex numbers $x, y \in \mathbb{C}$, so it follows that when we multiply two complex numbers that both lie on the unit circle, then the result lies again on the unit circle. Thus, using complex multiplication we see that there is a multiplication operation on the circle. And there is a shadow of this operation in type theory, even though our circle arises in a very different way!

Definition 20.3.1 We define a binary operation

$$\mathsf{mul}_{\mathbf{S}^1} : \mathbf{S}^1 \to (\mathbf{S}^1 \to \mathbf{S}^1).$$

Construction Using the universal property of the circle, we define $\mathsf{mul}_{\mathbf{S}^1}$ as the unique map $\mathbf{S}^1 \to (\mathbf{S}^1 \to \mathbf{S}^1)$ equipped with an identification

$$\mathsf{base\text{-}mul}_{\mathbf{S}^1} : \mathsf{mul}_{\mathbf{S}^1}(\mathsf{base}) = \mathsf{id}$$

and an identification $\mathsf{loop\text{-}mul}_{S^1}$ witnessing that the square

$$\begin{array}{ccc} \mathsf{mul}_{S^1}(\mathsf{base}) & \overset{\mathsf{base\text{-}mul}_{S^1}}{=\!=\!=} & \mathsf{id} \\ {\scriptstyle \mathsf{ap}_{\mathsf{mul}_{S^1}}(\mathsf{loop})}\Big\| & & \Big\|{\scriptstyle \mathsf{eq\text{-}htpy}(H)} \\ \mathsf{mul}_{S^1}(\mathsf{base}) & \underset{\mathsf{base\text{-}mul}_{S^1}}{=\!=\!=} & \mathsf{id} \end{array}$$

commutes. Note that in this square we have a homotopy $H : \mathsf{id} \sim \mathsf{id}$, which is not yet defined. We use the dependent universal property of the circle with respect to the family $E_{\mathsf{id},\mathsf{id}}$ given by

$$E_{\mathsf{id},\mathsf{id}}(x) :\equiv (x = x),$$

to define H as the unique homotopy equipped with an identification

$$\alpha : H(\mathsf{base}) = \mathsf{loop}$$

and an identification β witnessing that the square

$$\begin{array}{ccc} \mathsf{tr}_{E_{\mathsf{id},\mathsf{id}}}(\mathsf{loop}, H(\mathsf{base})) & \overset{\mathsf{ap}_{\mathsf{tr}_{E_{\mathsf{id},\mathsf{id}}}(\mathsf{loop})}(\alpha)}{=\!=\!=} & \mathsf{tr}_{E_{\mathsf{id},\mathsf{id}}}(\mathsf{loop}, \mathsf{loop}) \\ {\scriptstyle \mathsf{apd}_H(\mathsf{loop})}\Big\| & & \Big\|{\scriptstyle \gamma} \\ H(\mathsf{base}) & \underset{\alpha}{=\!=\!=} & \mathsf{loop} \end{array}$$

commutes. Now it remains to define the path $\gamma : \mathsf{tr}_{E_{\mathsf{id},\mathsf{id}}}(\mathsf{loop}, \mathsf{loop}) = \mathsf{loop}$ in the above square. To proceed, we first observe that a simple path induction argument yields a function

$$\Big(p \cdot r = q \cdot p\Big) \to \Big(\mathsf{tr}_{E_{\mathsf{id},\mathsf{id}}}(p, q) = r\Big),$$

for any $p : \mathsf{base} = x$, $q : \mathsf{base} = \mathsf{base}$ and $r : x = x$. In particular, we have a function

$$\Big(\mathsf{loop} \cdot \mathsf{loop} = \mathsf{loop} \cdot \mathsf{loop}\Big) \to \Big(\mathsf{tr}_{E_{\mathsf{id},\mathsf{id}}}(\mathsf{loop}, \mathsf{loop}) = \mathsf{loop}\Big).$$

Now we apply this function to $\mathsf{refl}_{\mathsf{loop} \cdot \mathsf{loop}}$ to obtain the desired identification

$$\gamma : \mathsf{tr}_{E_{\mathsf{id},\mathsf{id}}}(\mathsf{loop}, \mathsf{loop}) = \mathsf{loop}. \qquad \square$$

Remark 20.3.2 In the definition of $H : \mathsf{id} \sim \mathsf{id}$ above, it is important that we didn't choose H to be $\mathsf{refl\text{-}htpy}$. If we had done so, the resulting operation would be homotopic to $x, y \mapsto y$, which is clearly not what we had in mind with the multiplication operation on the circle. See also Exercise 20.2.

The left unit law $\mathsf{mul}_{\mathbf{S}^1}(\mathsf{base}, x) = x$ holds by the computation rule of the universal property. More precisely, we define

$$\mathsf{left\text{-}unit}_{\mathbf{S}^1} :\equiv \mathsf{htpy\text{-}eq}(\mathsf{base\text{-}mul}_{\mathbf{S}^1}).$$

For the right unit law, however, we need to give a separate argument that is surprisingly involved, because all the aspects of the definition of $\mathsf{mul}_{\mathbf{S}^1}$ will come out and play their part.

Theorem 20.3.3 *The multiplication operation on the circle satisfies the right unit law, i.e., we have*

$$\mathsf{mul}_{\mathbf{S}^1}(x, \mathsf{base}) = x$$

for any $x : \mathbf{S}^1$.

Proof The proof is by induction on the circle. In the base case we use the left unit law

$$\mathsf{left\text{-}unit}_{\mathbf{S}^1}(\mathsf{base}) : \mathsf{mul}_{\mathbf{S}^1}(\mathsf{base}, \mathsf{base}) = \mathsf{base}.$$

Thus, it remains to show that

$$\mathsf{tr}_P(\mathsf{loop}, \mathsf{left\text{-}unit}_{\mathbf{S}^1}(\mathsf{base})) = \mathsf{left\text{-}unit}_{\mathbf{S}^1}(\mathsf{base}),$$

where P is the family over the circle given by

$$P(x) :\equiv \mathsf{mul}_{\mathbf{S}^1}(x, \mathsf{base}) = x.$$

Now we observe that there is a function

$$\Big(\mathsf{htpy\text{-}eq}(\mathsf{ap}_{\mathsf{mul}_{\mathbf{S}^1}}(p))(\mathsf{base}) \cdot r = q \cdot p\Big) \to \Big(\mathsf{tr}_P(p, q) = r\Big),$$

for any

$$\begin{aligned} p &: \mathsf{base} = x \\ q &: \mathsf{mul}_{\mathbf{S}^1}(\mathsf{base}, \mathsf{base}) = \mathsf{base} \\ r &: \mathsf{mul}_{\mathbf{S}^1}(x, \mathsf{base}) = x. \end{aligned}$$

Thus we see that, in order to construct an identification

$$\mathsf{tr}_P(\mathsf{loop}, \mathsf{left\text{-}unit}_{\mathbf{S}^1}) = \mathsf{left\text{-}unit}_{\mathbf{S}^1},$$

it suffices to show that the square

$$\begin{array}{ccc} \mathsf{mul}_{\mathbf{S}^1}(\mathsf{base}, \mathsf{base}) & \xrightarrow{\mathsf{left\text{-}unit}_{\mathbf{S}^1}(\mathsf{base})} & \mathsf{base} \\ {\scriptstyle \mathsf{htpy\text{-}eq}(\mathsf{ap}_{\mathsf{mul}_{\mathbf{S}^1}}(\mathsf{loop}))(\mathsf{base})}\Big\| & & \Big\|{\scriptstyle \mathsf{loop}} \\ \mathsf{mul}_{\mathbf{S}^1}(\mathsf{base}, \mathsf{base}) & \xrightarrow[\mathsf{left\text{-}unit}_{\mathbf{S}^1}(\mathsf{base})]{} & \mathsf{base} \end{array}$$

commutes. Now we note that we have an identification $H(\mathsf{base}) = \mathsf{loop}$. It is indeed at this point, where it is important that H is not the trivial homotopy, because now we can proceed by observing that the above square commutes if and only if the square

$$\begin{array}{ccc} \mathsf{mul}_{\mathbf{S}^1}(\mathsf{base},\mathsf{base}) & \xrightarrow{\mathsf{htpy\text{-}eq}(\mathsf{base\text{-}mul}_{\mathbf{S}^1})(\mathsf{base})} & \mathsf{base} \\ {\scriptstyle \mathsf{htpy\text{-}eq}(\mathsf{ap}_{\mathsf{mul}_{\mathbf{S}^1}}(\mathsf{loop}))(\mathsf{base})}\Big\| & & \Big\|{\scriptstyle H(\mathsf{base})} \\ \mathsf{mul}_{\mathbf{S}^1}(\mathsf{base},\mathsf{base}) & \xrightarrow[\mathsf{htpy\text{-}eq}(\mathsf{base\text{-}mul}_{\mathbf{S}^1})(\mathsf{base})]{} & \mathsf{base} \end{array}$$

commutes. The commutativity of this square easily follows from the identification $\mathsf{loop\text{-}mul}_{\mathbf{S}^1}$ constructed in Definition 20.3.1. □

Exercises

20.1 (a) Let $P : \mathbf{S}^1 \to \mathsf{Prop}$ be a family of propositions over the circle. Show that

$$P(\mathsf{base}) \to \textstyle\prod_{(x:\mathbf{S}^1)} P(x).$$

In this sense the circle is *connected*.

(b) Show that any embedding $m : \mathbf{S}^1 \to \mathbf{S}^1$ is an equivalence.

(c) Show that for any embedding $m : X \to \mathbf{S}^1$, there is a proposition P and an equivalence $e : X \simeq \mathbf{S}^1 \times P$ for which the triangle

$$\begin{array}{ccc} X & \xrightarrow{e} & \mathbf{S}^1 \times P \\ & {\scriptstyle m}\searrow \quad \swarrow{\scriptstyle \mathsf{pr}_1} & \\ & \mathbf{S}^1 & \end{array}$$

commutes. In other words, all the embeddings into the circle are of the form $\mathbf{S}^1 \times P \to \mathbf{S}^1$.

20.2 Show that for any type X and any $x : X$, the map

$$\mathsf{ind}_{\mathbf{S}^1}(x, \mathsf{refl}_x) : \mathbf{S}^1 \to X$$

is homotopic to the constant map const_x.

20.3 (a) Show that for any $x : \mathbf{S}^1$, both functions

$$\mathsf{mul}_{\mathbf{S}^1}(x, -) \qquad \text{and} \qquad \mathsf{mul}_{\mathbf{S}^1}(-, x)$$

are equivalences.

(b) Show that the function

$$\mathsf{mul}_{\mathbf{S}^1} : \mathbf{S}^1 \to (\mathbf{S}^1 \to \mathbf{S}^1)$$

is an embedding. Compare this fact with Exercise 19.2.

(c) Show that multiplication on the circle is associative and commutative.

20.4 (a) Show that a type X is a set if and only if the map

$$\lambda x.\, \lambda t.\, x : X \to (\mathbf{S}^1 \to X)$$

is an equivalence.

(b) Show that a type X is a set if and only if the map

$$\lambda f.\, f(\mathsf{base}) : (\mathbf{S}^1 \to X) \to X$$

is an equivalence.

20.5 Show that the multiplicative operation on the circle is commutative, i.e. construct an identification

$$\mathsf{mul}_{\mathbf{S}^1}(x, y) = \mathsf{mul}_{\mathbf{S}^1}(y, x).$$

for every $x, y : \mathbf{S}^1$.

20.6 Show that the circle, equipped with the multiplicative operation $\mathsf{mul}_{\mathbf{S}^1}$ is an abelian group, i.e. construct an inverse operation

$$\mathsf{inv}_{\mathbf{S}^1} : \mathbf{S}^1 \to \mathbf{S}^1$$

and construct identifications

$$\mathsf{left\text{-}inv}_{\mathbf{S}^1} : \mathsf{mul}_{\mathbf{S}^1}(\mathsf{inv}_{\mathbf{S}^1}(x), x) = \mathsf{base}$$
$$\mathsf{right\text{-}inv}_{\mathbf{S}^1} : \mathsf{mul}_{\mathbf{S}^1}(x, \mathsf{inv}_{\mathbf{S}^1}(x)) = \mathsf{base}.$$

Moreover, show that the square

$$\begin{array}{ccc} \mathsf{inv}_{\mathbf{S}^1}(\mathsf{base}) & = & \mathsf{mul}_{\mathbf{S}^1}(\mathsf{base}, \mathsf{inv}_{\mathbf{S}^1}(\mathsf{base})) \\ \| & & \| \\ \mathsf{mul}_{\mathbf{S}^1}(\mathsf{inv}_{\mathbf{S}^1}(\mathsf{base}), \mathsf{base}) & = & \mathsf{base} \end{array}$$

commutes.

20.7 Show that for any multiplicative operation

$$\mu : \mathbf{S}^1 \to (\mathbf{S}^1 \to \mathbf{S}^1)$$

that satisfies the condition that $\mu(x, -)$ and $\mu(-, x)$ are equivalences for any $x : \mathbf{S}^1$, there is an element $e : \mathbf{S}^1$ such that

$$\mu(x, y) = \mathsf{mul}_{\mathbf{S}^1}(x, \mathsf{mul}_{\mathbf{S}^1}(\bar{e}, y))$$

for every $x, y : \mathbf{S}^1$, where $\bar{e} :\equiv \mathsf{inv}_{\mathbf{S}^1}(e)$ is the complex conjucation of e on $\mathbf{S}^1$.

21 The universal cover of the circle

In this section we show that the loop space of the circle is equivalent to $\mathbb{Z}$ by constructing the universal cover of the circle as an application of the univalence axiom.

21.1 Families over the circle

The type of small families over $\mathbf{S}^1$ is just the function type $\mathbf{S}^1 \to \mathcal{U}$, so in fact we may use the universal property of the circle to construct small dependent types over the circle. By the universal property, small type families over $\mathbf{S}^1$ are equivalently described as pairs (X, p) consisting of a type $X : \mathcal{U}$ and an identification $p : X = X$. This is where the univalence axiom comes in. By the map

$$\mathsf{eq\text{-}equiv}_{X,X} : (X \simeq X) \to (X = X)$$

it suffices to provide an equivalence $X \simeq X$.

Definition 21.1.1 Consider a type X and every equivalence $e : X \simeq X$. We will construct a dependent type $\mathcal{D}(X, e) : \mathbf{S}^1 \to \mathcal{U}$ with an equivalence $x \mapsto x_{\mathcal{D}} : X \simeq \mathcal{D}(X, e, \mathsf{base})$ for which the square

$$\begin{array}{ccc} X & \xrightarrow{\simeq} & \mathcal{D}(X, e, \mathsf{base}) \\ {\scriptstyle e}\downarrow & & \downarrow{\scriptstyle \mathsf{tr}_{\mathcal{D}(X,e)}(\mathsf{loop})} \\ X & \xrightarrow[\simeq]{} & \mathcal{D}(X, e, \mathsf{base}) \end{array}$$

commutes. We also write $d \mapsto d_X$ for the inverse of this equivalence, so that the relations

$$(x_{\mathcal{D}})_X = x \qquad (e(x)_{\mathcal{D}}) = \mathsf{tr}_{\mathcal{D}(X,e)}(\mathsf{loop}, x_{\mathcal{D}})$$
$$(d_X)_{\mathcal{D}} = d \qquad (\mathsf{tr}_{\mathcal{D}(X,e)}(d))_X = e(d_X)$$

hold.

The type $\sum_{(X:\mathcal{U})} X \simeq X$ is also called the type of **descent data** for the circle.

Construction An easy path induction argument reveals that

$$\mathsf{equiv\text{-}eq}(\mathsf{ap}_P(\mathsf{loop})) = \mathsf{tr}_P(\mathsf{loop})$$

for each dependent type $P : \mathbf{S}^1 \to \mathcal{U}$. Therefore we see that the triangle

$$\begin{array}{ccc} & (\mathbf{S}^1 \to \mathcal{U}) & \\ {\scriptstyle \mathsf{gen}_{\mathbf{S}^1}}\swarrow & & \searrow{\scriptstyle \mathsf{desc}_{\mathbf{S}^1}} \\ \sum_{(X:\mathcal{U})} X = X & \xrightarrow[\mathsf{tot}\,(\lambda X.\,\mathsf{equiv\text{-}eq}_{X,X})]{} & \sum_{(X:\mathcal{U})} X \simeq X \end{array}$$

commutes, where the map $\mathsf{desc}_{\mathbf{S}^1}$ is given by $P \mapsto (P(\mathsf{base}), \mathsf{tr}_P(\mathsf{loop}))$ and the bottom map is an equivalence by the univalence axiom and Theorem 11.1.3. Now it follows by the 3-for-2 property that $\mathsf{desc}_{\mathbf{S}^1}$ is an equivalence, since $\mathsf{gen}_{\mathbf{S}^1}$ is an equivalence by Theorem 20.2.3. This means that for every type X and every $e : X \simeq X$ there is a type family $\mathcal{D}(X, e) : \mathbf{S}^1 \to \mathcal{U}$ such that

$$(\mathcal{D}(X, e, \mathsf{base}), \mathsf{tr}_{\mathcal{D}(X,e)}(\mathsf{loop})) = (X, e).$$

Equivalently, we have $p : \mathcal{D}(X, e, \mathsf{base}) = X$ and $\mathsf{tr}(p, \mathsf{tr}_{\mathcal{D}(X,e)}(\mathsf{loop})) = e$. Thus, we obtain $\mathsf{equiv\text{-}eq}(p) : \mathcal{D}(X, e, \mathsf{base}) \simeq X$, for which the square

$$\begin{array}{ccc} \mathcal{D}(X, e, \mathsf{base}) & \xrightarrow{\mathsf{equiv\text{-}eq}(p)} & X \\ {\scriptstyle \mathsf{tr}_{\mathcal{D}(X,e)}(\mathsf{loop})}\downarrow & & \downarrow{\scriptstyle e} \\ \mathcal{D}(X, e, \mathsf{base}) & \xrightarrow[\mathsf{equiv\text{-}eq}(p)]{} & X \end{array}$$

commutes. □

21.2 The universal cover of the circle

The *universal cover* of the circle is a family of sets over the circle with contractible total space. Classically, the universal cover is described as a map $\mathbb{R} \to \mathbf{S}^1$ that winds the real line around the circle. In homotopy type theory there is no analogue of such a construction.

Recall from Example 9.2.5 that the successor function $\mathsf{succ} : \mathbb{Z} \to \mathbb{Z}$ is an equivalence. Its inverse is the predecessor function defined in Exercise 4.1 (a).

Definition 21.2.1 The **universal cover** of the circle is the dependent type $\mathcal{E}_{\mathbf{S}^1} :\equiv \mathcal{D}(\mathbb{Z}, \mathsf{succ}) : \mathbf{S}^1 \to \mathcal{U}$.

Remark 21.2.2 The universal cover of the circle comes equipped with an equivalence

$$e : \mathbb{Z} \simeq \mathcal{E}_{\mathbf{S}^1}(\mathsf{base})$$

and a homotopy witnessing that the square

$$\begin{array}{ccc} \mathbb{Z} & \xrightarrow{e} & \mathcal{E}_{\mathbf{S}^1}(\mathsf{base}) \\ {\scriptstyle \mathsf{succ}}\downarrow & & \downarrow{\scriptstyle \mathrm{tr}_{\mathcal{E}_{\mathbf{S}^1}}(\mathsf{loop})} \\ \mathbb{Z} & \xrightarrow[e]{} & \mathcal{E}_{\mathbf{S}^1}(\mathsf{base}) \end{array}$$

commutes.

For convenience, we write $k_{\mathcal{E}}$ for the element $e(k) : \mathcal{E}_{\mathbf{S}^1}(\mathsf{base})$, for any $k : \mathbb{Z}$.

The picture of the universal cover is that of a helix over the circle. This picture emerges from the path liftings of loop in the total space. The segments of the helix connecting k to $k + 1$ in the total space of the helix, are constructed in the following lemma.

Lemma 21.2.3 *For any $k : \mathbb{Z}$, there is an identification*

$$\mathsf{segment\text{-}helix}_k : (\mathsf{base}, k_{\mathcal{E}}) = (\mathsf{base}, \mathsf{succ}(k)_{\mathcal{E}})$$

in the total space $\sum_{(t:\mathbf{S}^1)} \mathcal{E}(t)$.

Proof By Theorem 9.3.4 it suffices to show that

$$\prod_{(k:\mathbb{Z})} \sum_{(\alpha:\mathsf{base}=\mathsf{base})} \mathrm{tr}_{\mathcal{E}}(\alpha, k_{\mathcal{E}}) = \mathsf{succ}(k)_{\mathcal{E}}.$$

We just take $\alpha :\equiv \mathsf{loop}$. Then we have $\mathrm{tr}_{\mathcal{E}}(\alpha, k_{\mathcal{E}}) = \mathsf{succ}(k)_{\mathcal{E}}$ by the commuting square provided in the definition of $\mathcal{E}$. □

21.3 Contractibility of general total spaces

Consider a type X, a family P over X, an element $c : \sum_{(x:X)} P(x)$, and suppose our goal is to construct a contraction

$$\prod_{(t:\sum_{(x:X)} P(x))} c = t.$$

Of course, the first step is to apply the induction principle of Σ-types, so it suffices to construct a dependent function of type

$$\prod_{(x:X)} \prod_{(y:P(x))} c = (x, y).$$

In the case where P is the universal cover of the circle, we are given an equivalence $e : \mathbb{Z} \simeq \mathcal{E}(\mathsf{base})$. Using this equivalence, we obtain an equivalence

$$\left(\prod_{(y:\mathcal{E}(y))} c = (\mathsf{base}, y)\right) \to \left(\prod_{(k:\mathbb{Z})} c = (\mathsf{base}, k_{\mathcal{E}})\right).$$

More generally, if we are given an equivalence $e : F \simeq P(x)$ for some $x : X$, then we have an equivalence

$$\left(\prod_{(y:P(x))} c = (x, y)\right) \to \left(\prod_{(y:F)} c = (x, e(y))\right) \tag{21.3.1}$$

by precomposing with the equivalence e. Therefore we can construct a dependent function of type $\prod_{(y:P(x))} c = (x, y)$ by constructing a dependent function of type $\prod_{(y:F)} c = (x, e(y))$.

Furthermore, if we consider a path $p : x = x'$ in X and a commuting square

$$\begin{array}{ccc} F & \xrightarrow{e} & P(x) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{tr}_P(p)} \\ F' & \xrightarrow{e'} & P(x') \end{array}$$

where e, e', and f are all equivalences, then we obtain a function

$$\psi : \left(\prod_{(y:F)} c = (x, e(y))\right) \to \left(\prod_{(y':F')} c = (x, e'(y'))\right).$$

The function ψ is constructed as follows. Given $h : \prod_{(y:F)} c = (x, e(y))$ and $y' : F'$ we have the path $h(f^{-1}(y')) : c = (x, e(f^{-1}(y')))$. Moreover, writing G for the homotopy $f \circ f^{-1} \sim \mathsf{id}$, we have the path

$$\mathsf{tr}_P(p, e(f^{-1}(y'))) \xrightarrow{H(f^{-1}(y'))} e'(f(f^{-1}(y'))) \xrightarrow{\mathsf{ap}_{e'}(G(y'))} e'(y').$$

From this concatenated path we obtain the path

$$(x, e(f^{-1}(y'))) \xrightarrow{\mathsf{eq\text{-}pair}(p, H(f^{-1}(y'))\cdot \mathsf{ap}_{e'}(G(y')))} (x', e'(y')).$$

Now we define the function ψ by

$$h \mapsto \lambda y'.\, h(f^{-1}(y')) \cdot \mathsf{eq\text{-}pair}(p, H(f^{-1}(y')) \cdot \mathsf{ap}_{e'}(G(y'))).$$

Note that ψ is an equivalence, since it is given as precomposition by the equivalence f^{-1}, followed by postcomposition by concatenation, which is also an equivalence. Now we state the main technical result of this section, which will help us prove the contractibility of the total space of the universal cover of the circle by computing transport in the family $x \mapsto \prod_{(y:P(x))} c = (x, y)$.

Definition 21.3.1 Consider a path $p : x = x'$ in X and a commuting square

$$\begin{array}{ccc} F & \xrightarrow{e} & P(x) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{tr}_P(p)} \\ F' & \xrightarrow{e'} & P(x') \end{array}$$

with $H : e' \circ f \sim \mathsf{tr}_P(p) \circ e$, where e, e', and f are all equivalences. Then there is for any $y : F$ an identification

$$\mathsf{segment\text{-}tot}(y) : (x, e(y)) = (x', e'(f(y)))$$

defined as $\mathsf{segment\text{-}tot}(y) :\equiv \mathsf{eq\text{-}pair}(p, H(y)^{-1})$.

Lemma 21.3.2 *Consider a path $p : x = x'$ in X and a commuting square*

$$\begin{array}{ccc} F & \xrightarrow{e} & P(x) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{tr}_P(p)} \\ F' & \xrightarrow{e'} & P(x') \end{array}$$

with $H : e' \circ f \sim \mathsf{tr}_P(p) \circ e$, where e, e', and f are all equivalences. Furthermore, let

$$h : \textstyle\prod_{(y:F)} c = (x, e(y))$$
$$h' : \textstyle\prod_{(y':F')} c = (x', e'(y')).$$

Then there is an equivalence

$$\left(\textstyle\prod_{(y:F)} h'(f(y)) = h(y) \cdot \mathsf{segment\text{-}tot}(y)\right) \simeq \left(\mathsf{tr}_C(p, \varphi(h)) = \varphi'(h')\right).$$

Proof We first note that we have a commuting square

$$\begin{array}{ccc} \prod_{(y:B(x))} c = (x, y) & \xrightarrow{-\circ e} & \prod_{(y:F)} c = (x, e(y)) \\ {\scriptstyle \mathsf{tr}_C(p)}\downarrow & & \uparrow{\scriptstyle \psi} \\ \prod_{(y':B(x'))} c = (x', y') & \xrightarrow[-\circ e']{} & \prod_{(y':F')} c = (x', e'(y')) \end{array}$$

where $\psi(h') = \lambda y.\, h'(f(y)) \cdot \mathsf{segment\text{-}tot}(y)^{-1}$. All the maps in this square are equivalences. In particular, the inverses of the top and bottom maps are φ and φ', respectively. The claim follows from this observation, but we will spell out the details.

Since any equivalence is an embedding, we see immediately that the type $\mathsf{tr}_C(p)(\varphi(h)) = \varphi'(h')$ is equivalent to the type

$$\psi(\mathsf{tr}_C(p)(\varphi(h)) \circ e') = \psi(\varphi'(h') \circ e').$$

By the commutativity of the square, the left-hand side is h. The right-hand side is $\psi(h')$. Therefore it follows that

$$\begin{aligned} \left(\mathsf{tr}_C(p)(\varphi(h)) = \varphi'(h')\right) &\simeq \left(h = \lambda y.\, h'(f(y)) \cdot \mathsf{segment\text{-}tot}(y)^{-1}\right) \\ &\simeq \left(h' \circ f \sim (\lambda y.\, h(y) \cdot \mathsf{segment\text{-}tot}(y)\right). \end{aligned}$$

□

Applying these observations to the universal cover of the circle, we obtain the following lemma that we will use to prove that the total space of $\mathcal{E}$ is contractible.

Corollary 21.3.3 *In order to show that the total space of $\mathcal{E}$ is contractible, it suffices to construct a function*

$$h : \prod_{(k:\mathbb{Z})} (\mathsf{base}, 0_{\mathcal{E}}) = (\mathsf{base}, k_{\mathcal{E}})$$

equipped with a homotopy

$$H : \prod_{(k:\mathbb{Z})} h(\mathsf{succ}(k)_{\mathcal{E}}) = h(k) \cdot \mathsf{segment\text{-}helix}(k).$$

In the next section we establish the dependent universal property of the integers, which we will use with Corollary 21.3.3 to show that the total space of the universal cover is contractible.

21.4 The dependent universal property of the integers

Lemma 21.4.1 *Let B be a family over $\mathbb{Z}$, equipped with an element $b_0 : B(0)$, and an equivalence*

$$e_k : B(k) \simeq B(\mathsf{succ}(k))$$

for each $k : \mathbb{Z}$. Then there is a dependent function $f : \prod_{(k:\mathbb{Z})} B(k)$ equipped with identifications $f(0) = b_0$ and

$$f(\mathsf{succ}(k)) = e_k(f(k))$$

for any $k : \mathbb{Z}$.

Proof The map is defined using the induction principle for the integers, stated in Remark 4.5.2. First we take

$$\begin{aligned} f(-1) &:\equiv e^{-1}(b_0) \\ f(0) &:\equiv b_0 \\ f(1) &:\equiv e(b_0). \end{aligned}$$

For the induction step on the negative integers we use

$$\lambda n.\, e^{-1}_{\mathsf{neg}(S(n))} : \prod_{(n:\mathbb{N})} B(\mathsf{neg}(n)) \to B(\mathsf{neg}(S(n)))$$

For the induction step on the positive integers we use

$$\lambda n.\, e(\mathsf{pos}(n)) : \prod_{(n:\mathbb{N})} B(\mathsf{pos}(n)) \to B(\mathsf{pos}(S(n))).$$

The computation rules follow in a straightforward way from the computation rules of $\mathbb{Z}$-induction and the fact that e^{-1} is an inverse of e. □

Example 21.4.2 For any type A, we obtain a map $f : \mathbb{Z} \to A$ from any $x : A$ and any equivalence $e : A \simeq A$, such that $f(0) = x$ and the square

$$\begin{array}{ccc} \mathbb{Z} & \xrightarrow{f} & A \\ {\scriptstyle \mathsf{succ}}\downarrow & & \downarrow{\scriptstyle e} \\ \mathbb{Z} & \xrightarrow[f]{} & A \end{array}$$

commutes. In particular, if we take $A \doteq (x = x)$ for some $x : X$, then for any $p : x = x$ we have the equivalence $\lambda q.\, p \cdot q : (x = x) \to (x = x)$. This equivalence induces a map

$$k \mapsto p^k : \mathbb{Z} \to (x = x),$$

for any $p : x = x$. This induces the **degree** k **map** on the circle

$$\mathsf{deg}(k) : \mathbf{S}^1 \to \mathbf{S}^1,$$

for any $k : \mathbb{Z}$, see **??**.

In the following theorem we show that the dependent function constructed in Lemma 21.4.1 is unique.

Theorem 21.4.3 *Consider a type family $B : \mathbb{Z} \to \mathcal{U}$ equipped with $b : B(0)$ and a family of equivalences*

$$e : \textstyle\prod_{(k:\mathbb{Z})} B(k) \simeq B(\mathsf{succ}(k)).$$

Then the type

$$\textstyle\sum_{(f:\prod_{(k:\mathbb{Z})} B(k))} (f(0) = b) \times \prod_{(k:\mathbb{Z})} f(\mathsf{succ}(k)) = e_k(f(k))$$

is contractible.

Proof In Lemma 21.4.1 we have already constructed an element of the asserted type. Therefore it suffices to show that any two elements of this type can be identified. Note that the type $(f, p, H) = (f', p', H')$ is equivalent to the type

$$\textstyle\sum_{(K:f \sim f')} (K(0) = p \cdot (p')^{-1}) \times \prod_{(k:\mathbb{Z})} K(\mathsf{succ}(k)) = (H(k) \cdot \mathsf{ap}_{e_k}(K(k))) \cdot H'(k)^{-1}.$$

We obtain an element of this type by applying Lemma 21.4.1 to the family C over $\mathbb{Z}$ given by $C(k) :\equiv f(k) = f'(k)$, which comes equipped with a base point

$$p \cdot (p')^{-1} : C(0),$$

and the family of equivalences

$$\lambda(\alpha : f(k) = f'(k)).\, (H(k) \cdot \mathsf{ap}_{e_k}(\alpha)) \cdot H'(k)^{-1} : \textstyle\prod_{(k:\mathbb{Z})} C(k) \simeq C(\mathsf{succ}(k)).$$

□

One way of phrasing the following corollary, is that $\mathbb{Z}$ is the 'initial type equipped with a point and an automorphism'.

Corollary 21.4.4 *For any type X equipped with a base point $x_0 : X$ and an automorphism $e : X \simeq X$, the type*

$$\textstyle\sum_{(f:\mathbb{Z}\to X)} (f(0) = x_0) \times ((f \circ \mathsf{succ}) \sim (e \circ f))$$

is contractible.

21.5 The identity type of the circle

Lemma 21.5.1 *The total space $\sum_{(t:\mathbf{S}^1)} \mathcal{E}(t)$ of the universal cover of $\mathbf{S}^1$ is contractible.*

Proof By Corollary 21.3.3 it suffices to construct a function

$$h : \textstyle\prod_{(k:\mathbb{Z})} (\mathsf{base}, 0_{\mathcal{E}}) = (\mathsf{base}, k_{\mathcal{E}})$$

equipped with a homotopy

$$H : \textstyle\prod_{(k:\mathbb{Z})} h(\mathsf{succ}(k)_{\mathcal{E}}) = h(k) \cdot \mathsf{segment\text{-}helix}(k).$$

We obtain h and H by the elimination principle of Lemma 21.4.1. Indeed, the family P over the integers given by $P(k) :\equiv (\mathsf{base}, 0_{\mathcal{E}}) = (\mathsf{base}, k_{\mathcal{E}})$ comes equipped with the element $\mathsf{refl}_{(\mathsf{base},0_{\mathcal{E}})} : P(0)$, and the family of equivalences

$$\textstyle\prod_{(k:\mathbb{Z})} P(k) \simeq P(\mathsf{succ}(k))$$

given by $k, p \mapsto p \cdot \mathsf{segment\text{-}helix}(k)$. □

Theorem 21.5.2 *The family of maps*

$$\textstyle\prod_{(t:\mathbf{S}^1)} (\mathsf{base} = t) \to \mathcal{E}(t)$$

sending $\mathsf{refl}_{\mathsf{base}}$ to $0_{\mathcal{E}}$ is a family of equivalences. In particular, the loop space of the circle is equivalent to $\mathbb{Z}$.

Proof This is a direct corollary of Lemma 21.5.1 and Theorem 11.2.2. □

Corollary 21.5.3 *The circle is a 1-type and not a 0-type.*

Proof To see that the circle is a 1-type we have to show that $s = t$ is a 0-type for every $s, t : \mathbf{S}^1$. By Exercise 20.1 it suffices to show that the loop space of the circle is a 0-type. This is indeed the case, because $\mathbb{Z}$ is a 0-type, and we have an equivalence $(\mathsf{base} = \mathsf{base}) \simeq \mathbb{Z}$.

Furthermore, since $\mathbb{Z}$ is a 0-type and not a (-1)-type, it follows that the circle is a 1-type and not a 0-type. □

Exercises

21.1 Show that the map

$$\mathbb{Z} \to \Omega(\mathbf{S}^1)$$

is a group homomorphism. Conclude that the loop space $\Omega(\mathbf{S}^1)$ as a group is isomorphic to $\mathbb{Z}$.

21.2 (a) Show that

$$\prod_{(x:\mathbf{S}^1)} \neg\neg(\mathsf{base} = x).$$

(b) On the other hand, use the universal cover of the circle to show that

$$\neg\left(\prod_{(x:\mathbf{S}^1)} \mathsf{base} = x\right).$$

(c) Conclude that

$$\neg\left(\prod_{(X:\mathcal{U})} \neg\neg X \to X\right)$$

for any univalent universe $\mathcal{U}$ containing the circle.

21.3 (a) Show that for every $x : X$, we have an equivalence

$$\left(\sum_{(f:\mathbf{S}^1\to X)} f(\mathsf{base}) = x\right) \simeq (x = x)$$

(b) Show that for every $t : \mathbf{S}^1$, we have an equivalence

$$\left(\sum_{(f:\mathbf{S}^1\to\mathbf{S}^1)} f(\mathsf{base}) = t\right) \simeq \mathbb{Z}$$

The base point preserving map $f : \mathbf{S}^1 \to \mathbf{S}^1$ corresponding to $k : \mathbb{Z}$ is called the **degree** k **map** on the circle, and is denoted by $\mathsf{deg}(k)$.

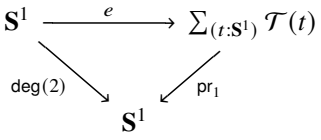
(c) Show that for every $t : \mathbf{S}^1$, we have an equivalence

$$\left(\sum_{(e:\mathbf{S}^1\simeq\mathbf{S}^1)} e(\mathsf{base}) = t\right) \simeq \mathsf{bool}$$

21.4 The **(twisted) double cover** of the circle is defined as the type family $\mathcal{T} :\equiv \mathcal{D}(\mathsf{bool}, \mathsf{neg}) : \mathbf{S}^1 \to \mathcal{U}$, where $\mathsf{neg} : \mathsf{bool} \simeq \mathsf{bool}$ is the negation equivalence of Example 9.2.4.

(a) Show that $\neg(\prod_{(t:\mathbf{S}^1)} \mathcal{T}(t))$.

(b) Construct an equivalence $e : \mathbf{S}^1 \simeq \sum_{(t:\mathbf{S}^1)} \mathcal{T}(t)$ for which the triangle

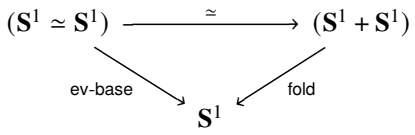


commutes.

There are 306 exercises.

Ljubljana, November 10, 2021

21.5 Construct an equivalence $(\mathbf{S}^1 \simeq \mathbf{S}^1) \simeq \mathbf{S}^1 + \mathbf{S}^1$ for which the triangle



commutes. Conclude that a univalent universe containing a circle is not a 1-type.

21.6 (a) Construct a family of equivalences

$$\prod_{(t:\mathbf{S}^1)} \big((t = t) \simeq \mathbb{Z}\big).$$

(b) Use Exercise 20.4 to show that $(\mathsf{id}_{\mathbf{S}^1} \sim \mathsf{id}_{\mathbf{S}^1}) \simeq \mathbb{Z}$.

(c) Use Exercise 13.5 (b) to show that

$$\mathsf{has\text{-}inverse}(\mathsf{id}_{\mathbf{S}^1}) \simeq \mathbb{Z},$$

and conclude that $\mathsf{has\text{-}inverse}(\mathsf{id}_{\mathbf{S}^1}) \neq \mathsf{is\text{-}equiv}(\mathsf{id}_{\mathbf{S}^1})$.

21.7 Consider a map $i : A \to \mathbf{S}^1$, and assume that i has a retraction. Construct an element of type

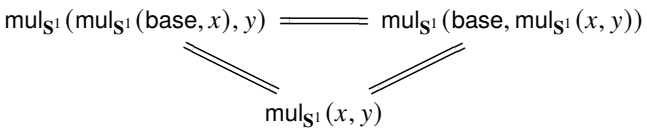
$$\mathsf{is\text{-}contr}(A) + \mathsf{is\text{-}equiv}(i).$$

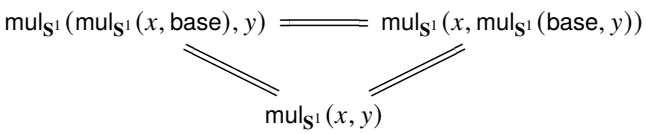
21.8 (a) Show that the multiplicative operation on the circle is associative, i.e. construct an identification

$$\mathsf{assoc}_{\mathbf{S}^1}(x, y, z) : \mathsf{mul}_{\mathbf{S}^1}(\mathsf{mul}_{\mathbf{S}^1}(x, y), z) = \mathsf{mul}_{\mathbf{S}^1}(x, \mathsf{mul}_{\mathbf{S}^1}(y, z))$$

for any $x, y, z : \mathbf{S}^1$.

(b) Show that the associator satisfies unit laws, in the sense that the following triangles commute:





There are 306 exercises.

Ljubljana, November 10, 2021

$$\begin{array}{ccc} \mathsf{mul}_{\mathbf{S}^1}(\mathsf{mul}_{\mathbf{S}^1}(x,y),\mathsf{base}) & = & \mathsf{mul}_{\mathbf{S}^1}(x,\mathsf{mul}_{\mathbf{S}^1}(y,\mathsf{base})) \\ & \searrow \quad \swarrow & \\ & \mathsf{mul}_{\mathbf{S}^1}(x,y). & \end{array}$$

(c) State the laws that compute

$$\begin{aligned} &\mathsf{assoc}_{\mathbf{S}^1}(\mathsf{base},\mathsf{base},x)\\ &\mathsf{assoc}_{\mathbf{S}^1}(\mathsf{base},x,\mathsf{base})\\ &\mathsf{assoc}_{\mathbf{S}^1}(x,\mathsf{base},\mathsf{base})\\ &\mathsf{assoc}_{\mathbf{S}^1}(\mathsf{base},\mathsf{base},\mathsf{base}). \end{aligned}$$

Note: the first three laws should be 3-cells and the last law should be a 4-cell. The laws are automatically satisfied, since the circle is a 1-type.

21.9 Construct the **Mac Lane pentagon** for the circle, i.e. show that the pentagon

$$\begin{array}{ccccc} \mathsf{mul}_{\mathbf{S}^1}(\mathsf{mul}_{\mathbf{S}^1}(\mathsf{mul}_{\mathbf{S}^1}(x,y),z),w) & = & \mathsf{mul}_{\mathbf{S}^1}(\mathsf{mul}_{\mathbf{S}^1}(x,y),\mathsf{mul}_{\mathbf{S}^1}(z,w)) & & \\ \| & & & \searrow & \\ \mathsf{mul}_{\mathbf{S}^1}(\mathsf{mul}_{\mathbf{S}^1}(x,\mathsf{mul}_{\mathbf{S}^1}(y,z)),w) & & & & \mathsf{mul}_{\mathbf{S}^1}(x,\mathsf{mul}_{\mathbf{S}^1}(y,\mathsf{mul}_{\mathbf{S}^1}(z,w))) \\ & \searrow & & \swarrow & \\ & & \mathsf{mul}_{\mathbf{S}^1}(x,\mathsf{mul}_{\mathbf{S}^1}(\mathsf{mul}_{\mathbf{S}^1}(y,z),w)) & & \end{array}$$

commutes for every $x, y, z, w : \mathbf{S}^1$.

21.10 Recall from Exercise 17.4 that if $f : A \to B$ is a surjective map, then the precomposition map

$$- \circ f : (B \to C) \to (A \to C)$$

is an embedding for every set C. Give an example of a surjective map $f : A \to B$, such that the precomposition function

$$- \circ f : (B \to \mathbf{S}^1) \to (A \to \mathbf{S}^1)$$

is *not* an embedding, showing that the condition that C is a set is essential.

22 Homotopy pullbacks

Suppose we are given a map $f : A \to B$, and type families P over A, and Q over B. Then any family of maps

$$g : \prod_{(x:A)} P(x) \to Q(f(x))$$

gives rise to a commuting square

$$\begin{array}{ccc} \sum_{(x:A)} P(x) & \xrightarrow{\mathsf{tot}_f(g)} & \sum_{(y:B)} Q(y) \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle \mathsf{pr}_1} \\ A & \xrightarrow[f]{} & B \end{array}$$

where $\mathsf{tot}_f(g)$ is defined as $\lambda(x, y).\, (f(x), g(x, y))$. In the main theorem of this chapter we show that g is a family of equivalences if and only if this square satisfies a certain universal property: the universal property of *pullback squares*.

Pullback squares are of interest because they appear in many situations. Cartesian products, fibers of maps, and substitutions can all be presented as pullbacks. Moreover, the fact that a family of maps $g : \prod_{(x:A)} P(x) \to Q(f(x))$ is a family of equivalences if and only if it induces a pullback square has the very useful corollary that a square of the form

$$\begin{array}{ccc} C & \longrightarrow & D \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle q} \\ A & \xrightarrow[f]{} & B \end{array}$$

is a pullback square if and only if the induced family of maps between the fibers

$$\prod_{(x:A)} \mathsf{fib}_p(x) \to \mathsf{fib}_q(f(x))$$

is a family of equivalences. This connection between pullbacks and *fiberwise equivalences* has an important role in the descent theorem in Section 25.

A second reason for studying pullback squares is that the dual notion of *pushouts* is an important tool to construct new types, including the n-spheres for arbitrary n. The duality of pullbacks and pushouts makes it possible to obtain proofs of many statements about pushouts from their dual statements about pullbacks.

22.1 The universal property of pullbacks

Definition 22.1.1 A **cospan** consists of three types A, X, and B, and maps $f : A \to X$ and $g : B \to X$.

Definition 22.1.2 Consider a cospan

$$A \xrightarrow{f} X \xleftarrow{g} B$$

and a type C. A **cone** on the cospan $A \to X \leftarrow B$ with **vertex** C consists of maps $p : C \to A$, $q : C \to B$ and a homotopy $H : f \circ p \sim g \circ q$ witnessing that the square

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

commutes. We write

$$\mathsf{cone}(C) :\equiv \textstyle\sum_{(p:C\to A)} \sum_{(q:C\to B)} f \circ p \sim g \circ q$$

for the type of cones with vertex C.

It is good practice to characterize the identity type of any type of importance. In the following lemma we give a characterization of the identity type of the type $\mathsf{cone}(C)$ of cones on $A \to X \leftarrow B$ with vertex C. Such characterizations are entirely routine in homotopy type theory.

Lemma 22.1.3 *Let (p, q, H) and (p', q', H') be cones on a cospan $f : A \to X \leftarrow B : g$, both with vertex C. Then the type $(p, q, H) = (p', q', H')$ is equivalent to the type of triples (K, L, M) consisting of*

$$K : p \sim p'$$
$$L : q \sim q'$$

and a homotopy $M : H \cdot (g \cdot L) \sim (f \cdot K) \cdot H'$ witnessing that the square

$$\begin{array}{ccc} f \circ p & \xrightarrow{f \cdot K} & f \circ p' \\ {\scriptstyle H}\downarrow & & \downarrow{\scriptstyle H'} \\ g \circ q & \xrightarrow[g \cdot L]{} & g \circ q' \end{array}$$

of homotopies commutes.

Proof By the fundamental theorem of identity types (Theorem 11.2.2) it suffices to show that the type

$$\textstyle\sum_{((p',q',H'):\sum_{(p':C\to A)} \sum_{(q':C\to B)} f\circ p' \sim g\circ q')} \sum_{(K:p\sim p')} \sum_{(L:q\sim q')} H\cdot(g\cdot L) \sim (f\cdot K)\cdot H'$$

is contractible. Using associativity of Σ-types and commutativity of cartesian products, it is easy to show that this type is equivalent to the type

$$\sum_{((p',K):\sum_{(p':C\to A)} p\sim p')} \sum_{((q',L):\sum_{(q':C\to B)} q\sim q')} \sum_{(H:f\circ p'\sim g\circ q')} H\cdot(g\cdot L) \sim (f\cdot K)\cdot H'$$

Now we observe that the types $\sum_{(p':C\to A)} p \sim p'$ and $\sum_{(q':C\to B)} q \sim q'$ are contractible, with centers of contraction

$$(p, \mathsf{refl\text{-}htpy}_p) : \sum_{(p':C'\to A)} p \sim p'$$
$$(q, \mathsf{refl\text{-}htpy}_q) : \sum_{(q':C'\to B)} q \sim q'.$$

Thus we apply Exercise 10.6 to see that the type of tuples $((p', K), (q', L), (H', M))$ is equivalent to the type

$$\sum_{(H':f\circ p'\sim g\circ q')} H \cdot \mathsf{refl\text{-}htpy}_{g\circ q} \sim \mathsf{refl\text{-}htpy}_{f\circ p} \cdot H'.$$

Of course, the type $H \cdot \mathsf{refl\text{-}htpy}_{g\circ q} \sim \mathsf{refl\text{-}htpy}_{f\circ p} \cdot H'$ is equivalent to the type $H \sim H'$, and $\sum_{(H':f\circ p\sim g\circ q)} H \sim H'$ is contractible. □

Given a cone with vertex C on a span $A \xrightarrow{f} X \xleftarrow{g} B$ and a map $h : C' \to C$, we construct a new cone with vertex C' in the following definition.

Definition 22.1.4 For any cone (p, q, H) with vertex C and any type C', we define a map

$$\mathsf{cone\text{-}map}(p, q, H) : (C' \to C) \to \mathsf{cone}(C')$$

by $h \mapsto (p \circ h, q \circ h, H \circ h)$.

Definition 22.1.5 We say that a commuting square

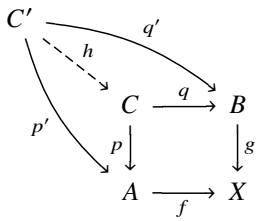
$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with $H : f \circ p \sim g \circ q$ is a **pullback square**, or that it is **cartesian**, if it satisfies the **universal property** of pullbacks, which asserts that the map

$$\mathsf{cone\text{-}map}(p, q, H) : (C' \to C) \to \mathsf{cone}(C')$$

is an equivalence for every type C'.

We often indicate the universal property with a diagram as follows:



since the universal property states that for every cone (p', q', H') with vertex C', the type of pairs (h, α) consisting of $h : C' \to C$ equipped with $\alpha : \mathsf{cone\text{-}map}((p, q, H), h) = (p', q', H')$ is contractible by Theorem 10.4.6.

As a corollary we obtain the following characterization of the universal property of pullbacks.

Lemma 22.1.6 *Consider a commuting square*

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with $H : f \circ p \sim g \circ q$ Then the following are equivalent:

(i) *The square is a pullback square.*

(ii) *For every type C' and every cone (p', q', H') with vertex C', the type of quadruples (h, K, L, M) consisting of a map $h : C' \to C$, homotopies*

$$K : p \circ h \sim p'$$
$$L : q \circ h \sim q',$$

and a homotopy $M : (H \cdot h) \cdot (g \cdot L) \sim (f \cdot K) \cdot H'$ witnessing that the square

$$\begin{array}{ccc} f \circ p \circ h & \xrightarrow{f \cdot K} & f \circ p' \\ {\scriptstyle H \cdot h}\downarrow & & \downarrow{\scriptstyle H'} \\ g \circ q \circ h & \xrightarrow[g \cdot L]{} & g \circ q' \end{array}$$

commutes, is contractible.

Proof The map $\mathsf{cone\text{-}map}(p, q, H)$ is an equivalence if and only if its fibers are contractible. By Lemma 22.1.3 it follows that the fibers of $\mathsf{cone\text{-}map}(p, q, H)$ are equivalent the the described type of quadruples (h, K, L, M). □

In the following lemma we establish the uniqueness of pullbacks up to equivalence via a *3-for-2 property* for pullbacks.

Lemma 22.1.7 *Consider the squares*

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array} \qquad \begin{array}{ccc} C' & \xrightarrow{q'} & B \\ {\scriptstyle p'}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with homotopies $H : f \circ p \sim g \circ q$ *and* $H' : f \circ p' \sim g \circ q'$*. Furthermore, suppose we have a map* $h : C' \to C$ *equipped with*

$$\begin{aligned} K &: p \circ h \sim p' \\ L &: q \circ h \sim q' \\ M &: (H \cdot h) \cdot (g \cdot L) \sim (f \cdot K) \cdot H'. \end{aligned}$$

If any two of the following three properties hold, so does the third:

(i) *C is a pullback.*
(ii) *C' is a pullback.*
(iii) *h is an equivalence.*

Proof By the characterization of the identity type of $\mathsf{cone}(C')$ given in Lemma 22.1.3 we obtain an identification

$$\mathsf{cone\text{-}map}((p, q, H), h) = (p', q', H')$$

from the triple (K, L, M). Let D be a type, and let $k : D \to C'$ be a map. We observe that

$$\begin{aligned} \mathsf{cone\text{-}map}((p, q, H), (h \circ k)) &\doteq (p \circ (h \circ k), q \circ (h \circ k), H \circ (h \circ k)) \\ &\doteq ((p \circ h) \circ k, (q \circ h) \circ k, (H \circ h) \circ k) \\ &\doteq \mathsf{cone\text{-}map}(\mathsf{cone\text{-}map}((p, q, H), h), k) \\ &= \mathsf{cone\text{-}map}((p', q', H'), k). \end{aligned}$$

Thus we see that the triangle

$$\begin{array}{ccc} (D \to C') & \xrightarrow{h \circ -} & (D \to C) \\ & {\scriptstyle \mathsf{cone\text{-}map}(p', q', H')} \searrow \quad \swarrow {\scriptstyle \mathsf{cone\text{-}map}(p, q, H)} & \\ & \mathsf{cone}(D) & \end{array}$$

commutes. Therefore it follows from the 3-for-2 property of equivalences established in Exercise 9.4, that if any two of the maps in this triangle is an

equivalence, then so is the third. Now the claim follows from the fact that h is an equivalence if and only if $h \circ - : (D \to C') \to (D \to C)$ is an equivalence for any type D, which was established in Exercise 13.12 (d). □

Pullbacks are not only unique in the sense that any two pullbacks of the same cospan are equivalent, they are *uniquely unique* in the sense that the type of quadruples (h, K, L, M) as in Lemma 22.1.7 is contractible.

Corollary 22.1.8 *Suppose both commuting squares*

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array} \qquad \begin{array}{ccc} C' & \xrightarrow{q'} & B \\ {\scriptstyle p'}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with homotopies $H : f \circ p \sim g \circ q$ and $H' : f \circ p' \sim g \circ q'$ are pullback squares. Then the type of quadruples (e, K, L, M) consisting of an equivalence $e : C' \simeq C$ equipped with

$$\begin{aligned} K &: p \circ e \sim p' \\ L &: q \circ e \sim q' \\ M &: (H \cdot h) \centerdot (g \cdot L) \sim (f \cdot K) \centerdot H'. \end{aligned}$$

is contractible.

Proof We have seen that the type of quadruples (h, K, L, M) is equivalent to the fiber of $\mathsf{cone\text{-}map}(p, q, H)$ at (p', q', H'). By Lemma 22.1.7 it follows that h is an equivalence. Since $\mathsf{is\text{-}equiv}(h)$ is a proposition by Exercise 13.4, and hence contractible as soon as it is inhabited, it follows that the type of quadruples (e, K, L, M) is contractible. □

Corollary 22.1.9 *For any two maps $f : A \to X$ and $g : B \to X$, and any universe $\mathcal{U}$, the type*

$$\textstyle\sum_{(C:\mathcal{U})} \sum_{(c:\mathsf{cone}(f,g,C))} \prod_{(C':\mathcal{U})} \mathsf{is\text{-}equiv}(\mathsf{cone\text{-}map}_{C'}(c))$$

of pullbacks in $\mathcal{U}$, is a proposition.

Proof It is straightforward to see that the type of identifications

$$(C, (p, q, H), u) = (C', (p', q', H'), u')$$

of any two pullbacks is equivalent to the type of quadruples (e, K, L, M) as in Corollary 22.1.8. Since Corollary 22.1.8 claims that this type of quadruples is contractible, the claim follows. □

22.2 Canonical pullbacks

For every cospan we can construct a *canonical pullback*.

Definition 22.2.1 Let $f : A \to X$ and $g : B \to X$ be maps. Then we define

$$\begin{aligned} A \times_X B &:\equiv \textstyle\sum_{(x:A)} \sum_{(y:B)} f(x) = g(y) \\ \pi_1 &:\equiv \mathsf{pr}_1 && : A \times_X B \to A \\ \pi_2 &:\equiv \mathsf{pr}_1 \circ \mathsf{pr}_2 && : A \times_X B \to B \\ \pi_3 &:\equiv \mathsf{pr}_2 \circ \mathsf{pr}_2 && : f \circ \pi_1 \sim g \circ \pi_2. \end{aligned}$$

The type $A \times_X B$ is called the **canonical pullback** of f and g.

Note that $A \times_X B$ depends on f and g, although this dependency is not visible in the notation.

Remark 22.2.2 Given (x, y, p) and (x', y', p') in the canonical pullback $A \times_X B$, the identity type $(x, y, p) = (x', y', p')$ is equivalent to the type of triples (α, β, γ) consisting of $\alpha : x = x'$, $\beta : y = y'$, and an identification $\gamma : p \cdot \mathsf{ap}_g(\beta) = \mathsf{ap}_f(\alpha) \cdot p'$ witnessing that the square

$$\begin{array}{ccc} f(x) & \xrightarrow{\mathsf{ap}_f(\alpha)} & f(x') \\ {\scriptstyle p}\| & & \|{\scriptstyle p'} \\ g(y) & \xrightarrow[\mathsf{ap}_g(\beta)]{} & g(y') \end{array}$$

commutes. The proof of this fact is similar to the proof of Lemma 22.1.3.

Theorem 22.2.3 *Given maps $f : A \to X$ and $g : B \to X$, the commuting square*

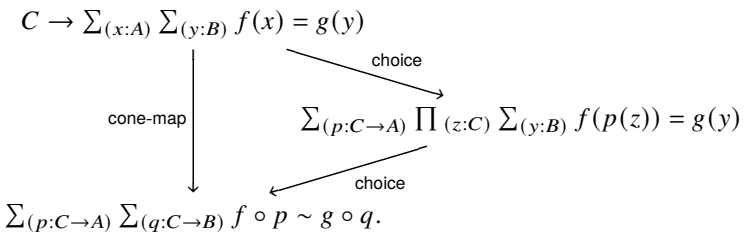
$$\begin{array}{ccc} A \times_X B & \xrightarrow{\pi_2} & B \\ {\scriptstyle \pi_1}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X, \end{array}$$

is a pullback square.

Proof Let C be a type. Our goal is to show that the map

$$\mathsf{cone\text{-}map}(\pi_1, \pi_2, \pi_3) : (C \to A \times_X B) \to \mathsf{cone}(C)$$

is an equivalence. Note that we have the commuting triangle



In this triangle the functions choice are equivalences by Theorem 13.2.1. Therefore, their composite is an equivalence. □

The following corollary is now a special case of **??**, where we make sure that $f : A \to X$ and $g : B \to X$ are both maps in $\mathcal{U}$.

Corollary 22.2.4 *For any two maps $f : A \to X$ and $g : B \to X$ in $\mathcal{U}$, the type*

$$\sum_{(C:\mathcal{U})} \sum_{(c:\mathsf{cone}(f,g,C))} \prod_{(C':\mathcal{U})} \mathsf{is\text{-}equiv}(\mathsf{cone\text{-}map}_{C'}(c))$$

of pullbacks in $\mathcal{U}$, is contractible.

Definition 22.2.5 Given a commuting square

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with $H : f \circ p \sim g \circ q$, we define the **gap map**

$$\mathsf{gap}(p, q, H) : C \to A \times_X B$$

by $\lambda z.\, (p(z), q(z), H(z))$.

The following theorem provides a useful characterization of pullback squares, because in many situations it is easier to show that the gap map is an equivalence.

Theorem 22.2.6 *Consider a commuting square*

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with $H : f \circ p \sim g \circ q$. The following are equivalent:

(i) *The square is a pullback square*

(ii) *There is a term of type*

$$\mathsf{is\text{-}pullback}(p, q, H) :\equiv \mathsf{is\text{-}equiv}(\mathsf{gap}(p, q, H)).$$

Proof Observe that we are in the situation of Lemma 22.1.7. Indeed, we have two commuting squares

$$\begin{array}{ccc} A \times_X B & \xrightarrow{\pi_2} & B \\ {\scriptstyle \pi_2}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array} \qquad\qquad \begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X, \end{array}$$

and we have the gap map $\mathsf{gap} : C \to A \times_X B$, which comes equipped with the homotopies

$$\begin{aligned} K &: \pi_1 \circ \mathsf{gap} \sim p & K &:\equiv \lambda z.\, \mathsf{refl}_{p(z)} \\ L &: \pi_2 \circ \mathsf{gap} \sim q & L &:\equiv \lambda z.\, \mathsf{refl}_{q(z)} \\ M &: (\pi_3 \cdot \mathsf{gap}) \cdot (g \cdot L) \sim (f \cdot K) \cdot H & M &:\equiv \lambda z.\, \mathsf{right\text{-}unit}(H(z)). \end{aligned}$$

Since $A \times_X B$ is shown to be a pullback in Theorem 22.2.3, it follows from Lemma 22.1.7 that C is a pullback if and only if the gap map is an equivalence. □

22.3 Cartesian products and fiberwise products as pullbacks

An important special case of pullbacks occurs when the cospan is of the form

$$A \longrightarrow \mathbf{1} \longleftarrow B.$$

In this case, the pullback is just the *cartesian product*.

Lemma 22.3.1 *Let A and B be types. Then the square*

$$\begin{array}{ccc} A \times B & \xrightarrow{\mathsf{pr}_2} & B \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle \mathsf{const}_\star} \\ A & \xrightarrow[\mathsf{const}_\star]{} & \mathbf{1} \end{array}$$

which commutes by the homotopy $\mathsf{const}_{\mathsf{refl}_\star}$ is a pullback square.

Proof By Theorem 22.2.6 it suffices to show that

$$\mathsf{gap}(\mathsf{pr}_1, \mathsf{pr}_2, \lambda(a, b).\, \mathsf{refl}_\star)$$

is an equivalence. Its inverse is the map $\lambda(a, b, p).\, (a, b)$. □

The following generalization of Lemma 22.3.1 is the reason why pullbacks are sometimes called **fiber products**.

Theorem 22.3.2 *Let P and Q be families over a type X. Then the square*

$$\begin{array}{ccc} \sum_{(x:X)} P(x) \times Q(x) & \xrightarrow{\lambda(x,(p,q)).\,(x,q)} & \sum_{(x:X)} Q(x) \\ {\scriptstyle \lambda(x,(p,q)).\,(x,p)}\downarrow & & \downarrow{\scriptstyle \mathsf{pr}_1} \\ \sum_{(x:X)} P(x) & \xrightarrow[\mathsf{pr}_1]{} & X, \end{array}$$

which commutes by the homotopy

$$H :\equiv \lambda(x,(p,q)).\,\mathsf{refl}_x,$$

is a pullback square.

Proof By Theorem 22.2.6 it suffices to show that the gap map is an equivalence. The gap map is homotopic to the function

$$\lambda(x,(p,q)).\,((x,p),(x,q),\mathsf{refl}_x).$$

It is easy to check that this function is an equivalence. I ts inverse is the map

$$\lambda((x,p),(y,q),\alpha).\,(y,(\mathsf{tr}_P(\alpha,p),q)).$$ □

Corollary 22.3.3 *For any $f : A \to X$ and $g : B \to X$, the square*

$$\begin{array}{ccc} \sum_{(x:X)} \mathsf{fib}_f(x) \times \mathsf{fib}_g(x) & \xrightarrow{\lambda(x,((a,p),(b,q))).\,b} & B \\ {\scriptstyle \lambda(x,((a,p),(b,q))).\,a}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

is a pullback square.

22.4 Fibers of maps as pullbacks

Lemma 22.4.1 *For any function $f : A \to B$, and any $b : B$, consider the square*

$$\begin{array}{ccc} \mathsf{fib}_f(b) & \xrightarrow{\mathsf{const}_\star} & \mathbf{1} \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle \mathsf{const}_b} \\ A & \xrightarrow[f]{} & B \end{array}$$

which commutes by $\mathsf{pr}_2 : \prod_{(t:\mathsf{fib}_f(b))} f(\mathsf{pr}_1(t)) = b$. This is a pullback square.

Proof By Theorem 22.2.6 it suffices to show that the gap map is an equivalence. The gap map is homotopic to the function

$$\mathsf{tot}(()\lambda x.\,\lambda p.\,(\star,p))$$

The map $\lambda x.\, \lambda p.\, (\star, p)$ is a family of equivalences by Exercise 10.6, so it induces an equivalence on total spaces by Theorem 11.1.3. □

Corollary 22.4.2 *For any type family B over A and any $a : A$ the square*

$$\begin{array}{ccc} B(a) & \xrightarrow{\mathsf{const}_\star} & \mathbf{1} \\ {\scriptstyle \lambda y.\,(a,y)}\downarrow & & \downarrow{\scriptstyle \lambda \star.\, a} \\ \sum_{(x:A)} B(x) & \xrightarrow[\mathsf{pr}_1]{} & A \end{array}$$

is a pullback square.

Proof To see this, note that the triangle

$$\begin{array}{ccc} B(a) & \xrightarrow{\lambda b.\,((a,b),\mathsf{refl}_a)} & \mathsf{fib}_{\mathsf{pr}_1}(a) \\ & {\scriptstyle \mathsf{gap}}\searrow \quad \swarrow{\scriptstyle \mathsf{gap}} & \\ & \left(\sum_{(x:A)} B(x)\right) \times_A \mathbf{1}. & \end{array}$$

Since the top map is an equivalence by Exercise 10.8, and the map on the right is an equivalence by Lemma 22.4.1, it follows that the map on the left is an equivalence. The claim follows. □

22.5 Families of equivalences

Lemma 22.5.1 *Let $f : A \to B$, and let Q be a type family over B. Then the square*

$$\begin{array}{ccc} \sum_{(x:A)} Q(f(x)) & \xrightarrow{\lambda(x,q).\,(f(x),q)} & \sum_{(y:B)} Q(b) \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle \mathsf{pr}_1} \\ A & \xrightarrow[f]{} & B \end{array}$$

commutes by $H :\equiv \lambda(x,q).\, \mathsf{refl}_{f(x)}$. This is a pullback square.

Proof By Theorem 22.2.6 it suffices to show that the gap map is an equivalence. The gap map is homotopic to the function

$$\lambda(x,q).\, (x, (f(x), q), \mathsf{refl}_{f(x)}).$$

The inverse of this map is given by $\lambda(x,((y,q),p)).\, (x, \mathsf{tr}_Q(p^{-1}, q))$, and it is straightforward to see that these maps are indeed mutual inverses. □

Theorem 22.5.2 *Let $f : A \to B$, and let $g : \prod_{(a:A)} P(a) \to Q(f(a))$ be a family of maps. The following are equivalent:*

(i) *The commuting square*

$$\begin{array}{ccc} \sum_{(a:A)} P(a) & \xrightarrow{\mathsf{tot}_f\,(g)} & \sum_{(b:B)} Q(b) \\ \downarrow & & \downarrow \\ A & \xrightarrow[f]{} & B \end{array}$$

is a pullback square.

(ii) *g is a family of equivalences.*

Proof The gap map is homotopic to the composite

$$\sum_{(x:A)} P(x) \xrightarrow{\mathsf{tot}\,(g)} \sum_{(x:A)} Q(f(x)) \xrightarrow{\mathsf{gap}'} A \times_B \left(\sum_{(y:B)} Q(y) \right)$$

where gap' is the gap map for the square in Lemma 22.5.1. Since gap' is an equivalence, it follows by Exercise 9.4 and Theorem 11.1.3 that the gap map is an equivalence if and only if g is a family of equivalences. □

Our goal is now to extend Theorem 22.5.2 to arbitrary pullback squares. Note that every commuting square

$$\begin{array}{ccc} A & \xrightarrow{h} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ X & \xrightarrow[i]{} & Y \end{array}$$

with $H : i \circ f \; g \circ h$ induces a map

$$\mathsf{fib\text{-}sq} : \prod_{(x:X)} \mathsf{fib}_f\,(x) \to \mathsf{fib}_g\,(f(x))$$

on the fibers, by

$$\mathsf{fib\text{-}sq}(x, (a, p)) :\equiv (h(a), H(a)^{-1} \cdot \mathsf{ap}_i(p)).$$

Theorem 22.5.3 *Consider a commuting square*

$$\begin{array}{ccc} A & \xrightarrow{h} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ X & \xrightarrow[i]{} & Y \end{array}$$

with $H : i \circ f \; g \circ h$. The following are equivalent:

(i) *The square is a pullback square.*

(ii) *The induced map on fibers*

$$\mathsf{fib\text{-}sq} : \prod_{(x:X)} \mathsf{fib}_f(x) \to \mathsf{fib}_g(f(x))$$

is a family of equivalences.

Proof First we observe that the square

$$\begin{array}{ccc} \sum_{(x:X)} \mathsf{fib}_f(x) & \xrightarrow{\mathsf{tot}(\mathsf{fib\text{-}sq})} & \sum_{(x:X)} \mathsf{fib}_g(f(x)) \\ \simeq \downarrow & & \downarrow \mathsf{tot}(\mathsf{tot}(\mathsf{inv})) \\ A & \xrightarrow[\mathsf{gap}]{} & X \times_Y B \end{array}$$

commutes. To construct such a homotopy, we need to construct an identification

$$(f(a), h(a), H(a)) = (x, h(a), (H(a)^{-1} \cdot \mathsf{ap}_i(p))^{-1})$$

for every $x : X$, $a : A$, and $p : f(a) = x$. This is shown by path induction on $p : f(a) = x$. Thus, it suffices to show that

$$(f(a), h(a), H(a)) = (f(a), h(a), (H(a)^{-1} \cdot \mathsf{refl}_{i(f(a))})^{-1}),$$

which is a routine exercise.

Now we note that the left and right maps in this square are both equivalences. Therefore it follows that the top map is an equivalence if and only if the bottom map is. The claim now follows by Theorem 11.1.3. □

Corollary 22.5.4 *Consider a pullback square*

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X. \end{array}$$

If g is a k-truncated map, then so is p. In particular, if g is an embedding then so is p.

Proof Since the square is assumed to be a pullback square, it follows from Theorem 22.5.3 that for each $x : A$, the fiber $\mathsf{fib}_p(x)$ is equivalent to the fiber $\mathsf{fib}_g(f(x))$, which is k-truncated. Since k-truncated types are closed under equivalences by Proposition 12.4.5, it follows that p is a k-truncated map. □

Corollary 22.5.5 *Consider a commuting square*

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X. \end{array}$$

and suppose that g is an equivalence. Then the following are equivalent:

(i) *The square is a pullback square.*
(ii) *The map $p : C \to A$ is an equivalence.*

Proof If the square is a pullback square, then by Theorem 22.5.2 the fibers of p are equivalent to the fibers of g, which are contractible by Theorem 10.4.6. Thus it follows that p is a contractible map, and hence that p is an equivalence.

If p is an equivalence, then by Theorem 10.4.6 both $\mathsf{fib}_p(x)$ and $\mathsf{fib}_g(f(x))$ are contractible for any $x : X$. It follows by Exercise 10.3 that the induced map $\mathsf{fib}_p(x) \to \mathsf{fib}_g(f(x))$ is an equivalence. Thus we apply Theorem 22.5.3 to conclude that the square is a pullback. □

Theorem 22.5.6 *Consider a diagram of the form*

$$\begin{array}{ccc} A & & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ X & \xrightarrow[h]{} & Y. \end{array}$$

Then the type of triples (i, H, p) consisting of a map $i : A \to B$, a homotopy $H : h \circ f \sim g \circ i$, and a term p witnessing that the square

$$\begin{array}{ccc} A & \xrightarrow{i} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ X & \xrightarrow[h]{} & Y. \end{array}$$

is a pullback square, is equivalent to the type of families of equivalences

$$\textstyle\prod_{(x:X)} \mathsf{fib}_f(x) \simeq \mathsf{fib}_g(h(x)).$$

Corollary 22.5.7 *Let $h : X \to Y$ be a map, and let P and Q be families over X and Y, respectively. Then the type of triples (i, H, p) consisting of a map*

$$i : \left(\textstyle\sum_{(x:X)} P(x)\right) \to \left(\textstyle\sum_{(y:Y)} Q(y)\right),$$

a homotopy $H : h \circ \mathsf{pr}_1 \sim \mathsf{pr}_1 \circ i$, and a term p witnessing that the square

$$\begin{array}{ccc} \sum_{(x:X)} P(x) & \xrightarrow{i} & \sum_{(y:Y)} Q(y) \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle \mathsf{pr}_1} \\ X & \xrightarrow[h]{} & Y. \end{array}$$

is a pullback square, is equivalent to the type of families of equivalences

$$\textstyle\prod_{(x:X)} P(x) \simeq Q(h(x)).$$

One useful application of the connection between pullbacks and families of equivalences is the following theorem, which is also called the **pasting property** of pullbacks.

Theorem 22.5.8 *Consider a commuting diagram of the form*

$$\begin{array}{ccccc} A & \xrightarrow{k} & B & \xrightarrow{l} & C \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} & & \downarrow{\scriptstyle h} \\ X & \xrightarrow[i]{} & Y & \xrightarrow[j]{} & Z \end{array}$$

with homotopies $H : i \circ f \sim g \circ k$ and $K : j \circ g \sim h \circ l$, and the homotopy

$$(j \cdot H) \centerdot (K \cdot k) : j \circ i \circ f \sim h \circ l \circ k$$

witnessing that the outer rectangle commutes. Furthermore, suppose that the square on the right is a pullback square. Then the following are equivalent:

(i) *The square on the left is a pullback square.*
(ii) *The outer rectangle is a pullback square.*

Proof The commutativity of the two squares and the outer rectangle induces a commuting triangle

$$\begin{array}{ccc} \mathsf{fib}_f(x) & \xrightarrow{\mathsf{fib\text{-}sq}_{(f,k,H)}(x)} & \mathsf{fib}_g(i(x)) \\ & {\scriptstyle \mathsf{fib\text{-}sq}_{f,l\circ k,(j\cdot H)\centerdot(K\cdot k)}(x)} \searrow \quad \swarrow {\scriptstyle \mathsf{fib\text{-}sq}_{(g,l,K)}(i(x))} & \\ & \mathsf{fib}_h(j(i(x))). & \end{array}$$

A homotopy witnessing that the triangle commutes is constructed by a routine calculation.

Since the triangle commutes, and since the map $\mathsf{fib\text{-}sq}_{(g,l,K)}(i(x))$ is an equivalence for each $x : X$ by Theorem 22.5.3, it follows by the 3-for-2 property of equivalences that for each $x : X$ the top map in the triangle is an equivalence if and only if the left map is an equivalence. The claim now follows by a second application of Theorem 22.5.3. □

22.6 Descent theorems for coproducts and Σ-types

Theorem 22.6.1 *Consider maps $f : A' \to A$ and $g : B' \to B$, a map $h : X' \to X$, and commuting squares of the form*

$$\begin{array}{ccc} A' & \longrightarrow & X' \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle h} \\ A & \longrightarrow & X \end{array} \qquad \begin{array}{ccc} B' & \longrightarrow & X' \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ B & \longrightarrow & X. \end{array}$$

Then the following are equivalent:

(i) *Both squares are pullback squares.*
(ii) *The commuting square*

$$\begin{array}{ccc} A' + B' & \longrightarrow & X' \\ {\scriptstyle f+g}\downarrow & & \downarrow{\scriptstyle h} \\ A + B & \longrightarrow & X \end{array}$$

is a pullback square.

Proof By Theorem 22.5.3 it suffices to show that the following are equivalent:

(i) For each $x : A$ the map

$$\mathsf{fib\text{-}sq} : \mathsf{fib}_f(x) \to \mathsf{fib}_h(\alpha_A(x))$$

is an equivalence, and for each $y : B$ the map

$$\mathsf{fib\text{-}sq} : \mathsf{fib}_g(y) \to \mathsf{fib}_h(\alpha_B(y))$$

is an equivalence.
(ii) For each $t : A + B$ the map

$$\mathsf{fib\text{-}sq} : \mathsf{fib}_{f+g}(t) \to \mathsf{fib}_h(\alpha(t))$$

is an equivalence.

By the dependent universal property of coproducts, the second claim is equivalent to the claim that both for each $x : A$ the map

$$\mathsf{fib\text{-}sq} : \mathsf{fib}_{f+g}(\mathsf{inl}(x)) \to \mathsf{fib}_h(\alpha_A(x))$$

is an equivalence, and for each $y : B$, the map

$$\mathsf{fib\text{-}sq} : \mathsf{fib}_{f+g}(\mathsf{inr}(y)) \to \mathsf{fib}_h(\alpha_B(y))$$

is an equivalence.

We claim that there is a commuting triangle

$$\begin{array}{ccc} \mathsf{fib}_f(x) & \longrightarrow & \mathsf{fib}_{f+g}(\mathsf{inl}(x)) \\ & \searrow \quad \swarrow & \\ & \mathsf{fib}_h(\alpha_A(x)) & \end{array}$$

for every $x : A$. To see that the triangle commutes, we need to construct an identification

The top map is given by

$$(a', p) \mapsto (\mathsf{inl}(a'), \mathsf{ap}_{\mathsf{inl}}(p)).$$

The triangle then commutes by the homotopy

$$(a', p) \mapsto \mathsf{eq\text{-}pair}(\mathsf{refl}, \mathsf{ap}_{\mathsf{concat}(H(a')^{-1})}(\mathsf{ap\text{-}comp}_{[\alpha_A, \alpha_B], inl}))$$

We note that the top map is an equivalence, so it follows by the 3-for-2 property of equivalences that the left map is an equivalence if and only if the right map is an equivalence.

Similarly, there is a commuting triangle

$$\begin{array}{ccc} \mathsf{fib}_g(y) & \longrightarrow & \mathsf{fib}_{f+g}(\mathsf{inr}(y)) \\ & \searrow \quad \swarrow & \\ & \mathsf{fib}_h(\alpha_B(y)) & \end{array}$$

in which the top map is an equivalence, completing the proof. □

In the following corollary we conclude that coproducts distribute over pullbacks.

Corollary 22.6.2 *Consider a cospan of the form*

$$\begin{array}{ccc} & & Y \\ & & \downarrow \\ A + B & \longrightarrow & X. \end{array}$$

Then there is an equivalence

$$(A + B) \times_X Y \simeq (A \times_X Y) + (B \times_X Y).$$

Theorem 22.6.3 *Consider a family of maps $f_i : A'_i \to A_i$ indexed by a type I, a map $h : X' \to X$, and a commuting square*

$$\begin{array}{ccc} A'_i & \longrightarrow & X' \\ {\scriptstyle f_i}\downarrow & & \downarrow{\scriptstyle h} \\ A_i & \xrightarrow[\alpha_i]{} & X \end{array}$$

for each $i : I$. Then the following are equivalent:

(i) *For each $i : I$ the square is a pullback square.*

(ii) *The commuting square*

$$\begin{array}{ccc} \sum_{(i:I)} A'_i & \longrightarrow & X' \\ {\scriptstyle \mathsf{tot}(f)}\downarrow & & \downarrow{\scriptstyle h} \\ \sum_{(i:I)} A_i & \xrightarrow{\alpha} & X \end{array}$$

is a pullback square.

Proof By Theorem 22.5.3 it suffices to show that the following are equivalent for each $i : I$ and $a : A_i$:

(i) The map

$$\mathsf{fib\text{-}sq} : \mathsf{fib}_{f_i}(a) \to \mathsf{fib}_g(\alpha_i(a))$$

is an equivalence.

(ii) The map

$$\mathsf{fib\text{-}sq} : \mathsf{fib}_{\mathsf{tot}(f)}(i, a) \to \mathsf{fib}_g(\alpha_i(a))$$

is an equivalence.

To see this, note that we have a commuting triangle

$$\begin{array}{ccc} \mathsf{fib}_{f_i}(a) & \longrightarrow & \mathsf{fib}_{\mathsf{tot}(f)}(i,a) \\ & \searrow \quad \swarrow & \\ & \mathsf{fib}_g(\alpha_i(a)), & \end{array}$$

where the top map is an equivalence by Lemma 11.1.2. Therefore the claim follows by the 3-for-2 property of equivalences. □

In the following corollary we conclude that Σ distributes over coproducts.

Corollary 22.6.4 *Consider a cospan of the form*

$$\begin{array}{ccc} & & Y \\ & & \downarrow \\ \sum_{(i:I)} A_i & \longrightarrow & X. \end{array}$$

Then there is an equivalence

$$\left(\sum_{(i:I)} A_i\right) \times_X Y \simeq \sum_{(i:I)} (A_i \times_X Y).$$

Exercises

22.1 (a) Show that the square

$$\begin{array}{ccc} (x = y) & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow{\scriptstyle \mathsf{const}_y} \\ \mathbf{1} & \xrightarrow[\mathsf{const}_x]{} & A \end{array}$$

is a pullback square.

(b) Show that the square

$$\begin{array}{ccc} (x = y) & \xrightarrow{\mathsf{const}_x} & A \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle \delta_A} \\ \mathbf{1} & \xrightarrow[\mathsf{const}_{(x,y)}]{} & A \times A \end{array}$$

is a pullback square, where $\delta_A : A \to A \times A$ is the diagonal of A, defined in Exercise 12.5.

22.2 In this exercise we give an alternative characterization of the notion of k-truncated map, compared to Theorem 12.4.7. Given a map $f : A \to X$ define the **diagonal** of f to be the map $\delta_f : A \to A \times_X A$ given by $x \mapsto (x, x, \mathsf{refl}_{f(x)})$.

(a) Construct an equivalence

$$\mathsf{fib}_{\delta_f}((x, y, p)) \simeq \mathsf{fib}_{\mathsf{ap}_f}(p)$$

to show that the square

$$\begin{array}{ccc} \mathsf{fib}_{\mathsf{ap}_f}(p) & \xrightarrow{\mathsf{const}_x} & A \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle \delta_f} \\ \mathbf{1} & \xrightarrow[\mathsf{const}_{(x,y,p)}]{} & A \times_X A \end{array}$$

is a pullback square, for every $x, y : A$ and $p : f(x) = f(y)$.

(b) Show that a map $f : A \to X$ is $(k+1)$-truncated if and only if δ_f is k-truncated.

Conclude that f is an embedding if and only if δ_f is an equivalence.

22.3 Consider a commuting square

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with $H : f \circ p \sim g \circ q$. Show that this square is a pullback square if and only if the square

$$\begin{array}{ccc} C & \xrightarrow{p} & A \\ {\scriptstyle q}\downarrow & & \downarrow{\scriptstyle f} \\ B & \xrightarrow[g]{} & X \end{array}$$

with $H^{-1} : g \circ q \sim f \circ p$ is a pullback square.

22.4 Show that any square of the form

$$\begin{array}{ccc} C & \longrightarrow & B \\ \downarrow & & \downarrow \\ \emptyset & \longrightarrow & X \end{array}$$

commutes and is a pullback square. This is the *descent property* of the empty type.

22.5 Consider a commuting square

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with $H : f \circ p \sim g \circ q$. Show that the following are equivalent:

(i) The square is a pullback square.

(ii) For every type T, the commuting square

$$\begin{array}{ccc} C^T & \xrightarrow{q\circ -} & B^T \\ {\scriptstyle p\circ -}\downarrow & & \downarrow{\scriptstyle g\circ -} \\ A^T & \xrightarrow[f\circ -]{} & X^T \end{array}$$

is a pullback square.

Note: property (ii) is really just a rephrasing of the universal property of pullbacks.

22.6 Consider a commuting square

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with $H : f \circ p \sim g \circ q$. Show that the following are equivalent:

(i) The square is a pullback square.

(ii) The square

$$\begin{array}{ccc} C & \xrightarrow{g\circ q} & X \\ {\scriptstyle \lambda x.\,(p(x),q(x))}\downarrow & & \downarrow{\scriptstyle \delta_X} \\ A\times B & \xrightarrow[f\times g]{} & X\times X \end{array}$$

which commutes by $\lambda z.\,\mathsf{eq\text{-}pair}(H(z),\mathsf{refl}_{g(q(z))})$ is a pullback square.

22.7 Consider two commuting squares

$$\begin{array}{ccc} C_1 & \longrightarrow & B_1 \\ \downarrow & & \downarrow \\ A_1 & \longrightarrow & X_1 \end{array} \qquad \begin{array}{ccc} C_2 & \longrightarrow & B_2 \\ \downarrow & & \downarrow \\ A_2 & \longrightarrow & X_2. \end{array}$$

(a) Show that if both squares are pullback squares, then the square

$$\begin{array}{ccc} C_1\times C_2 & \longrightarrow & B_1\times B_2 \\ \downarrow & & \downarrow \\ A_1\times A_2 & \longrightarrow & X_1\times X_2. \end{array}$$

is also a pullback square.

(b) Show that if there are terms $t_1 : A_1 \times_{X_1} B_1$ and $t_2 : A_2 \times_{X_2} B_2$, then the converse of (a) also holds.

22.8 Consider for each $i : I$ a pullback square

$$\begin{array}{ccc} C_i & \xrightarrow{q_i} & B_i \\ {\scriptstyle p_i}\downarrow & & \downarrow{\scriptstyle g_i} \\ A_i & \xrightarrow[f_i]{} & X_i \end{array}$$

with $H_i : f_i \circ p_i \sim g_i \circ q_i$. Show that the commuting square

$$\begin{array}{ccc} \prod_{(i:I)} C_i & \longrightarrow & \prod_{(i:I)} B_i \\ \downarrow & & \downarrow \\ \prod_{(i:I)} A_i & \longrightarrow & \prod_{(i:I)} X_i \end{array}$$

is a pullback square.

22.9 Let $f : A \to B$ be a map. Show that the following are equivalent:

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

(i) The commuting square

$$\begin{array}{ccc} A & \longrightarrow & \|A\| \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \|f\|} \\ B & \longrightarrow & \|B\|. \end{array}$$

is a pullback square.

(ii) There is a term of type $A \to \mathsf{is\text{-}equiv}(f)$.

(iii) The commuting square

$$\begin{array}{ccc} A \times A & \xrightarrow{f \times f} & B \times B \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle \mathsf{pr}_1} \\ A & \xrightarrow[f]{} & B \end{array}$$

is a pullback square.

22.10 Consider a pullback square

$$\begin{array}{ccc} A' & \xrightarrow{p} & A \\ {\scriptstyle f'}\downarrow & & \downarrow{\scriptstyle f} \\ B' & \xrightarrow[q]{} & B, \end{array}$$

in which $q : B' \to B$ is surjective. Show that if $f' : A' \to B'$ is an embedding, then so is $f : A \to B$.

22.11 Consider a family of diagrams of the form

$$\begin{array}{ccccc} A_i & \longrightarrow & C & \longrightarrow & X \\ {\scriptstyle f_i}\downarrow & & \downarrow{\scriptstyle g} & & \downarrow{\scriptstyle h} \\ B_i & \longrightarrow & D & \longrightarrow & Y \end{array}$$

indexed by $i : I$, in which the left squares are pullback squares, and assume that the induced map

$$\left(\textstyle\sum_{(i:I)} B_i\right) \to D$$

is surjective. Show that the following are equivalent:

(i) For each $i : I$ the outer rectangle is a pullback square.

(ii) The right square is a pullback square.

Hint: By Theorem 22.6.3 it suffices to prove this equivalence for a single

diagram of the form

$$\begin{array}{ccccc} A & \longrightarrow & C & \longrightarrow & X \\ {\scriptstyle f}\downarrow & & {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ B & \longrightarrow & D & \longrightarrow & Y \end{array}$$

where the map $B \to D$ is assumed to be surjective.

22.12 Consider a pullback square

$$\begin{array}{ccc} E' & \xrightarrow{g} & E \\ {\scriptstyle p'}\downarrow & & \downarrow{\scriptstyle p} \\ B' & \xrightarrow[f]{} & B \end{array}$$

in which p is assumed to be surjective. Show that p' is also surjective, and show that the following are equivalent:

(i) The map f is an equivalence.
(ii) The map g is an equivalence.

22.13 Show that a map $f : A \to B$ is an equivalence if and only if the square

$$\begin{array}{ccc} A^B & \xrightarrow{f\circ -} & B^B \\ {\scriptstyle -\circ f}\downarrow & & \downarrow{\scriptstyle -\circ f} \\ A^A & \xrightarrow[f\circ -]{} & B^A \end{array}$$

is a pullback square.

23 Homotopy pushouts

A common way in topology to construct new spaces is by attaching cells to a given space. A 0-cell is just a point, an 1-cell is an interval, a 2-cell is a disc, a 3-cell is the solid ball, and so forth. Many spaces can be obtained by attaching cells. For example, the circle is obtained by attaching a 1-cell to a 0-cell, so that both end-points of the interval are mapped to the point. More generally, an n-sphere is obtained by attaching an n-disc to the point, so that its entire boundary gets mapped to the point.

In type theory we can also consider a notion of n-cells. Just as in topology, a 0-cell is just a point (i.e., a term). A 1-cell, however, is in type theory an identification, i.e., a term of the identity type. A 1-cell is then an identification of identifications, and so forth. Then we can attach cells to a type by taking a pushout, which is a process dual to taking a pullback.

The idea of pushouts is to glue two types A and B together using a mediating type S and maps $f : S \to A$ and $g : S \to B$. In other words, we start by a diagram of the form

$$A \xleftarrow{f} S \xrightarrow{g} B.$$

We call such a triple $\mathcal{S} \doteq (S, f, g)$ a **span** from A to B. A span from A to B can be thought of as a relation from A to B, relating $f(s)$ to $g(s)$ for any $s : S$. The pushout of the span $\mathcal{S}$ is then a type X that comes equipped with inclusion maps $i : A \to X$ and $j : B \to X$ and a homotopy H witnessing that the square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X \end{array}$$

Note that this homotopy makes sure that there is a path $H(s) : i(f(s)) = j(g(s))$ for every $s : S$. In other words, any $x : A$ and $y : B$ that are related by in $\mathcal{S}$ become identified in the pushout. The last requirement of the pushout is that it satisfies a universal property that is dual to the universal property of pullbacks.

There are several equivalent characterizations of pushouts. Two such characterizations are studied in this section, establishing the duality between pullbacks and pushouts. Other characterizations, including the induction principle of pushouts, and the *dependent universal property* of pushouts, are studied in Section 25.

Unlike pullbacks, however, it is not automatically the case that pushouts always exist. We will therefore postulate as an axiom that pushouts always exist. Moreover, we will assume that universes are closed under pushouts.

23.1 The universal property of pushouts

Definition 23.1.1 Consider a span $\mathcal{S} \doteq (S, f, g)$ from A to B, and let X be a type. A **cocone** with vertex X on $\mathcal{S}$ is a triple (i, j, H) consisting of maps $i : A \to X$ and $j : B \to X$, and a homotopy $H : i \circ f \sim j \circ g$ witnessing that the square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X \end{array}$$

commutes. We write $\mathsf{cocone}_{\mathcal{S}}(X)$ for the type of cocones on $\mathcal{S}$ with vertex X.

Remark 23.1.2 Given two cocones (i, j, H) and (i', j', H') with vertex X, the type of identifications $(i, j, H) = (i', j', H')$ in $\mathsf{cocone}_{\mathcal{S}}(X)$ is equivalent to the type of triples (K, L, M) consisting of

$$K : i \sim i'$$
$$L : j \sim j',$$

and a homotopy M witnessing that the square

$$\begin{array}{ccc} i \circ f & \xrightarrow{K \cdot f} & i' \circ f \\ {\scriptstyle H}\downarrow & & \downarrow{\scriptstyle H'} \\ j \circ g & \xrightarrow[L \cdot g]{} & j' \circ g \end{array}$$

of homotopies commutes.

Definition 23.1.3 Consider a cocone (i, j, H) with vertex X on the span $\mathcal{S} \doteq (S, f, g)$, as indicated in the following commuting square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

For every type Y, we define the map

$$\mathsf{cocone\text{-}map}(i, j, H) : (X \to Y) \to \mathsf{cocone}(Y)$$

by $h \mapsto (h \circ i, h \circ j, h \cdot H)$.

Definition 23.1.4 A commuting square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

with $H : i \circ f \sim j \circ g$ is said to be a **(homotopy) pushout square** if the cocone (i, j, H) with vertex X on the span $\mathcal{S} \doteq (S, f, g)$ satisfies the **universal property of pushouts**, which asserts that the map

$$\mathsf{cocone\text{-}map}(i, j, H) : (X \to Y) \to \mathsf{cocone}(Y)$$

is an equivalence for any type Y. Sometimes pushout squares are also called **cocartesian squares**.

Lemma 23.1.5 *Consider a pushout square*

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

with $H : i \circ f \sim j \circ g$, and consider a commuting square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j'} \\ A & \xrightarrow[i']{} & X'. \end{array}$$

with $H' : i' \circ f \sim j' \circ g$. Then the type of maps $h : X \to X'$ equipped with homotopies

$$K : h \circ i \sim i'$$
$$L : h \circ j \sim j'$$

and a homotopy M witnessing that the square

$$\begin{array}{ccc} h \circ i \circ f & \xrightarrow{K \cdot f} & i' \circ f \\ {\scriptstyle h \cdot H}\downarrow & & \downarrow{\scriptstyle H'} \\ h \circ j \circ g & \xrightarrow[L \cdot g]{} & j' \circ g \end{array}$$

commutes, is contractible.

Proof For any map $h : X \to X$', the type of triples (K, L, M) as in the statement of the lemma is equivalent to the type of identifications

$$\mathsf{cocone\text{-}map}((i, j, H), h) = (i', j', H'),$$

by Remark 23.1.2. Therefore it follows that the type of quadruples (h, K, L, M) is equivalent to the fiber of $\mathsf{cocone\text{-}map}(i, j, H)$ at (i', j', H'). Since we have assumed that the cocone (i, j, H) satisfies the universal property of the pushout of $\mathcal{S}$, the map $\mathsf{cocone\text{-}map}(i, j, H)$ is an equivalence, and therefore it has contractible fibers by Theorem 10.4.6. □

Theorem 23.1.6 *Consider two cocones*

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X \end{array} \qquad \begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j'} \\ A & \xrightarrow[i']{} & X' \end{array}$$

on a span $\mathcal{S} \doteq (S, f, g)$*, and let* $h : X \to X'$ *be a map equipped with homotopies*

$$K : h \circ i \sim i'$$
$$L : h \circ j \sim j'$$

and a homotopy M *witnessing that the square*

$$\begin{array}{ccc} h \circ i \circ f & \xrightarrow{K \cdot f} & i' \circ f \\ {\scriptstyle h \cdot H}\downarrow & & \downarrow{\scriptstyle H'} \\ h \circ j \circ g & \xrightarrow[L \cdot g]{} & j' \circ g \end{array}$$

commutes. Then if any two of the following three statements hold, so does the third:

(i) *The cocone* (i, j, H) *satisfies the universal property of the pushout of* $\mathcal{S}$*.*
(ii) *The cocone* (i', j', H') *satisfies the universal property of the pushotu of* $\mathcal{S}$*.*
(iii) *The map* h *is an equivalence.*

Proof First we observe that we have a commuting triangle

$$\begin{array}{ccc} (X' \to Y) & \xrightarrow{-\circ h} & (X \to Y) \\ & {\scriptstyle \mathsf{cocone\text{-}map}(i',j',H')} \searrow \quad \swarrow {\scriptstyle \mathsf{cocone\text{-}map}(i,j,H)} & \\ & \mathsf{cocone}_{\mathcal{S}}(Y) & \end{array}$$

for any type Y. Therefore it follows from the 3-for-2 property of equivalences that if any two of the maps in this triangle is an equivalence, so is the third. Now the claim follows from the observation in Theorem 13.4.1 that h is an equivalence if and only if the map $- \circ h : (X' \to Y) \to (X \to Y)$ is an equivalence for any type Y. □

In the following corollary we establish the fact that pushouts are *uniquely unique*.

Corollary 23.1.7 *Consider two pushouts*

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X \end{array} \qquad \begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j'} \\ A & \xrightarrow[i']{} & X' \end{array}$$

of a given span $\mathcal{S} \doteq (S, f, g)$*. Then the type of equivalences* $e : X \simeq X'$ *equipped with homotopies*

$$K : h \circ i \sim i'$$

$$L : h \circ j \sim j'$$

and a homotopy M witnessing that the square

$$\begin{array}{ccc} h \circ i \circ f & \xrightarrow{K \cdot f} & i' \circ f \\ {\scriptstyle h \cdot H}\downarrow & & \downarrow{\scriptstyle H'} \\ h \circ j \circ g & \xrightarrow[L \cdot g]{} & j' \circ g \end{array}$$

commutes, is contractible.

Proof This follows from combining Lemma 23.1.5 and Theorem 23.1.6. □

Corollary 23.1.8 *Consider a span*

$$A \xleftarrow{f} S \xrightarrow{g} B$$

in a universe $\mathcal{U}$. Then the type

$$\textstyle\sum_{(X:\mathcal{U})} \sum_{(c:\mathsf{cocone}(X))} \prod_{(Y:\mathcal{U})} \mathsf{is\text{-}equiv}(\mathsf{cocone\text{-}map}_Y(c))$$

of is a proposition.

Proof It is routine to verify that the type of quadruples (e, K, L, M) as in Corollary 23.1.8 is equivalent to the identity type of the type of pushouts of the span $\mathcal{S} \doteq (S, f, g)$. The claim then follows, since Corollary 23.1.8 asserts that this type of quadruples is contractible. □

23.2 Suspensions

A particularly important class of examples of pushouts are suspensions.

Definition 23.2.1 Let X be a type. A **suspension** of X is a type ΣX equipped with a **north pole** $\mathsf{N} : \Sigma X$, a **south pole** $\mathsf{S} : \Sigma X$, and a **meridian**

$$\mathsf{merid} : X \to (\mathsf{N} = \mathsf{S}),$$

such that the commuting square

$$\begin{array}{ccc} X & \xrightarrow{\mathsf{const}_\star} & \mathbf{1} \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle \mathsf{const}_\mathsf{S}} \\ \mathbf{1} & \xrightarrow[\mathsf{const}_\mathsf{N}]{} & \Sigma X \end{array}$$

is a pushout square.

We can use suspensions to present the spheres in type theory. The 2-sphere is a space which, like the surface of the earth, has a north pole and a south pole. Moreover, for each point of the equator there is a meridian that connects the north pole to the south pool. Of course, the equator is a circle, so we see that the 2-sphere is just the suspension of the circle.

Similarly we can see that the $(n+1)$-sphere must be the suspension of the n-sphere. The $(n+1)$-sphere is the unit sphere in the vector space $\mathbb{R}^{n+2}$. This vector space has an orthogonal basis $e_1, \ldots, e_{n+2}$. Then the north and the south pole are given by e_{n+2} and $-e_{n+2}$, respectively, and for each unit vector in $\mathbb{R}^{n+1} \subseteq \mathbb{R}^{n+2}$ we have a meridian connecting the north pole with the south pole. The unit sphere in $\mathbb{R}^{n+1}$ is of course the n-sphere, so we see that the $(n+1)$-sphere must be a suspension of the n-sphere.

These observations suggest that we can define the spheres by recursion on n. Note that the spheres in type theory are defined entirely synthetically, i.e., without reference to the ambient topological space $\mathbb{R}^{n+1}$. Indeed, from a homotopical point of view each space $\mathbb{R}^n$ is contractible, so in type theory it is just presented as the unit type[1].

Definition 23.2.2 We define the **n-sphere $\mathbf{S}^n$** for any $n : \mathbb{N}$ by induction on n, by taking

$$\begin{aligned} \mathbf{S}^0 &:\equiv \mathsf{bool} \\ \mathbf{S}^{n+1} &:\equiv \Sigma \mathbf{S}^n. \end{aligned}$$

Remark 23.2.3 Note that this recursive definition of the spheres only goes through in type theory if we have (or assume) a universe that is closed under suspensions.

In the following lemma we give a slight simplification of the universal property of suspensions, making it just a little easier to work with them.

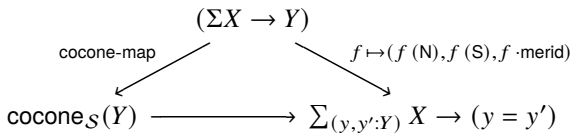
Lemma 23.2.4 *Let X and Y be types, and let ΣX be a suspension of X. Then the map*

$$(\Sigma X \to Y) \to \textstyle\sum_{(y,y':Y)} X \to (y = y')$$

given by $f \mapsto (f(\mathsf{N}), f(\mathsf{S}), f \cdot \mathsf{merid})$ is an equivalence.

[1] It is an entirely different matter to define the *set* $\mathbb{R}$ rather than the homotopy type of $\mathbb{R}$. See Chapter 11 of [5] for definitions of the Dedekind reals and the Cauchy reals.

Proof Note that we have a commuting triangle



where $\mathcal{S}$ is the span $\mathbf{1} \leftarrow X \rightarrow \mathbf{1}$. The bottom map is given by $(i, j, H) \mapsto (i(\star), j(\star), H)$. This map is an equivalence, and the map on the left is an equivalence by the assumption that ΣX is a suspension of X. Therefore the claim follows by the 3-for-2 property of equivalences. □

23.3 The duality of pullbacks and pushouts

Lemma 23.3.1 *For any span $\mathcal{S} \doteq (S, f, g)$ from A to B, and any type X the square*

$$\begin{array}{ccc} \mathsf{cocone}_{\mathcal{S}}(X) & \xrightarrow{\pi_2} & X^B \\ {\scriptstyle \pi_1}\downarrow & & \downarrow{\scriptstyle -\circ g} \\ X^A & \xrightarrow[-\circ f]{} & X^S, \end{array}$$

which commutes by the homotopy $\pi_3' :\equiv \lambda(i, j, H).\, \mathsf{eq\text{-}htpy}(H)$, is a pullback square.

Proof The gap map $\mathsf{cocone}_{\mathcal{S}}(X) \to X^A \times_{X^S} X^B$ is the function

$$\lambda(i, j, H).\, (i, j, \mathsf{eq\text{-}htpy}(H)).$$

This is an equivalence by Theorem 11.1.3, since it is the induced map on total spaces of the family of equivalences eq-htpy. Therefore, the square is a pullback square by Theorem 22.2.6. □

In the following theorem we establish the duality between pullbacks and pushouts.

Theorem 23.3.2 *Consider a commuting square*

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X, \end{array}$$

with $H : i \circ f \sim j \circ g$. The following are equivalent:

(i) *The square is a pushout square.*

(ii) *The square*

$$\begin{array}{ccc} T^X & \xrightarrow{-\circ j} & T^B \\ {\scriptstyle -\circ i}\downarrow & & \downarrow{\scriptstyle -\circ g} \\ T^A & \xrightarrow[-\circ f]{} & T^S \end{array}$$

which commutes by the homotopy

$$\lambda h.\, \mathsf{eq\text{-}htpy}(h \cdot H)$$

is a pullback square, for every type T.

Proof It is straightforward to verify that the triangle

$$\begin{array}{ccc} & T^X & \\ {\scriptstyle \mathsf{cocone\text{-}map}(i,j,H)}\swarrow & & \searrow{\scriptstyle \mathsf{gap}(-\circ i,-\circ j,\mathsf{eq\text{-}htpy}(-\cdot H))} \\ \mathsf{cocone}(T) & \xrightarrow[\mathsf{gap}(i,j,\mathsf{eq\text{-}htpy}(H))]{} & T^A \times_{T^S} T^B \end{array}$$

commutes. Since the bottom map is an equivalence by Lemma 23.3.1, it follows that if either one of the remaining maps is an equivalence, so is the other. The claim now follows by Theorem 22.2.6. □

Example 23.3.3 The square

$$\begin{array}{ccc} X^{\mathbf{S}^1} & \xrightarrow{-\circ \mathsf{const}_{\mathsf{base}}} & X^{\mathbf{1}} \\ {\scriptstyle -\circ \mathsf{const}_{\mathsf{base}}}\downarrow & & \downarrow{\scriptstyle -\circ \mathsf{const}_\star} \\ X^{\mathbf{1}} & \xrightarrow[-\circ \mathsf{const}_\star]{} & X^{\mathsf{bool}} \end{array}$$

is a pullback square for each type X. Therefore it follows by the second characterization of pushouts in Theorem 23.3.2 that the circle is a pushout

$$\begin{array}{ccc} \mathsf{bool} & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow \\ \mathbf{1} & \longrightarrow & \mathbf{S}^1. \end{array}$$

In other words, $\mathbf{S}^1 \simeq \Sigma\mathsf{bool}$.

Theorem 23.3.4 *Consider the following configuration of commuting squares:*

$$\begin{array}{ccccc} A & \xrightarrow{i} & B & \xrightarrow{k} & C \\ {\scriptstyle f}\downarrow & & {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ X & \xrightarrow[j]{} & Y & \xrightarrow[l]{} & Z \end{array}$$

with homotopies $H : j \circ f \sim g \circ i$ and $K : l \circ g \sim h \circ k$, and suppose that the square on the left is a pushout square. Then the square on the right is a pushout square if and only if the outer rectangle is a pushout square.

Proof Let T be a type. Taking the exponent $T^{(-)}$ of the entire diagram of the statement of the theorem, we obtain the following commuting diagram

$$\begin{array}{ccccc} T^Z & \xrightarrow{-\circ l} & T^Y & \xrightarrow{-\circ j} & T^X \\ {\scriptstyle -\circ h}\downarrow & & {\scriptstyle -\circ g}\downarrow & & \downarrow{\scriptstyle -\circ f} \\ T^C & \xrightarrow[-\circ k]{} & T^B & \xrightarrow[-\circ i]{} & T^A. \end{array}$$

By the assumption that Y is the pushout of $B \leftarrow A \rightarrow X$, it follows that the square on the right is a pullback square. It follows by Theorem 22.5.8 that the rectangle on the left is a pullback if and only if the outer rectangle is a pullback. Thus the statement follows by the second characterization in Theorem 23.3.2. □

Lemma 23.3.5 *Consider a map $f : A \to B$. Then the cofiber of the map $\mathsf{inr} : B \to \mathsf{cofib}_f$ is equivalent to the suspension ΣA of A.*

23.4 Fiber sequences and cofiber sequences

Definition 23.4.1 Given a map $f : A \to B$, we define the **cofiber** cofib_f of f as the pushout

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ \mathbf{1} & \xrightarrow[\mathsf{inl}]{} & \mathsf{cofib}_f. \end{array}$$

The cofiber of a map is sometimes also called the **mapping cone**.

Example 23.4.2 The suspension ΣX of X is the cofiber of the map $X \to \mathbf{1}$.

23.5 Further examples of pushouts

Definition 23.5.1 We define the **join** $X * Y$ of X and Y to be the pushout

$$\begin{array}{ccc} X \times Y & \xrightarrow{\mathsf{pr}_2} & Y \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ X & \xrightarrow[\mathsf{inl}]{} & X * Y. \end{array}$$

Definition 23.5.2 Suppose A and B are pointed types, with base points a_0 and b_0, respectively. The **(binary) wedge** $A \vee B$ of A and B is defined as the pushout

$$\begin{array}{ccc} \mathsf{bool} & \longrightarrow & A + B \\ \downarrow & & \downarrow \\ \mathbf{1} & \longrightarrow & A \vee B. \end{array}$$

Definition 23.5.3 Given a type I, and a family of pointed types A over i, with base points $a_0(i)$. We define the **(indexed) wedge** $\bigvee_{(i:I)} A_i$ as the pushout

$$\begin{array}{ccc} I & \xrightarrow{\lambda i.\,(i, a_0(i))} & \sum_{(i:I)} A_i \\ \downarrow & & \downarrow \\ \mathbf{1} & \longrightarrow & \bigvee_{(i:I)} A_i. \end{array}$$

Definition 23.5.4 Let X and Y be types with base points x_0 and y_0, respectively. We define the **wedge** $X \vee Y$ of X and Y to be the pushout

$$\begin{array}{ccc} \mathsf{bool} & \xrightarrow{\mathsf{ind}_{\mathsf{bool}}(\mathsf{inl}(x_0), \mathsf{inr}(y_0))} & X + Y \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ \mathbf{1} & \xrightarrow[\mathsf{inl}]{} & X \vee Y \end{array}$$

Definition 23.5.5 Let X and Y be types with base points x_0 and y_0, respectively. We define a map

$$\mathsf{wedge\text{-}incl} : X \vee Y \to X \times Y.$$

as the unique map obtained from the commutative square

$$\begin{array}{ccc} \mathsf{bool} & \xrightarrow{\mathsf{ind}_{\mathsf{bool}}(\mathsf{inl}(x_0), \mathsf{inr}(y_0))} & X + Y \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle \mathsf{ind}_{X+Y}(\lambda x.\,(x, y_0), \lambda y.\,(x_0, y))} \\ \mathbf{1} & \xrightarrow[\lambda t.\,(x_0, y_0)]{} & X \times Y. \end{array}$$

Definition 23.5.6 We define the **smash product** $X \wedge Y$ of X and Y to be the pushout

$$\begin{array}{ccc} X \vee Y & \xrightarrow{\mathsf{wedge\text{-}incl}} & X \times Y \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ \mathbf{1} & \xrightarrow[\mathsf{inl}]{} & X \wedge Y. \end{array}$$

Exercises

23.1 Use Theorems 13.4.1 and 23.3.2 and Corollary 22.5.5 to show that for any commuting square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow{\scriptstyle \simeq} & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & C \end{array}$$

where f is an equivalence, the square is a pushout square if and only if $j : B \to C$ is an equivalence. Use this observation to conclude the following:

(i) If X is contractible, then ΣX is contractible.
(ii) The cofiber of any equivalence is contractible.
(iii) The cofiber of a point in B (i.e., of a map of the type $\mathbf{1} \to B$) is equivalent to B.
(iv) There is an equivalence $X \simeq \emptyset * X$.
(v) If X is contractible, then $X * Y$ is contractible.
(vi) If A is contractible, then there is an equivalence $A \vee B \simeq B$ for any pointed type B.

23.2 Let P and Q be propositions.

(a) Show that $P * Q$ satisfies the *universal property of disjunction*, i.e., that for any proposition R, the map

$$(P * Q \to R) \to (P \to R) \times (Q \to R)$$

given by $f \mapsto (f \circ \mathsf{inl}, f \circ \mathsf{inr})$, is an equivalence.
(b) Use the proposition $R :\equiv \mathsf{is\text{-}contr}(P * Q)$ to show that $P * Q$ is again a proposition.

23.3 Let Q be a proposition, and let A be a type. Show that the following are equivalent:

(i) The map $(Q \to A) \to (\emptyset \to A)$ is an equivalence.
(ii) The type A^Q is contractible.
(iii) There is a term of type $Q \to \mathsf{is\text{-}contr}(A)$.
(iv) The map $\mathsf{inr} : A \to Q * A$ is an equivalence.

23.4 Let P be a proposition. Show that ΣP is a set, with an equivalence

$$\Big(\mathsf{inl}(\star) = \mathsf{inr}(\star)\Big) \simeq P.$$

23.5 Show that $A \sqcup^{\mathcal{S}} B \simeq B \sqcup^{\mathcal{S}^{\mathsf{op}}} A$, where $\mathcal{S}^{\mathsf{op}} :\equiv (S, g, f)$ is the **opposite span** of $\mathcal{S}$.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

23.6 Use Exercise 22.8 to show that if

$$\begin{array}{ccc} S & \longrightarrow & Y \\ \downarrow & & \downarrow \\ X & \longrightarrow & Z \end{array}$$

is a pushout square, then so is

$$\begin{array}{ccc} A \times S & \longrightarrow & A \times Y \\ \downarrow & & \downarrow \\ A \times X & \longrightarrow & A \times Z \end{array}$$

for any type A.

23.7 Use Exercise 22.7 to show that if

$$\begin{array}{ccc} S_1 & \longrightarrow & Y_1 \\ \downarrow & & \downarrow \\ X_1 & \longrightarrow & Z_1 \end{array} \qquad \begin{array}{ccc} S_2 & \longrightarrow & Y_2 \\ \downarrow & & \downarrow \\ X_2 & \longrightarrow & Z_2 \end{array}$$

are pushout squares, then so is

$$\begin{array}{ccc} S_1 + S_2 & \longrightarrow & Y_1 + Y_2 \\ \downarrow & & \downarrow \\ X_1 + X_2 & \longrightarrow & Z_1 + Z_2. \end{array}$$

23.8 (a) Consider a span (S, f, g) from A to B. Use Exercise 22.6 to show that the square

$$\begin{array}{ccc} S + S & \xrightarrow{[\mathsf{id},\mathsf{id}]} & S \\ {\scriptstyle f+g}\downarrow & & \downarrow{\scriptstyle \mathsf{inr}\circ g} \\ A + B & \xrightarrow[[\mathsf{inl},\mathsf{inr}]]{} & A \sqcup^S B \end{array}$$

is again a pushout square.

(b) Show that $\Sigma X \simeq \mathsf{bool} * X$.

23.9 Consider a commuting triangle

$$\begin{array}{ccc} A & \xrightarrow{h} & B \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle g} & \\ & X & \end{array}$$

with $H : f \sim g \circ h$.

(a) Construct a map $\mathsf{cofib}_{(h,H)} : \mathsf{cofib}_g \to \mathsf{cofib}_f$.

There are 306 exercises.

Ljubljana, November 10, 2021

(b) Use **??** to show that $\mathsf{cofib}_{\mathsf{cofib}(h,H)} \simeq \mathsf{cofib}_h$.

23.10 Use Exercise 20.4 to show that for $n \geq 0$, X is an n-type if and only if the map

$$\lambda x.\, \mathsf{const}_x : X \to (\mathbf{S}^{n+1} \to X)$$

is an equivalence.

23.11 (a) Construct for every $f : X \to Y$ a function

$$\Sigma f : \Sigma X \to \Sigma Y.$$

(b) Show that if $f \sim g$, then $\Sigma f \sim \Sigma g$.

(c) Show that $\Sigma \mathsf{id}_X \sim \mathsf{id}_{\Sigma X}$

(d) Show that

$$\Sigma(g \circ f) \sim (\Sigma g) \circ (\Sigma f).$$

for any $f : X \to Y$ and $g : Y \to Z$.

23.12 (a) Let I be a type, and let A be a family over I. Construct an equivalence

$$\Big(\bigvee_{(i:I)} \Sigma A_i\Big) \simeq \Sigma\Big(\bigvee_{(i:I)} A_i\Big).$$

(b) Show that for any type X there is an equivalence

$$\Big(\bigvee_{(x:X)} \mathsf{bool}\Big) \simeq X + \mathbf{1}.$$

(c) Construct an equivalence

$$\Sigma(\mathsf{Fin}(n+1)) \simeq \bigvee_{(i:\mathsf{Fin}(n))} \mathbf{S}^1.$$

23.13 Show that $\mathsf{Fin}(n+1) * \mathsf{Fin}(m+1) \simeq \bigvee_{(i:\mathsf{Fin}(n \cdot m))} \mathbf{S}^1$, for any $n, m : \mathbb{N}$.

23.14 For any pointed set X, show that the squares

$$\begin{array}{ccc} \mathbf{S}^1 & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow \\ \bigvee_{(x:X)} \mathbf{S}^1 & \longrightarrow & \Sigma X \end{array} \qquad \text{and} \qquad \begin{array}{ccc} X \times \mathbf{S}^1 & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow \\ \bigvee_{(x:X)} \mathbf{S}^1 & \longrightarrow & \Sigma X \end{array}$$

are pushout squares.

23.15 Show that the square

$$\begin{array}{ccc} \mathbf{S}^1 & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow \\ \mathbf{S}^1 \times \mathbf{S}^1 & \longrightarrow & \mathbf{S}^2 \vee \mathbf{S}^1 \end{array}$$

is a pushout square.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

23.16 For any type X, show that the mapping cone of the fold map $X + X \to X$ is the suspension of $X + \mathbf{1}$, i.e. show that the following square

$$\begin{array}{ccc} X + X & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow \\ X & \longrightarrow & \Sigma X + \mathbf{1} \end{array}$$

is a pushout square.

23.17 Consider a map $f : A \to B$. Show that f is a k-truncated map if and only if the square

$$\begin{array}{ccc} A & \xrightarrow{\delta} & A^{\mathbf{S}^{k+1}} \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f^{\mathbf{S}^{k+1}}} \\ B & \xrightarrow[\delta]{} & B^{\mathbf{S}^{k+1}} \end{array}$$

is a pullback square.

23.18 Show that a type A is a proposition if and only if the map $\mathsf{inl} : A \to A * A$ is an equivalence.

23.19 Let A be a type, and let P be a proposition.

(a) Show that $\mathsf{inl} : P \to P * A$ is an embedding.
(b) Show that $\mathsf{inl} : P \to P * A$ is an equivalence if and only if $\|A\| \to P$ holds.

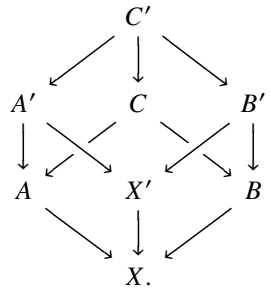
23.20 A **cogroup** is a pointed type X equipped with a pointed map

$$\nu : X \to X \vee X$$

Show that any suspension can be given the structure of a cogroup.

24 Cubical diagrams

In order to proceed with the development of pullbacks and pushouts, it is useful to study commuting diagrams of the form



In these diagrams there are six homotopies witnessing that the faces of the cube commute, as well as a homotopy of homotopies witnessing that the cube as a whole commutes.

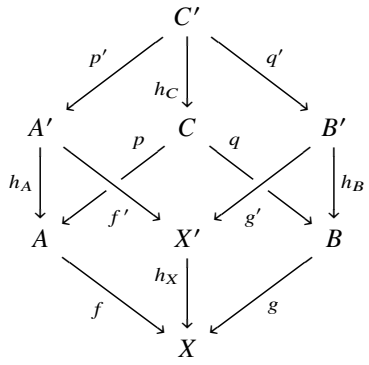
Once the basic definitions of cubes are established, we focus on pullbacks and pushouts that appear in different configurations in these cubical diagrams. For example, if all the vertical maps in a commuting cube are equivalences, then the top square is a pullback square if and only if the bottom square is a pullback square. In Section 25 we will use cubical diagrams in our formulation of the universality and descent theorems for pushouts.

In the first main theorem of this section we show that given a commuting cube in which the bottom square is a pullback square, the top square is a pullback square if and only if the induced square of fibers of the vertical maps is a pullback square. This theorem should be compared to Theorem 22.5.3, where we showed that a square is a pullback square if and only if it induces equivalences on the fibers of the vertical maps.

In our second main theorem we use the previous result to derive the 3-by-3 properties for pullbacks and pushouts.

24.1 Commuting cubes

Definition 24.1.1 A **commuting cube**



consists of types and maps as indicated in the diagram, equipped with

(i) homotopies

$$\begin{aligned}
\mathsf{top} &: f' \circ p' \sim g' \circ q' \\
\mathsf{back\text{-}left} &: p \circ h_C \sim h_A \circ p' \\
\mathsf{back\text{-}right} &: q \circ h_C \sim h_B \circ q' \\
\mathsf{front\text{-}left} &: f \circ h_A \sim h_X \circ f' \\
\mathsf{front\text{-}right} &: g \circ h_B \sim h_X \circ g' \\
\mathsf{bottom} &: f \circ p \sim g \circ q
\end{aligned}$$

witnessing that the 6 faces of the cube commute,

(ii) and a homotopy

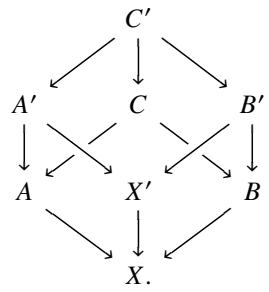
$$\begin{aligned}
\mathsf{coh\text{-}cube} : &((f \cdot \mathsf{back\text{-}left}) \centerdot (\mathsf{front\text{-}left} \cdot p')) \centerdot (h_X \cdot \mathsf{top}) \\
&\sim (\mathsf{bottom} \cdot h_C) \centerdot ((g \cdot \mathsf{back\text{-}right}) \centerdot (\mathsf{front\text{-}right} \cdot q'))
\end{aligned}$$

filling the cube.

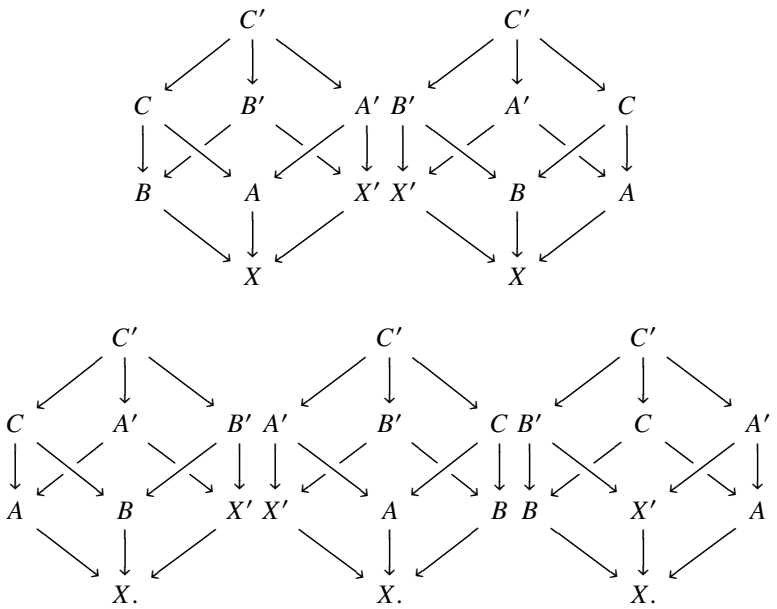
In the following lemma we show that if a cube commutes, then so do its rotations and mirror symmetries (that preserve the directions of the arrows).[2] This fact is obviously true, but there is some 'path algebra' involved that we wish to demonstrate at least once.

[2] The group acting on commuting cubes of maps is the *dihedral group* D_3 which has order 6.

Lemma 24.1.2 *Consider a commuting cube*



Then the cubes



also commute.

Proof We only show that the first cube commutes, which is obtained by a counter-clockwise rotation of the original cube around the axis through C' and X. The other cases are similar, and they are formalized in the accompagnying Agda library.

First we list the homotopies witnessing that the faces of the cube commute:

$$\mathsf{top}' := \mathsf{back\text{-}left}$$
$$\mathsf{back\text{-}left}' := \mathsf{back\text{-}right}^{-1}$$

$$\begin{aligned}\mathsf{back\text{-}right}' &:= \mathsf{top}^{-1}\\ \mathsf{front\text{-}left}' &:= \mathsf{bottom}^{-1}\\ \mathsf{front\text{-}right}' &:= \mathsf{front\text{-}left}^{-1}\\ \mathsf{bottom}' &:= \mathsf{front\text{-}right}.\end{aligned}$$

Thus, to show that the cube commutes, we have to show that there is a homotopy of type

$$\begin{aligned}&\Big((g \cdot \mathsf{back\text{-}right}^{-1}) \cdot (\mathsf{bottom}^{-1} \cdot h_C)\Big) \cdot (f \cdot \mathsf{back\text{-}left})\\ &\qquad \sim (\mathsf{front\text{-}right} \cdot q') \cdot \Big((h_X \cdot \mathsf{top}^{-1}) \cdot (\mathsf{front\text{-}left}^{-1} \cdot p')\Big).\end{aligned}$$

Recall that $h \cdot H^{-1} \sim (h \cdot H)^{-1}$ and $H^{-1} \cdot h \sim (H \cdot h)^{-1}$, so it suffices to construct a homotopy

$$\begin{aligned}&\Big((g \cdot \mathsf{back\text{-}right})^{-1} \cdot (\mathsf{bottom} \cdot h_C)^{-1}\Big) \cdot (f \cdot \mathsf{back\text{-}left})\\ &\qquad \sim (\mathsf{front\text{-}right} \cdot q') \cdot \Big((h_X \cdot \mathsf{top})^{-1} \cdot (\mathsf{front\text{-}left} \cdot p')^{-1}\Big).\end{aligned}$$

Now we note that pointwise, our goal is of the form

$$(\varepsilon^{-1} \cdot \delta^{-1}) \cdot \alpha = \zeta \cdot (\gamma^{-1} \cdot \beta^{-1}),$$

whereas the assumption that the original cube commutes yields an identification of the form

$$(\alpha \cdot \beta) \cdot \gamma = \delta \cdot (\varepsilon \cdot \zeta)$$

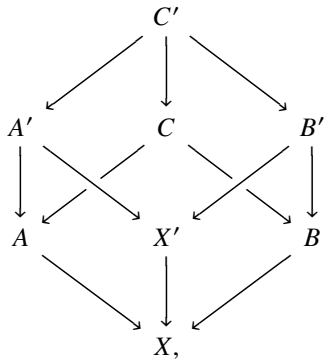
Indeed, in the case that α, β, γ, δ, ε, and ζ are general identifications, we can conclude our goal using path induction on all of them. □

Lemma 24.1.3 *Given a commuting cube as in Definition 24.1.1 we obtain a commuting square*

$$\begin{array}{ccc}\mathsf{fib}_{f_{1\check{1}1}}(x) & \longrightarrow & \mathsf{fib}_{f_{0\check{1}1}}(f_{\check{1}01}(x))\\ \downarrow & & \downarrow\\ \mathsf{fib}_{f_{1\check{1}0}}(f_{10\check{1}}(x)) & \longrightarrow & \mathsf{fib}_{f_{0\check{1}0}}(f_{00\check{1}}(x))\end{array}$$

for any $x : A_{101}$.

Lemma 24.1.4 *Consider a commuting cube*



If the bottom and front right squares are pullback squares, then the back left square is a pullback if and only if the top square is.

Remark 24.1.5 By rotating the cube we also obtain:

(i) If the bottom and front left squares are pullback squares, then the back right square is a pullback if and only if the top square is.
(ii) If the front left and front right squares are pullback, then the back left square is a pullback if and only if the back right square is.

By combining these statements it also follows that if the front left, front right, and bottom squares are pullback squares, then if any of the remaining three squares are pullback squares, all of them are. Cubes that consist entirely of pullback squares are sometimes called **strongly cartesian**.

24.2 Families of pullbacks

Lemma 24.2.1 *Consider a pullback square*

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X \end{array}$$

with $H : f \circ p \sim g \circ h$. Furthermore, consider type families P_X, P_A, P_B, and P_C over X, A, B, and C respectively, equipped with families of maps

$$f' : \prod_{(a:A)} P_A(a) \to P_X(f(a))$$
$$g' : \prod_{(b:B)} P_B(b) \to P_X(g(b))$$
$$p' : \prod_{(c:C)} P_C(c) \to P_A(p(c))$$

$$q' : \prod_{(c:C)} P_C(c) \to P_B(q(c)),$$

and for each $c : C$ *a homotopy* H'_c *witnessing that the square*

$$\begin{array}{ccc} P_C(c) & \xrightarrow{\quad q'_c \quad} & P_B(q(c)) \\ {\scriptstyle p'_c}\downarrow & & \downarrow{\scriptstyle g'_{q(c)}} \\ P_A(p(c)) & \xrightarrow[f'_{p(c)}]{} P_X(f(p(c))) \xrightarrow[\mathsf{tr}_{P_X}(H(c))]{} & P_X(g(q(c))) \end{array} \tag{24.2.1}$$

commutes. Then the following are equivalent:

(i) *For each* $c : C$ *the square in Eq. (24.2.1) is a pullback square.*

(ii) *The square*

$$\begin{array}{ccc} \sum_{(c:C)} P_C(c) & \xrightarrow{\mathsf{tot}_q(q')} & \sum_{(b:B)} P_B(b) \\ {\scriptstyle \mathsf{tot}_p(p')}\downarrow & & \downarrow{\scriptstyle \mathsf{tot}_g(g')} \\ \sum_{(a:A)} P_A(a) & \xrightarrow[\mathsf{tot}_f(f')]{} & \sum_{(x:X)} P_X(x) \end{array} \tag{24.2.2}$$

is a pullback square.

Corollary 24.2.2 *Consider a pullback square*

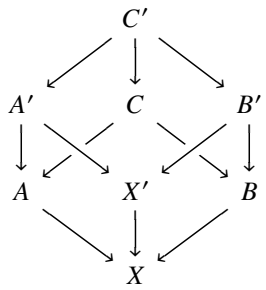
$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X, \end{array}$$

with $H : f \circ p \sim g \circ q$*, and let* $c_1, c_2 : C$*. Then the square*

$$\begin{array}{ccc} (c_1 = c_2) & \xrightarrow{\quad \mathsf{ap}_q \quad} & (q(c_1) = q(c_2)) \\ {\scriptstyle \mathsf{ap}_p}\downarrow & & \downarrow{\scriptstyle \lambda\beta.\, H(c_1) \cdot \mathsf{ap}_g(\beta)} \\ (p(c_1) = p(c_2)) & \xrightarrow[\lambda\alpha.\, \mathsf{ap}_f(\alpha) \cdot H(c_2)]{} & f(p(c_1)) = g(q(c_2)), \end{array}$$

commutes and is a pullback square.

Theorem 24.2.3 *Consider a commuting cube*



in which the bottom square is a pullback square. Then the following are equivalent:

(i) *The top square is a pullback square.*
(ii) *The square*

$$\begin{array}{ccc} \mathsf{fib}_\gamma(c) & \longrightarrow & \mathsf{fib}_\beta(q(c)) \\ \downarrow & & \downarrow \\ \mathsf{fib}_\alpha(p(c)) & \longrightarrow & \mathsf{fib}_\varphi(f(p(c))) \end{array}$$

is a pullback square for each $c : C$.

24.3 The 3-by-3-properties for pullbacks and pushouts

Theorem 24.3.1 *Consider a commuting diagram of the form*

$$\begin{array}{ccccc} AA & \xrightarrow{Af} & AX & \xleftarrow{Ag} & AB \\ {\scriptstyle fA}\downarrow & \Rightarrow & {\scriptstyle fX}\downarrow & \Leftarrow & \downarrow{\scriptstyle gB} \\ XA & \xrightarrow{Xf} & XX & \xleftarrow{Xg} & XB \\ {\scriptstyle gA}\uparrow & & {\scriptstyle gX}\uparrow & & \uparrow{\scriptstyle gB} \\ BA & \xrightarrow[Bf]{} & BX & \xleftarrow[Bg]{} & BB \end{array}$$

with homotopies

$$\begin{aligned} ff &: Xf \circ fA \sim Af \circ fX \\ fg &: Xg \circ gB \sim Ag \circ fX \\ gf &: \end{aligned}$$

filling the (small) squares. Furthermore, consider pullback squares

$$\begin{array}{ccc} AC & \longrightarrow & AB \\ \downarrow & & \downarrow \\ AA & \longrightarrow & AX \end{array} \qquad \begin{array}{ccc} XC & \longrightarrow & XB \\ \downarrow & & \downarrow \\ XA & \longrightarrow & XX \end{array} \qquad \begin{array}{ccc} BC & \longrightarrow & BB \\ \downarrow & & \downarrow \\ BA & \longrightarrow & BX \end{array}$$

$$\begin{array}{ccc} CA & \longrightarrow & BA \\ \downarrow & & \downarrow \\ AA & \longrightarrow & XA \end{array} \qquad \begin{array}{ccc} CX & \longrightarrow & BX \\ \downarrow & & \downarrow \\ AX & \longrightarrow & XX \end{array} \qquad \begin{array}{ccc} CB & \longrightarrow & BB \\ \downarrow & & \downarrow \\ AB & \longrightarrow & XB. \end{array}$$

Finally, consider a commuting square

$$\begin{array}{ccc} D_3 & \longrightarrow & D_2 \\ \downarrow & & \downarrow \\ D_0 & \longrightarrow & D_1. \end{array}$$

Then the following are equivalent:

(i) *This square is a pullback square.*
(ii) *The induced square*

$$\begin{array}{ccc} D_3 & \longrightarrow & C_3 \\ \downarrow & & \downarrow \\ A_3 & \longrightarrow & B_3 \end{array}$$

is a pullback square.

Proof First we construct an equivalence

$$(A_0 \times_{B_0} C_0) \times_{(A_1 \times_{B_1} C_1)} (A_2 \times_{B_2} C_2) \simeq (A_0 \times_{A_1} A_2) \times_{(B_0 \times_{B_1} B_2)} (C_0 \times_{C_1} C_2).$$

Now it follows that we have an equivalence

$$\mathsf{cone}(f_0, g_0)$$

□

Exercises

24.1 Some exercises.

25 Universality and descent for pushouts

We begin this section with the idea that pushouts can be presented as higher inductive types. The general idea behind higher inductive types is that we can introduce new inductive types not only with constructors at the level of points, but also with constructors at the level of identifications. Pushouts form a basic class of examples that can be obtained as higher inductive types, because they come equipped with the structure of a cocone. The cocone (i, j, H) in the commuting square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & C \end{array}$$

equips the type C with two *point constructors*

$$i : A \to C$$
$$j : B \to C$$

and a *path constructor*

$$H : \textstyle\prod_{(s:S)} i(f(s)) = j(g(s))$$

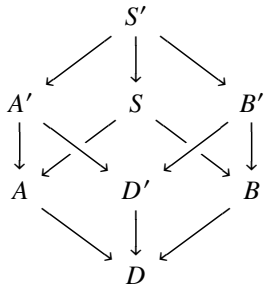
that provides an identification $H(s) : i(f(s)) = j(g(s))$ for every $s : S$. The induction principle then specifies how to construct sections of families over C. Naturally, it takes not only the point constructors i and j, but also the path constructor H into account.

The induction principle is one of several equivalent characterizations of pushouts. We will prove a theorem providing five equivalent characterizations of homotopy pushouts. Two of those we have already seen in Theorem 23.3.2: the universal property and the pullback property. The other three are

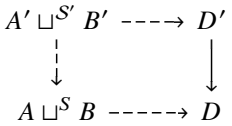
(i) the *dependent pullback property*,
(ii) the *dependent universal property*,
(iii) the *induction principle*.

An implication that is particularly useful among our five characterizations of pushouts, is the fact that the pullback property implies the dependent pullback property. We use the dependent pullback property to derive the *universality of pushouts* (not to be confused with the universal property of pushouts), showing

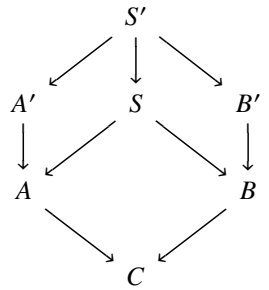
that for any commuting cube



in which the back left and right squares are pullback squares, if the front left and right squares are also pullback squares, then so is the induced square

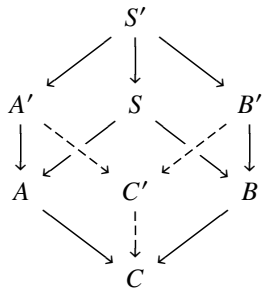


We then observe that the univalence axiom can be used together with the universal property of pushouts to obtain such families over pushouts in the first place. We prove the descent theorem, which asserts that for any diagram of the form



in which the bottom square is a pushout square and the back left and right squares are pullback squares, there is a unique way of extending this to a commuting

cube



in which also the front left and right squares are pullback squares. Thus the converse of the universality theorem for pushouts also follows. The descent property used to show that pullbacks distribute over pushouts, and to compute the fibers of maps out of pushouts (the source of many exercises).

We note that the computation rules in our treatment for the induction principle of homotopy pushouts are weak. In other words, they are identifications. In this course we have no need for judgmental computation rules. Our focus is instead on universal properties. We refer the reader who is interested in the more 'traditional' higher inductive types with judgmental computation rules to [5].

25.1 Five equivalent characterizations of homotopy pushouts

Consider a commuting square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & H \end{array} \tag{25.1.1}$$

with $H : i \circ f \sim j \circ g$, where we will sometimes write $\mathcal{S}$ for the span $A \leftarrow S \rightarrow B$. Our first goal is to formulate the induction principle for pushouts, which specifies how to construct a section of an arbitrary type family P over X. Like the induction principle for the circle, the induction principle of pushouts has to take both the point constructors and the path constructors of X into account. In our case, the point constructors are the maps

$$\begin{aligned} i &: A \to X \\ j &: B \to X, \end{aligned}$$

and the path constructor is the homotopy

$$H : \textstyle\prod_{(s:S)} i(f(s)) = j(g(s)).$$

Therefore, we obtain for any section $h : \prod_{(x:X)} P(x)$ a triple (h_A, h_B, h_S) consisting of

$$\begin{aligned} h_A &: \textstyle\prod_{(a:A)} P(i(a)) \\ h_B &: \textstyle\prod_{(b:B)} P(j(b)) \\ h_S &: \textstyle\prod_{(s:S)} \mathsf{tr}_P(H(s), h(i(f(s)))) = h(j(g(s))). \end{aligned}$$

The dependent functions h_A and h_B are simply given by

$$\begin{aligned} h_A &:\equiv h \circ i \\ h_B &:\equiv h \circ j. \end{aligned}$$

The homotopy h_S is defined by $h_S(s) :\equiv \mathsf{apd}_h(H(s))$, using the dependent action on paths of h. We call such triples (h_A, h_B, h_S) **dependent cocones** on P over the cocone (i, j, H), and will write $\mathsf{dep\text{-}cocone}_{(i,j,H)}(P)$ for this type of dependent cocones. Thus, we have a function

$$\mathsf{ev\text{-}pushout}(P) : \left(\textstyle\prod_{(x:X)} P(x)\right) \to \mathsf{dep\text{-}cocone}_{(i,j,H)}(P).$$

We are now in position to define the induction principle and the dependent universal property of pushouts.

Definition 25.1.1 We say that X satisfies the **induction principle of the pushout of** $\mathcal{S}$ if the function

$$\mathsf{ev\text{-}pushout}(P) : \left(\textstyle\prod_{(x:X)} P(x)\right) \to \mathsf{dep\text{-}cocone}_{(i,j,H)}(P).$$

has a section for every type family P over X.

Definition 25.1.2 We say that X satisfies the **dependent universal property of the pushout of** $\mathcal{S}$ if the function

$$\mathsf{ev\text{-}pushout}(P) : \left(\textstyle\prod_{(x:X)} P(x)\right) \to \mathsf{dep\text{-}cocone}_{(i,j,H)}(P).$$

is an equivalence for every type family P over X.

Remark 25.1.3 For (h_A, h_B, h_S) and (h'_A, h'_B, h'_S) in $\mathsf{dep\text{-}cocone}_{(i,j,H)}(P)$, the type of identifications $(h_A, h_B, h_S) = (h'_A, h'_B, h'_S)$ is equivalent to the type of triples (K_A, K_B, K_S) consisting of

$$\begin{aligned} K_A &: \textstyle\prod_{(a:A)} h_A(a) = h'_A(a) \\ K_B &: \textstyle\prod_{(b:B)} h_B(b) = h'_B(b), \end{aligned}$$

and a homotopy K_S witnessing that the square

$$\begin{array}{ccc} \mathsf{tr}_P(H(s), h_A(f(s))) & \xrightarrow{\mathsf{ap}_{\mathsf{tr}_P(H(s))}(K_A(f(s)))} & \mathsf{tr}_P(H(s), h'_A(f(s))) \\ {\scriptstyle h_S(s)}\Big\| & & \Big\|{\scriptstyle h'_S(s)} \\ h_B(g(s)) & \xrightarrow[K_B(g(s))]{} & h_B(g(s)) \end{array}$$

commutes for every $s : S$.

Therefore we see that the induction principle of the pushout of $\mathcal{S}$ provides us, for every dependent cocone (h_A, h_B, h_S) of P over (i, j, H), with a dependent function $h : \prod_{(x:A)} P(x)$ equipped with homotopies

$$K_A : \prod_{(a:A)} h(i(a)) = h_A(a)$$
$$K_B : \prod_{(b:B)} h(j(b)) = h_B(b),$$

and a homotopy K_S witnessing that the square

$$\begin{array}{ccc} \mathsf{tr}_P(H(s), h(i(f(s)))) & \xrightarrow{\mathsf{ap}_{\mathsf{tr}_P(H(s))}(K_A(f(s)))} & \mathsf{tr}_P(H(s), h_A(f(s))) \\ {\scriptstyle \mathsf{apd}_h(H(s))}\Big\| & & \Big\|{\scriptstyle h_S(s)} \\ h(j(g(s))) & \xrightarrow[K_B(g(s))]{} & h_B(g(s)) \end{array}$$

commutes for every $s : S$. These homotopies are the **computation rules** for pushouts. The dependent universal property is equivalent to the assertion that for every dependent cocone (h_A, h_B, h_S), the type of quadruples (h, K_A, K_B, K_S) is contractible.

Theorem 25.1.4 *Consider a commuting square*

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & C \end{array} \tag{25.1.2}$$

with $H : (i \circ f) \sim (j \circ g)$. Then the following are equivalent:

(i) *The square in Eq. (25.1.2) is a pushout square.*

(ii) *The square in Eq. (25.1.2) satisfies the pullback property of pushouts.*

(iii) *The square satisfies the **dependent pullback property** of pushouts: For*

every family P over C, the square

$$\begin{array}{ccccc}
\prod_{(z:C)} P(z) & & \xrightarrow{\quad h\mapsto h\circ j \quad} & & \prod_{(y:B)} P(j(y)) \\
{\scriptstyle h\mapsto h\circ i}\downarrow & & & & \downarrow{\scriptstyle h\mapsto h\circ g} \\
\prod_{(x:A)} P(i(x)) & \xrightarrow[h\mapsto h\circ f]{} & \prod_{(s:S)} P(i(f(s))) & \xrightarrow[\lambda h.\,\lambda s.\,\mathrm{tr}_P(H(s),h(s))]{} & \prod_{(s:S)} P(j(g(s))),
\end{array} \tag{25.1.3}$$

which commutes by the homotopy

$$\lambda h.\,\mathsf{eq\text{-}htpy}(\lambda s.\,\mathsf{apd}_h(H(s))),$$

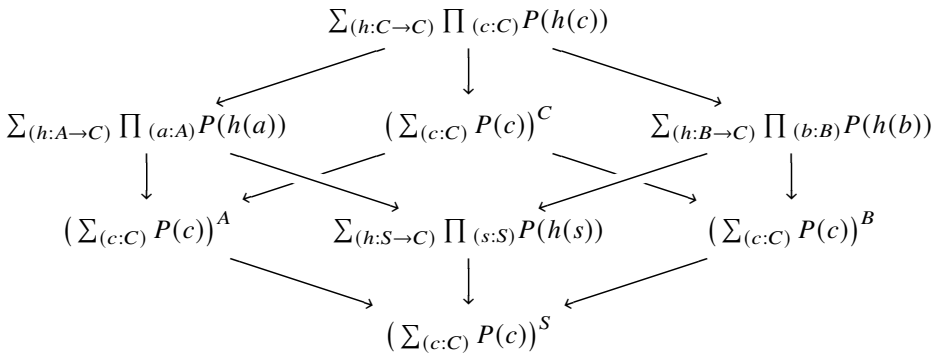
is a pullback square.

(iv) *The type C satisfies the **dependent universal property** of pushouts.*

(v) *The type C satisfies the **induction principle** of pushouts.*

Proof We have already seen in Theorem 23.3.2 that (i) and (ii) are equivalent.

To see that (ii) implies (iii), note that we have a commuting cube



in which the vertical maps are equivalences. Moreover, the bottom square is a pullback square by the pullback property of pushouts, so we conclude that the top square is a pullback square. Since this is a square of total spaces over a pullback square, we invoke Lemma 24.2.1 to conclude that for each $h : C \to C$, the square

$$\begin{array}{ccccc}
\prod_{(c:C)} P(h(c)) & & \longrightarrow & & \prod_{(b:B)} P(h(j(b))) \\
\downarrow & & & & \downarrow \\
\prod_{(a:A)} P(h(i(a))) & \longrightarrow & \prod_{(s:S)} P(h(i(f(s)))) & \xrightarrow[\mathrm{tr}_{((k:S\to C)\mapsto \prod_{(s:S)} P(k(s)))}(\mathsf{eq\text{-}htpy}(h\cdot H))]{} & \prod_{(s:S)} P(h(j(g(s))))
\end{array}$$

is a pullback square. Note that the transport with respect to the family $k \mapsto$

$\prod_{(s:S)}(Pk(s))$ along the identification $\mathsf{eq\text{-}htpy}(h \cdot H)$ is homotopic to the map

$$\lambda h.\, \lambda s.\, \mathsf{tr}_{P \circ h}(H(s), h(s)) : \prod_{(s:S)} P(h(i(f(s)))) \to \prod_{(s:S)} P(h(j(g(s)))).$$

Therefore we conclude that the square

$$\begin{array}{ccc} \prod_{(c:C)} P(h(c)) & \longrightarrow & \prod_{(b:B)} P(h(j(b))) \\ \downarrow & & \downarrow \\ \prod_{(a:A)} P(h(i(a))) & \longrightarrow & \prod_{(s:S)} P(h(i(f(s)))) \xrightarrow[\lambda h.\, \lambda s.\, \mathsf{tr}_{P \circ h}(H(s), h(s))]{} \prod_{(s:S)} P(h(j(g(s)))) \end{array}$$

is a pullback square for each $h : C \to C$. Using the case $h \doteq \mathsf{id} : C \to C$ we conclude that the cocone (i, j, H) satisfies the dependent pullback property.

To see that (iii) implies (ii) we recall that transport with respect to a trivial family is homotopic to the identity function. Thus we obtain the pullback property from the dependent pullback property using the trivial family $\lambda c.\, T$ over C.

To see that (iii) implies (iv) we note that $\mathsf{ev\text{-}pushout}(P)$ is an equivalence if and only if the gap map of the square in Eq. (25.1.3) is an equivalence.

It is clear that (iv) implies (v), so it remains to show that (v) implies (iv). If X satisfies the induction principle of pushouts, then the map

$$\mathsf{ev\text{-}pushout} : \left(\prod_{(x:X)} P(x)\right) \to \mathsf{dep\text{-}cocone}_{(i,j,H)}(P)$$

has a section, i.e., it comes equipped with

$$\mathsf{ind\text{-}pushout} : \mathsf{dep\text{-}cocone}_{(i,j,H)}(P) \to \left(\prod_{(x:X)} P(x)\right)$$
$$\mathsf{comp\text{-}pushout} : \mathsf{ev\text{-}pushout} \circ \mathsf{ind\text{-}pushout} \sim \mathsf{id}.$$

To see that ev-pushout is an equivalence it therefore suffices to construct a homotopy

$$\mathsf{ind\text{-}pushout}(\mathsf{ev\text{-}pushout}(h)) \sim h$$

for any $h : \prod_{(x:X)} P(x)$. From the fact that ind-pushout is a section of ev-pushout we obtain an identification

$$\mathsf{ev\text{-}pushout}(\mathsf{ind\text{-}pushout}(\mathsf{ev\text{-}pushout}(h))) = \mathsf{ev\text{-}pushout}(h).$$

Therefore we observe that it suffices to construct a homotopy $h \sim h'$ for any two functions $h, h' : \prod_{(x:X)} P(x)$ that come equipped with an identification

$$\mathsf{ev\text{-}pushout}(h) = \mathsf{ev\text{-}pushout}(h').$$

Now we recall from Remark 25.1.3 that this type of identifications is equivalent to the type of triples (K_A, K_B, K_S) consisting of

$$K_A : \prod_{(a:A)} h(i(a)) = h'(i(a))$$
$$K_B : \prod_{(b:B)} h(j(b)) = h'(j(b))$$

and a homotopy K_S witnessing that the square

$$\begin{array}{ccc} \mathsf{tr}_P(H(s), h(i(f(s)))) & \xrightarrow{\mathsf{ap}_{\mathsf{tr}_P(H(s))}(K_A(f(s)))} & \mathsf{tr}_P(H(s), h'(i(f(s)))) \\ {\scriptstyle \mathsf{apd}_h(H(s))}\Big\| & & \Big\|{\scriptstyle \mathsf{apd}_{h'}(H(s))} \\ h(j(g(s))) & \xrightarrow[K_B(g(s))]{} & h'(j(g(s))) \end{array}$$

commutes for every $s : S$. Note that from such an identification $K_S(s)$ we also obtain an identification

$$\mathsf{K}'_S(s) : \mathsf{tr}_{x \mapsto h(x)=h'(x)}(H(s), K_A(f(s))) = K_B(g(s)).$$

Indeed, by path inducgtion on $p : x = x'$ we obtain an identification $\mathsf{tr}_{x \mapsto h(x)=h'(x)}(p, q) = q'$, for any $p : x = x'$, any $q : h(x) = h'(x)$ and any $q' : h(x') = h'(x')$ for which the square

$$\begin{array}{ccc} \mathsf{tr}_P(p, h(x)) & \xrightarrow{\mathsf{ap}_{\mathsf{tr}_P(p)}(q)} & \mathsf{tr}_P(p, h'(x)) \\ {\scriptstyle \mathsf{apd}_h(p)}\Big\| & & \Big\|{\scriptstyle \mathsf{apd}_{h'}(p)} \\ h(x') & \xrightarrow[q']{} & h'(x') \end{array}$$

Now we see that the triple (K_A, K_B, K'_S) forms a dependent cocone on the family $x \mapsto h(x) = h'(x)$. Therefore we obtain a homotopy $h \sim h'$ as an application of the induction principle for pushouts at the family $x \mapsto h(x) = h'(x)$. □

25.2 Type families over pushouts

Given a pushout square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

with $H : i \circ f \sim j \circ g$, and a family $P : X \to \mathcal{U}$, we obtain

$$P \circ i : A \to \mathcal{U}$$
$$P \circ j : B \to \mathcal{U}$$

$$\lambda x.\, \mathrm{tr}_P(H(x)) : \textstyle\prod_{(x:S)} P(i(f(x))) \simeq P(j(g(x))).$$

Our goal in the current section is to show that the triple (P_A, P_B, P_S) consisting of $P_A :\equiv P \circ i$, $P_B :\equiv P \circ j$, and $P_S :\equiv \lambda x.\, \mathrm{tr}_P(H(x))$ characterizes the family P over X.

Definition 25.2.1 Consider a commuting square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

with $H : i \circ f \sim j \circ g$, where all types involved are in $\mathcal{U}$. The type $\mathsf{Desc}(\mathcal{S})$ of **descent data** for X, is defined defined to be the type of triples (P_A, P_B, P_S) consisting of

$$\begin{aligned} &P_A : A \to \mathcal{U} \\ &P_B : B \to \mathcal{U} \\ &P_S : \textstyle\prod_{(x:S)} P_A(f(x)) \simeq P_B(g(x)). \end{aligned}$$

Definition 25.2.2 Given a commuting square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

with $H : i \circ f \sim j \circ g$, we define the map

$$\mathsf{desc\text{-}fam}_{\mathcal{S}}(i, j, H) : (X \to \mathcal{U}) \to \mathsf{Desc}(\mathcal{S})$$

by $P \mapsto (P \circ i, P \circ j, \lambda x.\, \mathrm{tr}_P(H(x)))$.

Theorem 25.2.3 *Consider a pushout square*

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

with $H : i \circ f \sim j \circ g$, where all types involved are in $\mathcal{U}$, and suppose we have

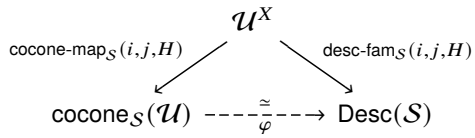
$$\begin{aligned} &P_A : A \to \mathcal{U} \\ &P_B : B \to \mathcal{U} \\ &P_S : \textstyle\prod_{(x:S)} P_A(f(x)) \simeq P_B(g(x)). \end{aligned}$$

Then the function

$$\mathsf{desc\text{-}fam}_{\mathcal{S}}(i,j,H) : (X \to \mathcal{U}) \to \mathsf{Desc}(\mathcal{S})$$

is an equivalence.

Proof By the 3-for-2 property of equivalences it suffices to construct an equivalence $\varphi : \mathsf{cocone}_{\mathcal{S}}(\mathcal{U}) \to \mathsf{Desc}(\mathcal{S})$ such that the triangle



commutes.

Since we have equivalences

$$\mathsf{equiv\text{-}eq} : \Big(P_A(f(x)) = P_B(g(x))\Big) \simeq \Big(P_A(f(x)) \simeq P_B(g(x))\Big)$$

for all $x : S$, we obtain by **??** an equivalence on the dependent products

$$\Big(\textstyle\prod_{(x:S)} P_A(f(x)) = P_B(g(x))\Big) \to \Big(\prod_{(x:S)} P_A(f(x)) \simeq P_B(g(x))\Big).$$

We define φ to be the induced map on total spaces. Explicitly, we have

$$\varphi :\equiv \lambda(P_A, P_B, K).\, (P_A, P_B, \lambda x.\, \mathsf{equiv\text{-}eq}(K(x))).$$

Then φ is an equivalence by Theorem 11.1.3, and the triangle commutes by **??**. □

Corollary 25.2.4 *Consider descent data (P_A, P_B, P_S) for a pushout square as in Theorem 25.2.3. Then the type of quadruples (P, e_A, e_B, e_S) consisting of a family $P : X \to \mathcal{U}$ equipped with two families of equivalences*

$$e_A : \textstyle\prod_{(a:A)} P_A(a) \simeq P(i(a))$$
$$e_B : \textstyle\prod_{(b:B)} P_B(a) \simeq P(j(b))$$

and a homotopy e_S witnessing that the square

$$\begin{array}{ccc} P_A(f(x)) & \xrightarrow{e_A(f(x))} & P(i(f(x))) \\ {\scriptstyle P_S(x)}\downarrow & & \downarrow{\scriptstyle \mathsf{tr}_P(H(x))} \\ P_B(g(x)) & \xrightarrow[e_B(g(x))]{} & P(j(g(x))) \end{array}$$

commutes, is contractible.

Proof The fiber of this map at (P_A, P_B, P_S) is equivalent to the type of quadruples (P, e_A, e_B, e_S) as described in the theorem, which are contractible by Theorem 10.4.6. □

25.3 The flattening lemma for pushouts

In this section we consider a pushout square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

with $H : i \circ f \sim j \circ g$, descent data

$$\begin{aligned} &P_A : A \to \mathcal{U} \\ &P_B : B \to \mathcal{U} \\ &P_S : \textstyle\prod_{(x:S)} P_A(f(x)) \simeq P_B(g(x)), \end{aligned}$$

and a family $P : X \to \mathcal{U}$ equipped with

$$\begin{aligned} e_A &: \textstyle\prod_{(a:A)} P_A(a) \simeq P(i(a)) \\ e_B &: \textstyle\prod_{(b:B)} P_B(a) \simeq P(j(b)) \end{aligned}$$

and a homotopy e_S witnessing that the square

$$\begin{array}{ccc} P_A(f(x)) & \xrightarrow{e_A(f(x))} & P(i(f(x))) \\ {\scriptstyle P_S(x)}\downarrow & & \downarrow{\scriptstyle \mathrm{tr}_P(H(x))} \\ P_B(g(x)) & \xrightarrow[e_B(g(x))]{} & P(j(g(x))) \end{array}$$

commutes.

Definition 25.3.1 We define a commuting square

$$\begin{array}{ccc} \sum_{(x:S)} P_A(f(x)) & \xrightarrow{g'} & \sum_{(b:B)} P_B(b) \\ {\scriptstyle f'}\downarrow & & \downarrow{\scriptstyle j'} \\ \sum_{(a:A)} P_A(a) & \xrightarrow[i']{} & \sum_{(x:X)} P(x) \end{array}$$

with a homotopy $H' : i' \circ f' \sim j' \circ g'$. We will write $\mathcal{S}'$ for the span

$$\textstyle\sum_{(a:A)} P_A(a) \xleftarrow{f'} \sum_{(x:S)} P_A(f(x)) \xrightarrow{g'} \sum_{(b:B)} P_B(b).$$

Construction We define

$$\begin{aligned} f' &:\equiv \mathsf{tot}_f\,(\lambda x.\,\mathsf{id}_{P_A(f\,(x))}) \\ g' &:\equiv \mathsf{tot}_g(e_S) \\ i' &:\equiv \mathsf{tot}_i(e_A) \\ j' &:\equiv \mathsf{tot}_j(e_B). \end{aligned}$$

Then it remains to construct a homotopy $H' : i' \circ f' \sim j' \circ g'$. In order to construct this homotopy, we have to construct an identification

$$(i(f(x)), e_A(y)) = (j(g(x)), e_B(e_S(y)))$$

for any $x : S$ and $y : P_A(f(x))$. Note that have the identification

$$\mathsf{eq\text{-}pair}(H(x), e_S(x, y)^{-1})$$

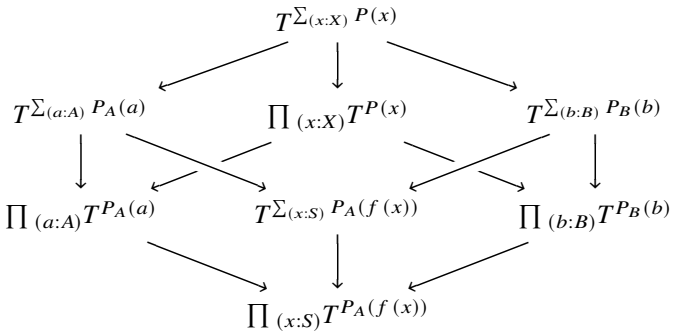
of this type. □

Lemma 25.3.2 (The flattening lemma) *The commuting square*

$$\begin{array}{ccc} \sum_{(x:S)} P_A(f(x)) & \xrightarrow{g'} & \sum_{(b:B)} P_B(b) \\ {\scriptstyle f'}\downarrow & & \downarrow{\scriptstyle j'} \\ \sum_{(a:A)} P_A(a) & \xrightarrow[i']{} & \sum_{(x:X)} P(x) \end{array}$$

is a pushout square.

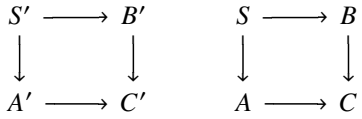
Proof To show that the square of total spaces satisfies the pullback property of pullbacks, note that we have a commuting cube



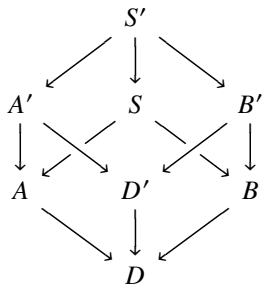
for any type T. In this cube, the vertical maps are all equivalences, and the bottom square is a pullback square by the dependent pullback property of pushouts. Therefore it follows that the top square is a pullback square. □

25.4 The universality theorem

Theorem 25.4.1 *Consider two pushout squares*

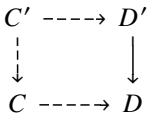


and a commuting cube



in which the back left and right squares are pullback squares. The following are equivalent:

(i) *The front left and right squares are pullback squares.*
(ii) *The induced commuting square*



is a pullback square.

25.5 The descent property for pushouts

In the previous section there was a significant role for families of equivalences, and we know by Theorems 22.5.2 and 22.5.3: families of equivalences indicate the presence of pullbacks. In this section we reformulate the results of the previous section using pullbacks where we used families of equivalences before, to obtain new and useful results. We begin by considering the type of descent data from the perspective of pullback squares.

Definition 25.5.1 Consider a span $\mathcal{S}$ from A to B, and a span $\mathcal{S}'$ from A' to

B'. A **cartesian transformation** of spans from $\mathcal{S}'$ to $\mathcal{S}$ is a diagram of the form

$$\begin{array}{ccccc} A' & \xleftarrow{f'} & S' & \xrightarrow{g'} & B' \\ {\scriptstyle h_A}\downarrow & & {\scriptstyle h_S}\downarrow & & \downarrow{\scriptstyle h_B} \\ A & \xleftarrow[f]{} & S & \xrightarrow[g]{} & B \end{array}$$

with $F : f \circ h_S \sim h_A \circ f'$ and $G : g \circ h_S \sim h_B \circ g'$, where both squares are pullback squares.

The type $\mathsf{cart}(\mathcal{S}', \mathcal{S})$ of cartesian transformation is the type of tuples

$$(h_A, h_S, h_B, F, G, p_f, p_g)$$

where $p_f : \mathsf{is\text{-}pullback}(h_S, h_A, F)$ and $p_g : \mathsf{is\text{-}pullback}(h_S, h_B, G)$, and we write

$$\mathsf{Cart}(\mathcal{S}) :\equiv \textstyle\sum_{(A',B':\mathcal{U})} \sum_{(\mathcal{S}':\mathsf{span}(A',B'))} \mathsf{cart}(\mathcal{S}', \mathcal{S}).$$

Lemma 25.5.2 *There is an equivalence*

$$\mathsf{cart\text{-}desc}_{\mathcal{S}} : \mathsf{Desc}(\mathcal{S}) \to \mathsf{Cart}(\mathcal{S}).$$

Proof Note that by Theorem 22.5.6 it follows that the types of triples (f', F, p_f) and (g', G, p_g) are equivalent to the types of families of equivalences

$$\textstyle\prod_{(x:S)} \mathsf{fib}_{h_S}(x) \simeq \mathsf{fib}_{h_A}(f(x))$$
$$\textstyle\prod_{(x:S)} \mathsf{fib}_{h_S}(x) \simeq \mathsf{fib}_{h_B}(g(x))$$

respectively. Furthermore, by **??** the types of pairs (S', h_S), (A', h_A), and (B', h_B) are equivalent to the types $S \to \mathcal{U}$, $A \to \mathcal{U}$, and $B \to \mathcal{U}$, respectively. Therefore it follows that the type $\mathsf{Cart}(\mathcal{S})$ is equivalent to the type of tuples $(Q, P_A, \varphi, P_B, P_S)$ consisting of

$$\begin{aligned} Q &: S \to \mathcal{U} \\ P_A &: A \to \mathcal{U} \\ P_B &: B \to \mathcal{U} \\ \varphi &: \textstyle\prod_{(x:S)} Q(x) \simeq P_A(f(x)) \\ P_S &: \textstyle\prod_{(x:S)} Q(x) \simeq P_B(g(x)). \end{aligned}$$

However, the type of φ is equivalent to the type $P_A \circ f = Q$. Thus we see that the type of pairs (Q, φ) is contractible, so our claim follows. □

Definition 25.5.3 We define an operation

$$\mathsf{cart\text{-}map}_{\mathcal{S}} : \left(\textstyle\sum_{(X':\mathcal{U})} X' \to X\right) \to \mathsf{Cart}(\mathcal{S}).$$

Construction Let $X' : \mathcal{U}$ and $h_X : X' \to X$. Then we define the types

$$A' :\equiv A \times_X X'$$
$$B' :\equiv B \times_X X'.$$

Next, we define a span $\mathcal{S}' :\equiv (S', f', g')$ from A' to B'. We take

$$S' :\equiv S \times_A A'$$
$$f' :\equiv \pi_2.$$

To define g', let $s : S$, let $(a, x', p) : A \times_X X'$, and let $q : f(s) = a$. Our goal is to construct a term of type $B \times_X X'$. We have $g(s) : B$ and $x' : X'$, so it remains to show that $j(g(s)) = h_X(x')$. We construct such an identification as a concatenation

$$j(g(s)) \xrightarrow{H(s)^{-1}} i(f(s)) \xrightarrow{\mathsf{ap}_i(q)} i(a) \xrightarrow{p} h_X(x').$$

To summarize, the map g' is defined as

$$g' :\equiv \lambda(s, (a, x', p), q).\, (g(s), x', H(s)^{-1} \cdot (\mathsf{ap}_i(q) \cdot p)).$$

Then we have commuting squares

$$\begin{array}{ccccc} A \times_X X' & \longleftarrow & S \times_A A' & \longrightarrow & B \times_X X' \\ \downarrow & & \downarrow & & \downarrow \\ A & \longleftarrow & S & \longrightarrow & B. \end{array}$$

Moreover, these squares are pullback squares by Theorem 22.5.8. □

The following theorem is analogous to Theorem 25.2.3.

Theorem 25.5.4 (The descent theorem for pushouts) *The operation* $\mathsf{cart\text{-}map}_{\mathcal{S}}$ *is an equivalence*

$$\left(\textstyle\sum_{(X':\mathcal{U})} X' \to X\right) \simeq \mathsf{Cart}(\mathcal{S})$$

Proof It suffices to show that the square

$$\begin{array}{ccc} X \to \mathcal{U} & \xrightarrow{\mathsf{desc\text{-}fam}_{\mathcal{S}}(i,j,H)} & \mathsf{Desc}(\mathcal{S}) \\ {\scriptstyle \mathsf{map\text{-}fam}_X}\downarrow & & \downarrow{\scriptstyle \mathsf{cart\text{-}desc}_{\mathcal{S}}} \\ \sum_{(X':\mathcal{U})} X' \to X & \xrightarrow[\mathsf{cart\text{-}map}_{\mathcal{S}}]{} & \mathsf{Cart}(\mathcal{S}) \end{array}$$

commutes. To see that this suffices, note that the operation $\mathsf{map\text{-}fam}_X$ is an equivalence by **??**, the operation $\mathsf{desc\text{-}fam}_{\mathcal{S}}(i, j, H)$ is an equivalence by Theorem 25.2.3, and the operation $\mathsf{cart\text{-}desc}_{\mathcal{S}}$ is an equivalence by Lemma 25.5.2.

To see that the square commutes, note that the composite

$$\mathsf{cart\text{-}map}_{\mathcal{S}} \circ \mathsf{map\text{-}fam}_X$$

takes a family $P : X \to \mathcal{U}$ to the cartesian transformation of spans

$$\begin{array}{ccccc} A \times_X \tilde{P} & \longleftarrow & S \times_A \left(A \times_X \tilde{P}\right) & \longrightarrow & B \times_X \tilde{P} \\ \downarrow{\scriptstyle \pi_1} & & \downarrow{\scriptstyle \pi_1} & & \downarrow{\scriptstyle \pi_1} \\ A & \longleftarrow & S & \longrightarrow & B, \end{array}$$

where $\tilde{P} :\equiv \sum_{(x:X)} P(x)$.

The composite

$$\mathsf{cart\text{-}desc}_{\mathcal{S}} \circ \mathsf{desc\text{-}fam}_X$$

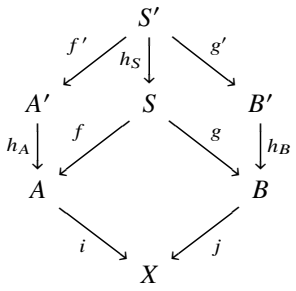
takes a family $P : X \to \mathcal{U}$ to the cartesian transformation of spans

$$\begin{array}{ccccc} \sum_{(a:A)} P(i(a)) & \longleftarrow & \sum_{(s:S)} P(i(f(s))) & \longrightarrow & \sum_{(b:B)} P(j(b)) \\ \downarrow & & \downarrow & & \downarrow \\ A & \longleftarrow & S & \longrightarrow & B \end{array}$$

These cartesian natural transformations are equal by Lemma 22.5.1 □

Since $\mathsf{cart\text{-}map}_{\mathcal{S}}$ is an equivalence it follows that its fibers are contractible. This is essentially the content of the following corollary.

Corollary 25.5.5 *Consider a diagram of the form*



with homotopies

$$\begin{aligned} F &: f \circ h_S \sim h_A \circ f' \\ G &: g \circ h_S \sim h_B \circ g' \\ H &: i \circ f \sim j \circ g, \end{aligned}$$

and suppose that the bottom square is a pushout square, and the top squares are

pullback squares. Then the type of tuples $((X', h_X), (i', I, p), (j', J, q), (H', C))$ *consisting of*

(i) *A type* $X' : \mathcal{U}$ *together with a morphism*

$$h_X : X' \to X,$$

(ii) *A map* $i' : A' \to X'$*, a homotopy* $I : i \circ h_A \sim h_X \circ i'$*, and a term* p *witnessing that the square*

$$\begin{array}{ccc} A' & \xrightarrow{i'} & X' \\ {\scriptstyle h_A}\downarrow & & \downarrow{\scriptstyle h_X} \\ A & \xrightarrow[i]{} & X \end{array}$$

is a pullback square.

(iii) *A map* $j' : B' \to X'$*, a homotopy* $J : j \circ h_B \sim h_X \circ j'$*, and a term* q *witnessing that the square*

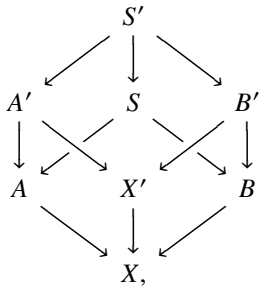
$$\begin{array}{ccc} B' & \xrightarrow{j'} & X' \\ {\scriptstyle h_B}\downarrow & & \downarrow{\scriptstyle h_X} \\ B & \xrightarrow[j]{} & X \end{array}$$

is a pullback square,

(iv) *A homotopy* $H' : i' \circ f' \sim j' \circ g'$*, and a homotopy*

$$C : (i \cdot F) \cdot ((I \cdot f') \cdot (h_X \cdot H')) \sim (H \cdot h_S) \cdot ((j \cdot G) \cdot (J \cdot g'))$$

witnessing that the cube

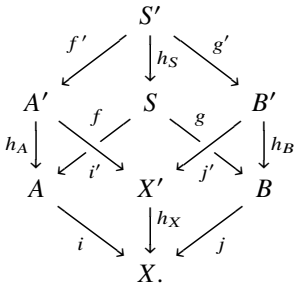


commutes,

is contractible.

The following theorem should be compared to the flattening lemma, Lemma 25.3.2.

Theorem 25.5.6 *Consider a commuting cube*



If each of the vertical squares is a pullback, and the bottom square is a pushout, then the top square is a pushout.

Proof By Theorem 22.5.3 we have families of equivalences

$$\begin{aligned} F &: \textstyle\prod_{(x:S)} \mathsf{fib}_{h_S}(x) \simeq \mathsf{fib}_{h_A}(f(x)) \\ G &: \textstyle\prod_{(x:S)} \mathsf{fib}_{h_S}(x) \simeq \mathsf{fib}_{h_B}(g(x)) \\ I &: \textstyle\prod_{(a:A)} \mathsf{fib}_{h_A}(a) \simeq \mathsf{fib}_{h_X}(i(a)) \\ J &: \textstyle\prod_{(b:B)} \mathsf{fib}_{h_B}(b) \simeq \mathsf{fib}_{h_X}(j(b)). \end{aligned}$$

Moreover, since the cube commutes we obtain a family of homotopies

$$K : \textstyle\prod_{(x:S)} I(f(x)) \circ F(x) \sim J(g(x)) \circ G(x).$$

We define the descent data (P_A, P_B, P_S) consisting of $P_A : A \to \mathcal{U}$, $P_B : B \to \mathcal{U}$, and $P_S : \prod_{(x:S)} P_A(f(x)) \simeq P_B(g(x))$ by

$$\begin{aligned} P_A(a) &:\equiv \mathsf{fib}_{h_A}(a) \\ P_B(b) &:\equiv \mathsf{fib}_{h_B}(b) \\ P_S(x) &:\equiv G(x) \circ F(x)^{-1}. \end{aligned}$$

We have

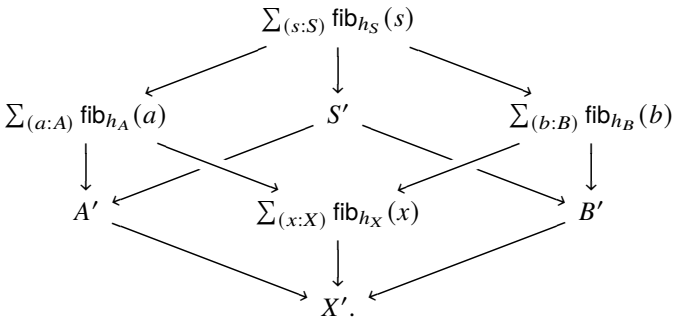
$$\begin{aligned} P &:\equiv \mathsf{fib}_{h_X} \\ e_A &:\equiv I \\ e_B &:\equiv J \\ e_S &:\equiv K. \end{aligned}$$

Now consider the diagram

$$\begin{array}{ccccc}
\sum_{(s:S)} \mathsf{fib}_{h_S}(s) & \longrightarrow & \sum_{(s:S)} \mathsf{fib}_{h_A}(f(s)) & \longrightarrow & \sum_{(b:B)} \mathsf{fib}_{h_B}(b) \\
\downarrow & & \downarrow & & \downarrow \\
\sum_{(a:A)} \mathsf{fib}_{h_A}(a) & \longrightarrow & \sum_{(a:A)} \mathsf{fib}_{h_A}(a) & \longrightarrow & \sum_{(x:X)} \mathsf{fib}_{h_X}(x)
\end{array}$$

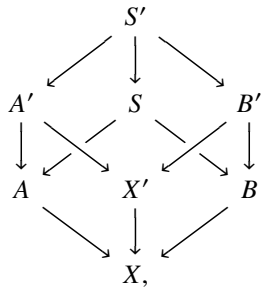
Since the top and bottom map in the left square are equivalences, we obtain from Exercise 23.1 that the left square is a pushout square. Moreover, the right square is a pushout by Lemma 25.3.2. Therefore it follows by Theorem 23.3.4 that the outer rectangle is a pushout square.

Now consider the commuting cube



We have seen that the top square is a pushout. The vertical maps are all equivalences, so the vertical squares are all pushout squares. Thus it follows from one more application of Theorem 23.3.4 that the bottom square is a pushout. □

Theorem 25.5.7 *Consider a commuting cube of types*



and suppose the vertical squares are pullback squares. Then the commuting

square

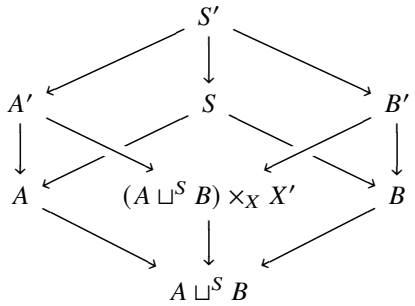
$$\begin{array}{ccc} A' \sqcup^{S'} B' & \longrightarrow & X' \\ \downarrow & & \downarrow \\ A \sqcup^{S} B & \longrightarrow & X \end{array}$$

is a pullback square.

Proof It suffices to show that the pullback

$$(A \sqcup^S B) \times_X X'$$

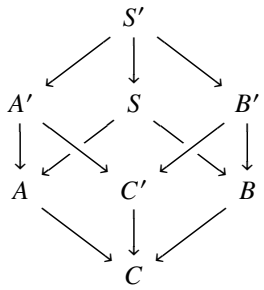
has the universal property of the pushout. This follows by the descent theorem, since the vertical squares in the cube



are pullback squares by Theorem 22.5.8. □

25.6 Applications of the descent theorem

Theorem 25.6.1 *Consider a commuting cube*



in which the bottom square is a pushout square. If the vertical sides are pullback squares, then for each $c : C$ the square of fibers

$$\begin{array}{ccc} \mathsf{fib}_{i\circ f\circ h_S}(c) & \longrightarrow & \mathsf{fib}_{j\circ g\circ h_S}(c) \longrightarrow \mathsf{fib}_{j\circ h_B}(c) \\ \downarrow & & \downarrow \\ \mathsf{fib}_{i\circ h_A}(c) & \longrightarrow & \mathsf{fib}_{h_C}(c) \end{array}$$

is a pushout square.

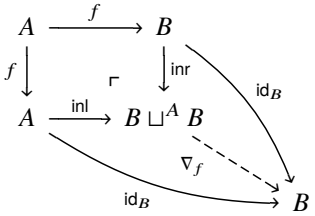
Exercises

25.1 Use the characterization of the circle as a pushout given in Example 23.3.3 to show that the square

$$\begin{array}{ccc} \mathbf{S}^1 + \mathbf{S}^1 & \xrightarrow{[\mathsf{id},\mathsf{id}]} & \mathbf{S}^1 \\ {\scriptstyle [\mathsf{id},\mathsf{id}]}\downarrow & & \downarrow{\scriptstyle \lambda t.\,(t,\mathsf{base})} \\ \mathbf{S}^1 & \xrightarrow[\lambda t.\,(t,\mathsf{base})]{} & \mathbf{S}^1 \times \mathbf{S}^1 \end{array}$$

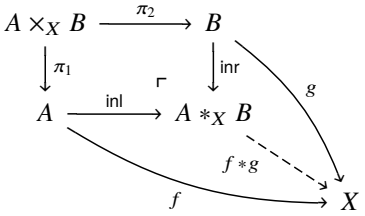
is a pushout square.

25.2 Let $f : A \to B$ be a map. The **codiagonal** ∇_f of f is the map obtained from the universal property of the pushout, as indicated in the diagram



Show that $\mathsf{fib}_{\nabla_f}(b) \simeq \Sigma(\mathsf{fib}_f(b))$ for any $b : B$.

25.3 Consider two maps $f : A \to X$ and $g : B \to X$. The **fiberwise join** $f * g$ is defined by the universal property of the pushout as the unique map rendering the diagram



The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

commutative, where $A *_X B$ is defined as a pushout, as indicated. Construct an equivalence

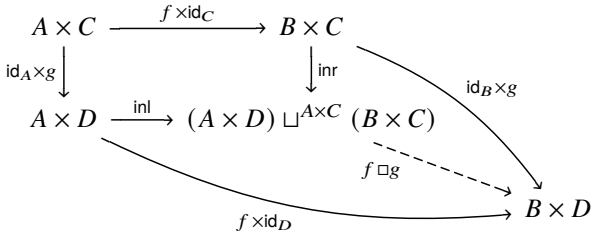
$$\mathsf{fib}_{f*g}(x) \simeq \mathsf{fib}_f(x) * \mathsf{fib}_g(x)$$

for any $x : X$.

25.4 Consider two maps $f : A \to B$ and $g : C \to D$. The **pushout-product**

$$f \square g : (A \times D) \sqcup^{A\times C} (B \times C) \to B \times D$$

of f and g is defined by the universal property of the pushout as the unique map rendering the diagram

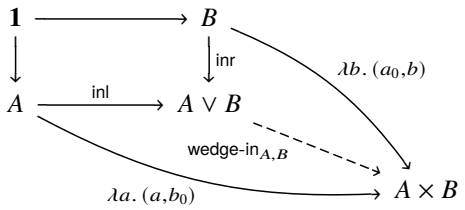


commutative. Construct an equivalence

$$\mathsf{fib}_{f\square g}(b, d) \simeq \mathsf{fib}_f(b) * \mathsf{fib}_g(d)$$

for all $b : B$ and $d : D$.

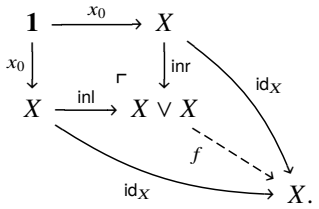
25.5 Let A and B be pointed types with base points $a_0 : A$ and $b_0 : B$. The **wedge inclusion** is defined as follows by the universal property of the wedge:



Show that the fiber of the wedge inclusion $A \vee B \to A \times B$ is equivalent to $\Omega(B) * \Omega(A)$.

25.6 Let $f : X \vee X \to X$ be the map defined by the universal property of the

wedge as indicated in the diagram



(a) Show that $\mathsf{fib}_f(x_0) \simeq \Sigma\Omega(X)$.
(b) Show that $\mathsf{cof}_f \simeq \Sigma X$.

25.7 Consider a pushout square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

(a) Show that if f is an embedding, then j is an embedding and the square is also a pullback square.
(b) Show that the following are equivalent:
   (i) The map f is surjective.
   (ii) The map j is surjective.

Note: In classical topology there are examples of *acyclic* spaces, which are non-contractible spaces X with the property that ΣX is contractible. Therefore we don't expect that if f is an embedding whenever j is an embedding.

26 Sequential colimits

Note: This chapter currently contains only the statements of the definitions and theorems, but no proofs. I hope to make a complete version available soon.

26.1 The universal property of sequential colimits

Type sequences are diagrams of the following form.

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots.$$

Their formal specification is as follows.

Definition 26.1.1 An **(increasing) type sequence** $\mathcal{A}$ consists of

$$A : \mathbb{N} \to \mathcal{U}$$
$$f : \prod_{(n:\mathbb{N})} A_n \to A_{n+1}.$$

In this section we will introduce the sequential colimit of a type sequence. The sequential colimit includes each of the types A_n, but we also identify each $x : A_n$ with its value $f_n(x) : A_{n+1}$. Imagine that the type sequence $A_0 \to A_1 \to A_2 \to \cdots$ defines a big telescope, with A_0 sliding into A_1, which slides into A_2, and so forth.

As usual, the sequential colimit is characterized by its universal property.

Definition 26.1.2 (i) A **(sequential) cocone** on a type sequence $\mathcal{A}$ with vertex B consists of

$$h : \prod_{(n:\mathbb{N})} A_n \to B$$
$$H : \prod_{(n:\mathbb{N})} h_n \sim h_{n+1} \circ H_n.$$

We write $\mathsf{cocone}(B)$ for the type of cocones with vertex X.

(ii) Given a cocone (h, H) with vertex B on a type sequence $\mathcal{A}$ we define the map

$$\mathsf{cocone\text{-}map}(h, H) : (B \to C) \to \mathsf{cocone}(C)$$

given by $f \mapsto (\lambda n.\, f \circ h_n, \lambda n.\, \lambda x.\, \mathsf{ap}_f(H_n(x)))$.

(iii) We say that a cocone (h, H) with vertex B is **colimiting** if $\mathsf{cocone\text{-}map}(h, H)$ is an equivalence for any type C.

Theorem 26.1.3 *Consider a cocone (h, H) with vertex B for a type sequence $\mathcal{A}$. The following are equivalent:*

(i) *The cocone (h, H) is colimiting.*

(ii) *The cocone (h, H) is inductive in the sense that for every type family $P : B \to \mathcal{U}$, the map*

$$\left(\prod_{(b:B)} P(b)\right) \to \sum_{(h:\prod_{(n:\mathbb{N})} \prod_{(x:A_n)} P(h_n(x)))} \prod_{(n:\mathbb{N})} \prod_{(x:A_n)} \mathsf{tr}_P(H_n(x), h_n(x)) = h_{n+1}(f_n(x))$$

given by

$$s \mapsto (\lambda n.\, s \circ h_n, \lambda n.\, \lambda x.\, \mathsf{apd}_s(H_n(x)))$$

has a section.

(iii) *The map in (ii) is an equivalence.*

26.2 The construction of sequential colimits

We construct sequential colimits using pushouts.

Definition 26.2.1 Let $\mathcal{A} \doteq (A, f)$ be a type sequence. We define the type A_∞ as a pushout

$$\begin{array}{ccc} \tilde{A} + \tilde{A} & \xrightarrow{[\mathsf{id}, \sigma_{\mathcal{A}}]} & \tilde{A} \\ {\scriptstyle [\mathsf{id},\mathsf{id}]}\downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ \tilde{A} & \xrightarrow[\mathsf{inl}]{} & A_\infty. \end{array}$$

Definition 26.2.2 The type A_∞ comes equipped with a cocone structure consisting of

$$\begin{aligned} \mathsf{seq\text{-}in} &: \textstyle\prod_{(n:\mathbb{N})} A_n \to A_\infty \\ \mathsf{seq\text{-}glue} &: \textstyle\prod_{(n:\mathbb{N})} \prod_{(x:A_n)} \mathsf{in}_n(x) = \mathsf{in}_{n+1}(f_n(x)). \end{aligned}$$

Construction We define

$$\begin{aligned} \mathsf{seq\text{-}in}(n, x) &\coloneqq \mathsf{inr}(n, x) \\ \mathsf{seq\text{-}glue}(n, x) &\coloneqq \mathsf{glue}(\mathsf{inl}(n, x))^{-1} \cdot \mathsf{glue}(\mathsf{inr}(n, x)). \end{aligned}$$

□

Theorem 26.2.3 *Consider a type sequence $\mathcal{A}$, and write $\tilde{A} \coloneqq \sum_{(n:\mathbb{N})} A_n$. Moreover, consider the map*

$$\sigma_{\mathcal{A}} : \tilde{A} \to \tilde{A}$$

defined by $\sigma_{\mathcal{A}}(n, a) \coloneqq (n + 1, f_n(a))$. Furthermore, consider a cocone (h, H) with vertex B. The following are equivalent:

(i) *The cocone (h, H) with vertex B is colimiting.*

(ii) *The defining square*

$$\begin{array}{ccc} \tilde{A} + \tilde{A} & \xrightarrow{[\mathsf{id}, \sigma_{\mathcal{A}}]} & \tilde{A} \\ {\scriptstyle [\mathsf{id},\mathsf{id}]}\downarrow & & \downarrow{\scriptstyle \lambda(n,x).\, h_n(x)} \\ \tilde{A} & \xrightarrow[\lambda(n,x).\, h_n(x)]{} & A_\infty, \end{array}$$

of A_∞ is a pushout square.

26.3 Descent for sequential colimits

Definition 26.3.1 The type of **descent data** on a type sequence $\mathcal{A} \doteq (A, f)$ is defined to be

$$\mathsf{Desc}(\mathcal{A}) :\equiv \textstyle\sum_{(B:\prod_{(n:\mathbb{N})} A_n \to \mathcal{U})} \prod_{(n:\mathbb{N})} \prod_{(x:A_n)} B_n(x) \simeq B_{n+1}(f_n(x)).$$

Definition 26.3.2 We define a map

$$\mathsf{desc\text{-}fam} : (A_\infty \to \mathcal{U}) \to \mathsf{Desc}(\mathcal{A})$$

by $B \mapsto (\lambda n.\, \lambda x.\, B(\mathsf{seq\text{-}in}(n, x)), \lambda n.\, \lambda x.\, \mathsf{tr}_B(\mathsf{seq\text{-}glue}(n, x)))$.

Theorem 26.3.3 *The map*

$$\mathsf{desc\text{-}fam} : (A_\infty \to \mathcal{U}) \to \mathsf{Desc}(\mathcal{A})$$

is an equivalence.

Definition 26.3.4 A **cartesian transformation** of type sequences from $\mathcal{A}$ to $\mathcal{B}$ is a pair (h, H) consisting of

$$h : \textstyle\prod_{(n:\mathbb{N})} A_n \to B_n$$
$$H : \textstyle\prod_{(n:\mathbb{N})} g_n \circ h_n \sim h_{n+1} \circ f_n,$$

such that each of the squares in the diagram

$$\begin{array}{ccccccc} A_0 & \xrightarrow{f_0} & A_1 & \xrightarrow{f_1} & A_2 & \xrightarrow{f_2} & \cdots \\ {\scriptstyle h_0}\downarrow & & {\scriptstyle h_1}\downarrow & & {\scriptstyle h_2}\downarrow & & \\ B_0 & \xrightarrow[g_0]{} & B_1 & \xrightarrow[g_1]{} & B_2 & \xrightarrow[g_2]{} & \cdots \end{array}$$

is a pullback square. We define

$$\mathsf{cart}(\mathcal{A}, \mathcal{B}) :\equiv \textstyle\sum_{(h:\prod_{(n:\mathbb{N})} A_n \to B_n)} \sum_{(H:\prod_{(n:\mathbb{N})} g_n \circ h_n \sim h_{n+1} \circ f_n)} \prod_{(n:\mathbb{N})} \mathsf{is\text{-}pullback}(h_n, f_n, H_n),$$

and we write

$$\mathsf{Cart}(\mathcal{B}) :\equiv \textstyle\sum_{(\mathcal{A}:\mathsf{Seq})} \mathsf{cart}(\mathcal{A}, \mathcal{B}).$$

Definition 26.3.5 We define a map

$$\mathsf{cart\text{-}map}(\mathcal{B}) : \left(\textstyle\sum_{(X':\mathcal{U})} X' \to X\right) \to \mathsf{Cart}(\mathcal{B}).$$

which associates to any morphism $h : X' \to X$ a cartesian transformation of type sequences into $\mathcal{B}$.

Theorem 26.3.6 *The operation* $\mathsf{cart\text{-}map}(\mathcal{B})$ *is an equivalence.*

26.4 The flattening lemma for sequential colimits

The flattening lemma for sequential colimits essentially states that sequential colimits commute with Σ.

Lemma 26.4.1 *Consider*

$$B : \prod_{(n:\mathbb{N})} A_n \to \mathcal{U}$$
$$g : \prod_{(n:\mathbb{N})} \prod_{(x:A_n)} B_n(x) \simeq B_{n+1}(f_n(x)).$$

and suppose $P : A_\infty \to \mathcal{U}$ is the unique family equipped with

$$e : \prod_{(n:\mathbb{N})} B_n(x) \simeq P(\mathsf{seq\text{-}in}(n,x))$$

and homotopies $H_n(x)$ witnessing that the square

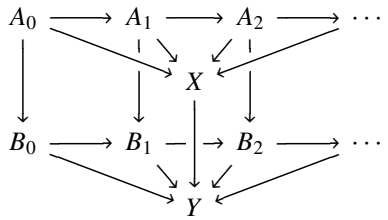
$$\begin{array}{ccc} B_n(x) & \xrightarrow{g_n(x)} & B_{n+1}(f_n(x)) \\ {\scriptstyle e_n(x)}\downarrow & & \downarrow{\scriptstyle e_{n+1}(f_n(x))} \\ P(\mathsf{seq\text{-}in}(n,x)) & \xrightarrow[\mathrm{tr}_P(\mathsf{seq\text{-}glue}(n,x))]{} & P(\mathsf{seq\text{-}in}(n+1,f_n(x))) \end{array}$$

commutes. Then $\sum_{(t:A_\infty)} P(t)$ satisfies the universal property of the sequential colimit of the type sequence

$$\sum_{(x:A_0)} B_0(x) \xrightarrow{\mathsf{tot}_{f_0}(g_0)} \sum_{(x:A_1)} B_1(x) \xrightarrow{\mathsf{tot}_{f_1}(g_1)} \sum_{(x:A_2)} B_2(x) \xrightarrow{\mathsf{tot}_{f_2}(g_2)} \cdots .$$

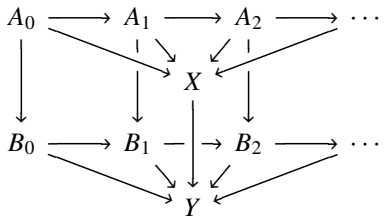
In the following theorem we rephrase the flattening lemma in using cartesian transformations of type sequences.

Theorem 26.4.2 *Consider a commuting diagram of the form*

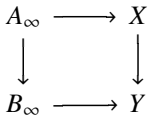


If each of the vertical squares is a pullback square, and Y is the sequential colimit of the type sequence B_n, then X is the sequential colimit of the type sequence A_n.

Corollary 26.4.3 *Consider a commuting diagram of the form*



If each of the vertical squares is a pullback square, then the square



is a pullback square.

26.5 Constructing the propositional truncation

The propositional truncation can be used to construct the image of a map, so we construct that first. We construct the propositional truncation of A via a construction called the **join construction**, as the colimit of the sequence of join-powers of A

$$A \longrightarrow A * A \longrightarrow A * (A * A) \longrightarrow \cdots$$

The join-powers of A are defined recursively on n, by taking[3]

$$\begin{aligned} A^{*0} &:= \emptyset \\ A^{*1} &:= A \\ A^{*(n+2)} &:= A * A^{*(n+1)}. \end{aligned}$$

We will write $A^{*\infty}$ for the colimit of the sequence

$$A \xrightarrow{\mathsf{inr}} A * A \xrightarrow{\mathsf{inr}} A * (A * A) \xrightarrow{\mathsf{inr}} \cdots.$$

The sequential colimit $A^{*\infty}$ comes equipped with maps $\mathsf{in\text{-}seq}_n : A^{*(n+1)} \to A^{*\infty}$, and we will write

$$\eta := \mathsf{in\text{-}seq}_0 : A \to A^{*\infty}.$$

[3] In this definition, the case $A^{*1} := A$ is slightly redundant because we have an equivalence

$$A * \emptyset \simeq A.$$

Nevertheless, it is nice to have that $A^{*1} \doteq A$.

Our goal is to show $A^{*\infty}$ is a proposition, and that $\eta : A \to A^{*\infty}$ satisfies the universal property of the propositional truncation of A. Before showing that $A^{*\infty}$ is indeed a proposition, let us show in two steps that for any proposition P, the map

$$(A^{*\infty} \to P) \to (A \to P)$$

is indeed an equivalence.

Lemma 26.5.1 *Suppose $f : A \to P$, where A is any type and P is a proposition. Then the precomposition function*

$$- \circ \mathsf{inr} : (A * B \to P) \to (B \to P)$$

is an equivalence, for any type B.

Proof Since the precomposition function

$$- \circ \mathsf{inr} : (A * B \to P) \to (B \to P)$$

is a map between propositions, it suffices to construct a map

$$(B \to P) \to (A * B \to P).$$

Let $g : B \to P$. Then the square

$$\begin{array}{ccc} A \times B & \xrightarrow{\mathsf{pr}_2} & B \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & P \end{array}$$

commutes since P is a proposition. Therefore we obtain a map $A * B \to P$ by the universal property of the join. □

Proposition 26.5.2 *Let A be a type, and let P be a proposition. Then the function*

$$- \circ \eta : (A^{*\infty} \to P) \to (A \to P)$$

is an equivalence.

Proof Since the map

$$- \circ \eta : (A^{*\infty} \to P) \to (A \to P)$$

is a map between propositions, it suffices to construct a map in the converse direction.

Let $f : A \to P$. First, we show by recursion that there are maps

$$f_n : A^{*(n+1)} \to P.$$

The map f_0 is of course just defined to be f. Given $f_n : A^{*(n+1)}$ we obtain $f_{n+1} : A * A^{*(n+1)} \to P$ by Lemma 26.5.1. Because P is assumed to be a proposition it is immediate that the maps f_n form a cocone with vertex P on the sequence

$$A \xrightarrow{\mathsf{inr}} A * A \xrightarrow{\mathsf{inr}} A * (A * A) \xrightarrow{\mathsf{inr}} \cdots .$$

From this cocone we obtain the desired map $(A^{*\infty} \to P)$. □

Proposition 26.5.3 *The type $A^{*\infty}$ is a proposition for any type A.*

Proof By Proposition 12.1.3 it suffices to show that

$$A^{*\infty} \to \mathsf{is\text{-}contr}(A^{*\infty}).$$

Since the type $\mathsf{is\text{-}contr}(A^{*\infty})$ is already known to be a proposition by Exercise 13.3, it follows from Proposition 26.5.2 that it suffices to show that

$$A \to \mathsf{is\text{-}contr}(A^{*\infty}).$$

Let $x : A$. To see that $A^{*\infty}$ is contractible it suffices by Exercise 26.3 to show that $\mathsf{inr} : A^{*n} \to A^{*(n+1)}$ is homotopic to the constant function $\mathsf{const}_{\mathsf{inl}(x)}$. However, we get a homotopy $\mathsf{const}_{\mathsf{inl}(x)} \sim \mathsf{inr}$ immediately from the path constructor glue. □

All the definitions are now in place to define the propositional truncation of a type.

Definition 26.5.4 For any type A we define the type

$$\|A\|_{-1} :\equiv A^{*\infty},$$

and we define $\eta : A \to \|A\|_{-1}$ to be the constructor $\mathsf{in\text{-}seq}_0$ of the sequential colimit $A^{*\infty}$. Often we simply write $\|A\|$ for $\|A\|_{-1}$.

The type $\|A\|_{-1}$ is a proposition by Proposition 26.5.3, and

$$\eta : A \to \|A\|_{-1}$$

satisfies the universal property of propositional truncation by Proposition 26.5.2.

Proposition 26.5.5 *The propositional truncation operation is functorial in the sense that for any map $f : A \to B$ there is a unique map $\|f\| : \|A\| \to \|B\|$ such that the square*

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle\eta}\downarrow & & \downarrow{\scriptstyle\eta} \\ \|A\| & \xrightarrow[\|f\|]{} & \|B\| \end{array}$$

commutes. Moreover, there are homotopies

$$\|\mathsf{id}_A\| \sim \mathsf{id}_{\|A\|}$$
$$\|g \circ f\| \sim \|g\| \circ \|f\|.$$

Proof The functorial action of propositional truncation is immediate by the universal property of propositional truncation. To see that the functorial action preserves the identity, note that the type of maps $\|A\| \to \|A\|$ for which the square

$$\begin{array}{ccc} A & \xrightarrow{\mathsf{id}} & A \\ {\scriptstyle\eta}\downarrow & & \downarrow{\scriptstyle\eta} \\ \|A\| & \dashrightarrow & \|A\| \end{array}$$

commutes is contractible. Since this square commutes for both $\|\mathsf{id}\|$ and for id, it must be that they are homotopic. The proof that the functorial action of propositional truncation preserves composition is similar. □

26.6 Proving type theoretical replacement

Our goal is now to show that the image of a map $f : A \to B$ from an essentially small type A into a locally small type B is again essentially small. This property is called the type theoretic replacement property. In order to prove this property, we have to find another construction of the image of a map. In order to make this construction, we define a join operation on maps.

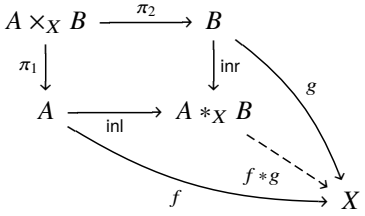
Definition 26.6.1 Consider two maps $f : A \to X$ and $g : B \to X$ with a common codomain X.

(i) We define the type $A *_X B$ as the pushout

$$\begin{array}{ccc} A \times_X B & \xrightarrow{\pi_2} & B \\ {\scriptstyle\pi_1}\downarrow & & \downarrow{\scriptstyle\mathsf{inr}} \\ A & \xrightarrow[\mathsf{inl}]{} & A *_X B. \end{array}$$

(ii) We define the **join** $f * g : A *_X B \to X$ to be the unique map for which

the diagram



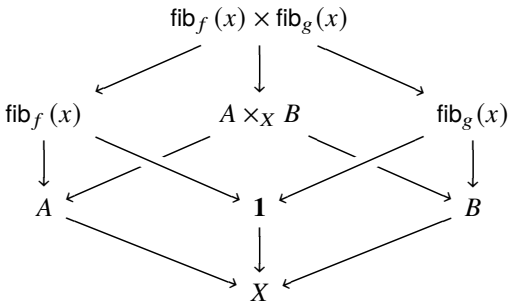
The reason to call the map $f * g$ the join of f and g is that the fiber of $f * g$ at any $x : X$ is equivalent to the join of the fibers of f and g at x.

Lemma 26.6.2 *Consider two maps $f : A \to X$ and $g : B \to X$. Then there is an equivalence*

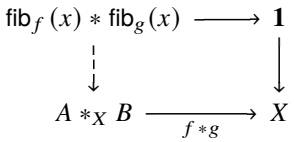
$$\mathsf{fib}_{f*g}(x) \simeq \mathsf{fib}_f(x) * \mathsf{fib}_g(x)$$

for any $x : X$.

Proof Consider the commuting cube



In this cube, the bottom square is a canonical pullback square. The two squares in the front are pullbacks by Lemma 22.4.1, and the top square is a pullback square by Lemma 22.3.1. Therefore it follows by Remark 24.1.5 that all the faces of this cube are pullback squares, and hence by Theorem 25.5.7 we obtain that the square



is a pullback square. Now the claim follows by the uniqueness of pullbacks, which was shown in Corollary 22.1.8. □

Lemma 26.6.3 *Consider a map $f : A \to X$, an embedding $m : U \to X$, and $h : \hom_X(f, m)$. Then the map*

$$\hom_X(f * g, m) \to \hom_X(g, m)$$

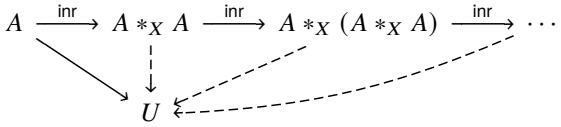
is an equivalence for any $g : B \to X$.

Proof Note that both types are propositions, so any equivalence can be used to prove the claim. Thus, we simply calculate

$$\begin{aligned}\hom_X(f * g, m) &\simeq \textstyle\prod_{(x:X)} \mathsf{fib}_{f*g}(x) \to \mathsf{fib}_m(x) \\ &\simeq \textstyle\prod_{(x:X)} \mathsf{fib}_f(x) * \mathsf{fib}_g(x) \to \mathsf{fib}_m(x) \\ &\simeq \textstyle\prod_{(x:X)} \mathsf{fib}_g(x) \to \mathsf{fib}_m(x) \\ &\simeq \hom_X(g, m).\end{aligned}$$

The first equivalence holds by Exercise 13.15; the second equivalence holds by Exercise 25.3, also using Theorem 13.4.1 and Exercise 13.12 (d) where we established that that pre- and postcomposing by an equivalence is an equivalence; the third equivalence holds by Lemma 26.5.1 and Exercise 13.12 (d); the last equivalence again holds by Exercise 13.15. □

For the construction of the image of $f : A \to X$ we observe that if we are given an embedding $m : U \to X$ and a map $(i, I) : \hom_X(f, m)$, then (i, I) extends uniquely along $\mathsf{inr} : A \to A *_X A$ to a map $\hom_X(f * f, m)$. This extension again extends uniquely along $\mathsf{inr} : A *_X A \to A *_X (A *_X A)$ to a map $\hom_X(f * (f * f), m)$ and so on, resulting in a diagram of the form



Definition 26.6.4 Suppose $f : A \to X$ is a map. Then we define the **fiberwise join powers**

$$f^{*n} : A^{*n}_X X.$$

Construction Note that the operation $(B, g) \mapsto (A *_X B, f * g)$ defines an endomorphism on the type

$$\textstyle\sum_{(B:\mathcal{U})} B \to X.$$

We also have $(\emptyset, \mathsf{ind}_\emptyset)$ and (A, f) of this type. For $n \geq 1$ we define

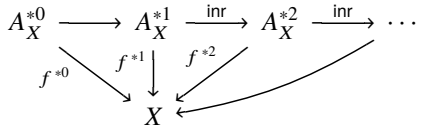
$$A^{*(n+1)}_X :\equiv A *_X A^{*n}_X$$

$$f^{*(n+1)} :\equiv f * f^{*n}. \qquad \square$$

Definition 26.6.5 We define $A_X^{*\infty}$ to be the sequential colimit of the type sequence

$$A_X^{*0} \longrightarrow A_X^{*1} \xrightarrow{\mathsf{inr}} A_X^{*2} \xrightarrow{\mathsf{inr}} \cdots.$$

Since we have a cocone



we also obtain a map $f^{*\infty} : A_X^{*\infty} \to X$ by the universal property of $A_X^{*\infty}$.

Lemma 26.6.6 *Let $f : A \to X$ be a map, and let $m : U \to X$ be an embedding. Then the function*

$$- \circ \mathsf{in\text{-}seq}_0 : \hom_X(f^{*\infty}, m) \to \hom_X(f, m)$$

is an equivalence.

Theorem 26.6.7 *For any map $f : A \to X$, the map $f^{*\infty} : A_X^{*\infty} \to X$ is an embedding that satisfies the universal property of the image inclusion of f.*

Lemma 26.6.8 *Consider a commuting square*

$$\begin{array}{ccc} A & \longrightarrow & B \\ \downarrow & & \downarrow \\ C & \longrightarrow & D. \end{array}$$

(i) *If the square is cartesian, B and C are essentially small, and D is locally small, then A is essentially small.*
(ii) *If the square is cocartesian, and A, B, and C are essentially small, then D is essentially small.*

Corollary 26.6.9 *Suppose $f : A \to X$ and $g : B \to X$ are maps from essentially small types A and B, respectively, to a locally small type X. Then $A \times_X B$ is again essentially small.*

Lemma 26.6.10 *Consider a type sequence*

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots$$

where each A_n is essentially small. Then its sequential colimit is again essentially small.

Theorem 26.6.11 *For any map $f : A \to B$ from an essentially small type A into a locally small type B, the image of f is again essentially small.*

Corollary 26.6.12 *Consider a $\mathcal{U}$-small type A, and an equivalence relation R over A valued in the $\mathcal{U}$-small propositions. Then the set quotient A/R is essentially small.*

Exercises

26.1 Show that the sequential colimit of a type sequence

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots$$

is equivalent to the sequential colimit of its shifted type sequence

$$A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} A_3 \xrightarrow{f_3} \cdots .$$

26.2 Let $P_0 \longrightarrow P_1 \longrightarrow P_2 \longrightarrow \cdots$ be a sequence of propositions. Show that

$$\mathsf{colim}_n(P_n) \simeq \exists_{(n:\mathbb{N})} P_n.$$

26.3 Consider a type sequence

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots$$

and suppose that $f_n \sim \mathsf{const}_{a_{n+1}}$ for some $a_n : \prod_{(n:\mathbb{N})} A_n$. Show that the sequential colimit is contractible.

26.4 Define the ∞-sphere $\mathbf{S}^\infty$ as the sequential colimit of

$$\mathbf{S}^0 \xrightarrow{f_0} \mathbf{S}^1 \xrightarrow{f_1} \mathbf{S}^2 \xrightarrow{f_2} \cdots$$

where $f_0 : \mathbf{S}^0 \to \mathbf{S}^1$ is defined by $f_0(\mathsf{false}) \doteq \mathsf{inl}(\star)$ and $f_0(\mathsf{true}) \doteq \mathsf{inr}(\star)$, and $f_{n+1} : \mathbf{S}^{n+1} \to \mathbf{S}^{n+2}$ is defined as $\Sigma(f_n)$. Use Exercise 26.3 to show that $\mathbf{S}^\infty$ is contractible.

26.5 Consider a type sequence

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots$$

in which $f_n : A_n \to A_{n+1}$ is weakly constant in the sense that

$$\prod_{(x,y:A_n)} f_n(x) = f_n(y)$$

Show that A_∞ is a mere proposition.

26.6 Show that $\mathbb{N}$ is the sequential colimit of

$$\mathsf{Fin}_0 \xrightarrow{\mathsf{inl}} \mathsf{Fin}_1 \xrightarrow{\mathsf{inl}} \mathsf{Fin}_2 \xrightarrow{\mathsf{inl}} \cdots .$$

27 Homotopy groups of types

27.1 Pointed types

Definition 27.1.1 A **pointed type** is a pair (A, a) consisting of a type A and a term $a : A$. The type of all pointed types in a universe $\mathcal{U}$ is defined to be

$$\mathcal{U}_* :\equiv \sum_{(X:\mathcal{U})} X.$$

Definition 27.1.2 Consider two pointed types (A, a) and (B, b). A **pointed map** from (A, a) to (B, b) is a pair (f, p) consisting of a function $f : A \to B$ and an identification $p : f(a) = b$. We write

$$A \to_* B :\equiv \sum_{(f:A\to B)} f(a) = b$$

for the type of all pointed maps from (A, a) to (B, b), leaving the base point implicit.

Since we have a type $\mathcal{U}_*$ of *all* pointed types in a universe $\mathcal{U}$, we can start defining operations on $\mathcal{U}_*$. An important example of such an operation is to take the loop space of a pointed type.

Definition 27.1.3 We define the **loop space** operation $\Omega : \mathcal{U}_* \to \mathcal{U}_*$

$$\Omega(A, a) :\equiv \big((a = a), \mathsf{refl}_a\big).$$

We can even go further and define the *iterated loop space* of a pointed type. Note that this definition could not be given in type theory if we didn't have universes.

Definition 27.1.4 Given a pointed type (A, a) and a natural number n, we define the n-th loop space $\Omega^n(A, a)$ by induction on $n : \mathbb{N}$, taking

$$\Omega^0(A, a) :\equiv (A, a)$$
$$\Omega^{n+1}(A, a) :\equiv \Omega(\Omega^n(A, a)).$$

27.2 The suspension-loop space adjunction

We get an even better version of the universal property of ΣX if we know in advance that the type X is a pointed type: on pointed types, the suspension functor is left adjoint to the loop space functor. This property manifests itself in the setting of pointed types, so we first give some definitions regarding pointed types.

Definition 27.2.1 (i) A pointed type consists of a type X equipped with a base point $x : X$. We will write $\mathcal{U}_*$ for the type $\sum_{(X:\mathcal{U})} X$ of all pointed types.

(ii) Let $(X, *_X)$ be a pointed type. A **pointed family** over $(X, *_X)$ consists of a type family $P : X \to \mathcal{U}$ equipped with a base point $*_P : P(*_X)$.

(iii) Let $(P, *_P)$ be a pointed family over $(X, *_X)$. A **pointed section** of $(P, *_P)$ consists of a dependent function $f : \prod_{(x:X)} P(x)$ and an identification $p : f(*_X) = *_P$. We define the **pointed Π-type** to be the type of pointed sections:

$$\textstyle\prod^*_{(x:X)} P(x) :\equiv \sum_{(f:\prod_{(x:X)} P(x))} f(*_X) = *_P$$

In the case of two pointed types X and Y, we may also view Y as a pointed family over X. In this case we write $X \to_* Y$ for the type of pointed functions.

(iv) Given any two pointed sections f and g of a pointed family P over X, we define the type of pointed homotopies

$$f \sim_* g :\equiv \Pi^*_{(x:X)} f(x) = g(x),$$

where the family $x \mapsto f(x) = g(x)$ is equipped with the base point $p \cdot q^{-1}$.

Remark 27.2.2 Since pointed homotopies are defined as certain pointed sections, we can use the same definition of pointed homotopies again to consider pointed homotopies between pointed homotopies, and so on.

Example 27.2.3 For any type X, the suspension ΣX is a pointed type where the base point is taken to be the north pole $\mathbb{N}$.

Definition 27.2.4 Let X be a pointed type with base point x. We define the **loop space** $\Omega(X, x)$ of X at x to be the pointed type $x = x$ with base point refl_x.

Definition 27.2.5 The loop space operation Ω is *functorial* in the sense that

(i) For every pointed map $f : X \to_* Y$ there is a pointed map

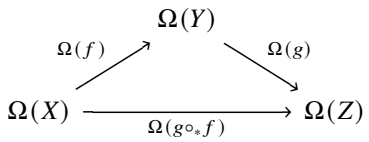
$$\Omega(f) : \Omega(X) \to_* \Omega(Y),$$

defined by $\Omega(f)(\omega) :\equiv p_f \cdot \mathsf{ap}_f(\omega) \cdot p_f^{-1}$, which is base point preserving by $\mathsf{right\text{-}inv}(p_f)$.

(ii) For every pointed type X there is a pointed homotopy

$$\Omega(\mathsf{id}^*_X) \sim_* \mathsf{id}^*_{\Omega(X)}.$$

(iii) For any two pointed maps $f : X \to_* Y$ and $g : Y \to_* X$, there is a pointed homotopy witnessing that the triangle



of pointed types commutes.

In order to introduce the suspension-loop space adjunction, we also need to construct the functorial action of suspension.

Definition 27.2.6 (i) Given a pointed map $f : X \to_* Y$, we define a map

$$\Sigma(f) : \Sigma X \to_* \Sigma Y$$

Definition 27.2.7 We define a pointed map

$$\varepsilon_X : X \to_* \Omega(\Sigma X)$$

for any pointed type X. This map is called the **counit** of the suspension-loop space adjunction. Moreover, ε is natural in X in the sense that for any pointed map $f : X \to_* Y$ we have a commuting square

$$\begin{array}{ccc} X & \xrightarrow{\varepsilon_X} & \Omega(\Sigma X) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \Omega(\Sigma f)} \\ Y & \xrightarrow[\varepsilon_Y]{} & \Omega(\Sigma Y) \end{array}$$

Construction The underlying map of ε_X takes $x : X$ to the concatenation

$$\mathsf{N} \overset{\mathsf{merid}(x)}{=\!=} \mathsf{S} \overset{\mathsf{merid}(*_X)^{-1}}{=\!=} \mathsf{N}.$$

This map preserves the base point, since $\mathsf{merid}(*_X) \cdot \mathsf{merid}(*_X)^{-1} = \mathsf{refl}_{\mathsf{N}}$. □

Definition 27.2.8 (i) For any pointed type X, we define the **pointed identity function** $\mathsf{id}^*_X :\equiv (\mathsf{id}_X, \mathsf{refl}_*)$.

(ii) For any two pointed maps $f : X \to_* Y$ and $g : Y \to_* Z$, we define the **pointed composite**

$$g \circ_* f :\equiv (g \circ f, \mathsf{ap}_g(p_f) \cdot p_g).$$

Definition 27.2.9 Given two pointed types X and Y, a pointed map from X to Y is a pair (f, p) consisting of a map $f : X \to Y$ and a path $p : f(x_0) = y_0$ witnessing that f preserves the base point. We write

$$X \to_* Y$$

for the type of **pointed maps** from X to Y. The type $X \to_* Y$ is itself a pointed type, with base point $(\mathsf{const}_{y_0}, \mathsf{refl}_{y_0})$.

Now suppose that we have a pointed map $f : \Sigma X \to_* Y$ with $p : f(x_0) = y_0$. Then the composite

$$X \xrightarrow{\varepsilon_X} \Omega(\Sigma X) \xrightarrow{\Omega(f)} \Omega(Y)$$

yields a pointed map $\tilde{f} : X \to \Omega(Y)$. Therefore we obtain a map

$$\tau_{X,Y} : (\Sigma X \to_* Y) \to (X \to_* \Omega(Y)).$$

It is not hard to see that also $\tau_{X,Y}$ is pointed. We leave this to the reader. The following theorem is also called the adjointness of the suspension and loop space functors. This is an extremely important relation that pops up in many calculations of homotopy groups.

Theorem 27.2.10 *Let X and Y be pointed types. Then the pointed map*

$$\tau_{X,Y} : (\Sigma X \to_* Y) \to_* (X \to_* \Omega(Y))$$

is an equivalence. Moreover, τ is pointedly natural in X and Y.

27.3 Set truncation

Lemma 27.3.1 *For each type A, the relation $I_{(-1)} : A \to (A \to \mathsf{Prop})$ given by*

$$I_{(-1)}(x, y) :\equiv \|x = y\|$$

is an equivalence relation.

Proof For every $x : A$ we have $|\mathsf{refl}_x| : \|x = x\|$, so the relation is reflexive. To see that the relation is symmetric note that by the universal property of propositional truncation there is a unique map $\|\mathsf{inv}\| : \|x = y\| \to \|y = x\|$ for which the square

$$\begin{array}{ccc} (x = y) & \xrightarrow{\mathsf{inv}} & (y = x) \\ {\scriptstyle |-|}\downarrow & & \downarrow{\scriptstyle |-|} \\ \|x = y\| & \xrightarrow[\|\mathsf{inv}\|]{} & \|y = x\| \end{array}$$

commutes. This shows that the relation is symmetric. Similarly, we show by the universal property of propositional truncation that the relation is transitive. □

Definition 27.3.2 For each type A we define the **set truncation**

$$\|A\|_0 :\equiv A/I_{(-1)},$$

and the unit of the set truncation is defined to be the quotient map.

Theorem 27.3.3 *For each type A, the set truncation satisfies the universal property of the set truncation.*

27.4 Homotopy groups

Definition 27.4.1 For $n \geq 1$, the **n-th homotopy group** of a type X at a base point $x : X$ consists of the type

$$|\pi_n(X, x)| :\equiv \|\Omega^n(X, x)\|_0$$

equipped with the group operations inherited from the path operations on $\Omega^n(X, x)$. Often we will simply write $\pi_n(X)$ when it is clear from the context what the base point of X is.

For $n \doteq 0$ we define $\pi_0(X, x) :\equiv \|X\|_0$.

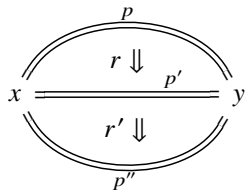
Example 27.4.2 In **??** we established that $\Omega(\mathbf{S}^1) \simeq \mathbb{Z}$. It follows that

$$\pi_1(\mathbf{S}^1) = \mathbb{Z} \qquad \text{and} \qquad \pi_n(\mathbf{S}^1) = 0 \qquad \text{for } n \geq 2.$$

Furthermore, we have seen in **??** that $\|\mathbf{S}^1\|_0$ is contractible. Therefore we also have $\pi_0(\mathbf{S}^1) = 0$.

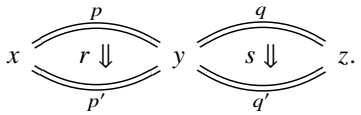
27.5 The Eckmann-Hilton argument

Given a diagram of identifications



in a type A, where $r : p = p'$ and $r' : p' = p''$, we obtain by concatenation an identification $r \cdot r' : p = p''$. This operation on identifications of identifications is sometimes called the **vertical concatenation**, because there is also a *horizontal* concatenation operation.

Definition 27.5.1 Consider identifications of identifications $r : p = p'$ and $s : q = q'$, where $p, p' : x = y$, and $q, q' : y = z$ are identifications in a type A, as indicated in the diagram



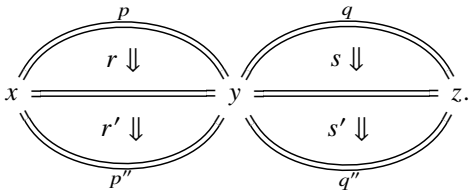
We define the **horizontal concatenation** $r \mathbin{\cdot_h} s : p \cdot q = p' \cdot q'$ of r and s.

Proof First we induct on r, so it suffices to define $\mathsf{refl}_p \mathbin{\cdot_h} s : p \cdot q = p \cdot q'$. Next, we induct on p, so it suffices to define $\mathsf{refl}_{\mathsf{refl}_y} \mathbin{\cdot_h} s : \mathsf{refl}_y \cdot q = \mathsf{refl}_y \cdot q'$. Since $\mathsf{refl}_y \cdot q \doteq q$ and $\mathsf{refl}_y \cdot q' \doteq q'$, we take $\mathsf{refl}_{\mathsf{refl}_y} \mathbin{\cdot_h} s :\equiv s$. □

Lemma 27.5.2 *Horizontal concatenation satisfies the left and right unit laws.*

In the following lemma we establish the **interchange law** for horizontal and vertical concatenation.

Lemma 27.5.3 *Consider a diagram of the form*



Then there is an identification

$$(r \cdot r') \mathbin{\cdot_h} (s \cdot s') = (r \mathbin{\cdot_h} s) \cdot (r' \mathbin{\cdot_h} s').$$

Proof We use path induction on both r and r', followed by path induction on p. Then it suffices to show that

$$(\mathsf{refl}_{\mathsf{refl}_y} \cdot \mathsf{refl}_{\mathsf{refl}_y}) \mathbin{\cdot_h} (s \cdot s') = (\mathsf{refl}_{\mathsf{refl}_y} \mathbin{\cdot_h} s) \cdot (\mathsf{refl}_{\mathsf{refl}_y} \mathbin{\cdot_h} s').$$

Using the computation rules, we see that this reduces to

$$s \cdot s' = s \cdot s',$$

which we have by reflexivity. □

Theorem 27.5.4 *For $n \geq 2$, the n-th homotopy group is abelian.*

Proof Our goal is to show that

$$\prod_{(r,s:\pi_2(X))} r \cdot s = s \cdot r.$$

Since we are constructing an identification in a set, we can use the universal property of 0-truncation on both r and s. Therefore it suffices to show that

$$\prod_{(r,s:\mathsf{refl}_{x_0}=\mathsf{refl}_{x_0})} |r|_0 \cdot |s|_0 = |s|_0 \cdot |r|_0.$$

Now we use that $|r|_0 \cdot |s|_0 \doteq |r \cdot s|_0$ and $|s|_0 \cdot |r|_0 \doteq |s \cdot r|_0$, to see that it suffices to show that $r \cdot s = s \cdot r$, for every $r, s : \mathsf{refl}_x = \mathsf{refl}_x$. Using the unit laws and the interchange law, this is a simple computation:

$$\begin{aligned} r \cdot s &= (r \cdot_h \mathsf{refl}_x) \cdot (\mathsf{refl}_x \cdot_h s) \\ &= (r \cdot \mathsf{refl}_x) \cdot_h (\mathsf{refl}_x \cdot s) \\ &= (\mathsf{refl}_x \cdot r) \cdot_h (s \cdot \mathsf{refl}_x) \\ &= (\mathsf{refl}_x \cdot_h s) \cdot (r \cdot_h \mathsf{refl}_x) \\ &= s \cdot r. \end{aligned}$$

□

Exercises

27.1 Show that the type of pointed families over a pointed type (X, x) is equivalent to the type

$$\textstyle\sum_{(Y:\mathcal{U}_*)} Y \to_* X.$$

27.2 Given two pointed types A and X, we say that A is a (pointed) retract of X if we have $i : A \to_* X$, a retraction $r : X \to_* A$, and a pointed homotopy $H : r \circ_* i \sim_* \mathsf{id}^*$.

(a) Show that if A is a pointed retract of X, then $\Omega(A)$ is a pointed retract of $\Omega(X)$.

(b) Show that if A is a pointed retract of X and $\pi_n(X)$ is a trivial group, then $\pi_n(A)$ is a trivial group.

27.3 Construct by path induction a family of maps

$$\textstyle\prod_{(A,B:\mathcal{U})} \prod_{(a:A)} \prod_{(b:B)} ((A, a) = (B, b)) \to \sum_{(e:A\simeq B)} e(a) = b,$$

and show that this map is an equivalence. In other words, an *identification of pointed types* is a base point preserving equivalence.

27.4 Let (A, a) and (B, b) be two pointed types. Construct by path induction a family of maps

$$\textstyle\prod_{(f,g:A\to B)} \prod_{(p:f(a)=b)} \prod_{(q:g(a)=b)} ((f, p)=(g, q)) \to \sum_{(H:f\sim g)} p = H(a)\cdot q,$$

and show that this map is an equivalence. In other words, an *identification of pointed maps* is a base point preserving homotopy.

27.5 Show that if $A \leftarrow S \rightarrow B$ is a span of pointed types, then for any pointed type X the square

$$\begin{array}{ccc} (A \sqcup^S B \to_* X) & \longrightarrow & (B \to_* X) \\ \downarrow & & \downarrow \\ (A \to_* X) & \longrightarrow & (S \to_* X) \end{array}$$

is a pullback square.

27.6 Let $f : A \to_* B$ be a pointed map. Show that the following are equivalent:

(i) f is an equivalence.

(ii) For any pointed type X, the precomposition map

$$- \circ_* f : (B \to_* X) \to_* (A \to_* X)$$

is an equivalence.

27.7 In this exercise we prove the suspension-loopspace adjunction.

(a) Construct a pointed equivalence

$$\tau_{X,Y} : (\Sigma(X) \to_* Y) \simeq_* (X \to \Omega(Y))$$

for any two pointed spaces X and Y.

(b) Show that for any $f : X \to_* X'$ and $g : Y' \to_* Y$, there is a pointed homotopy witnessing that the square

$$\begin{array}{ccc} (\Sigma(X') \to_* Y') & \xrightarrow{\tau_{X',Y'}} & (X' \to_* \Omega(Y')) \\ {\scriptstyle h \mapsto g \circ h \circ \Sigma(f)}\downarrow & & \downarrow{\scriptstyle h \mapsto \Omega(g) \circ h \circ f} \\ (\Sigma(X) \to_* Y) & \xrightarrow[\tau_{X,Y}]{} & (X \to_* \Omega(Y)) \end{array}$$

27.8 Show that if

$$\begin{array}{ccc} C & \longrightarrow & B \\ \downarrow & & \downarrow \\ A & \longrightarrow & X \end{array}$$

is a pullback square of pointed types, then so is

$$\begin{array}{ccc} \Omega(C) & \longrightarrow & \Omega(B) \\ \downarrow & & \downarrow \\ \Omega(A) & \longrightarrow & \Omega(X). \end{array}$$

27.9 (a) Show that if X is k-truncated, then its n-th homotopy group $\pi_n(X)$ is trivial for each choice of base point, and each $n > k$.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

(b) Show that if X is $(k+l)$-truncated, and for each $0 < i \leq l$ the $(k+i)$-th homotopy groups $\pi_{k+i}(X)$ are trivial for each choice of base point, then X is k-truncated.

It is consistent to assume that there are types for which all homotopy groups are trivial, but which aren't contractible nonetheless. Such types are called ∞**-connected**.

27.10 Consider a cospan

$$A \xrightarrow{f} X \xleftarrow{g} B$$

of pointed types and pointed maps between them.

(a) Define the type of pointed cones $\mathsf{cone}_*(C)$, where the vertex C is a pointed type. Also characterize its identity type.

(b) Define for any pointed cone (p, q, H) with vertex C the map

$$\mathsf{cone\text{-}map}_*(p, q, H) : (C' \to_* C) \to \mathsf{cone}_*(C').$$

Now we can say that the cone (p, q, H) satisfies the universal property of the pointed pullback of the cospan $A \to X \leftarrow A$ if this map is an equivalence for each pointed type C'.

(c) Now consider a commuting square

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X, \end{array}$$

where f and g are assumed to be pointed maps between pointed types (they come equipped with $\alpha : f(a_0) = x_0$ and $\beta : g(b_0) = x_0$, respectively). Show that if C is a pullback (in the usual unpointed sense), then C can be given the structure of a pointed pullback in a unique way, i.e., show that the type of

$$\begin{aligned} &c_0 : C \\ &\gamma : p(c_0) = a_0 \\ &\delta : q(c_0) = b_0 \\ &\varepsilon : \mathsf{ap}_f(\gamma) \cdot \alpha = H(c_0) \cdot (\mathsf{ap}_g(\delta) \cdot \beta) \end{aligned}$$

for which C satisfies the universal property of a pointed pullback, is contractible.

(d) Conclude that a commuting square of pointed types is a pointed

pullback square if and only if the underlying square of unpointed types is an ordinary pullback square.

27.11 Recall that $\mathcal{U}_*$ is the universe of pointed types.

(a) For any (A, a) and (B, b) in $\mathcal{U}_*$, write $(A, a) \simeq_* (B, b)$ for the type of **pointed equivalences** from A to B, i.e.,

$$(A, a) \simeq_* (B, b) :\equiv \sum_{(e:A\simeq B)} e(a) = b.$$

Show that the canonical map

$$\big((A, a) = (B, b)\big) \to \Big((A, a) \simeq (B, b)\Big)$$

sending $\mathsf{refl}_{(A,a)}$ to the pair $(\mathsf{id}, \mathsf{refl}_a)$, is an equivalence.

(b) Construct for any pointed type (X, x_0) an equivalence

$$\Big(\sum_{(P:X\to\mathcal{U})} P(x_0)\Big) \simeq \sum_{((A,a_0):\mathcal{U}_*)} (A, a_0) \to_* (X, x_0).$$

27.12 Consider a pointed connected type A with a finite fundamental group, and let B be a family over A of types with finitely many connected components. Show that the total space

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

also has finitely many connected components.

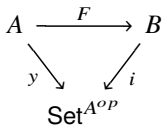
28 The classifying type of a group

28.1 The Rezk-completion of a category

Theorem 28.1.1 *Consider a functor $F : A \to B$ into a (Rezk-complete) category B. Then the following are equivalent:*

(i) *The functor F satisfies the universal property of the Rezk-completion of A.*

(ii) *The functor F is surjective on objects, and fits in a commuting diagram*



where i is an embedding.

28.2 Group actions

Definition 28.2.1 A G**-set** consists of a set X equipped with a group homomorphism

$$\varphi : G \to \mathrm{Aut}(X).$$

We will write $gx :\equiv \varphi(g, x)$ for any $g : G$ and $x : X$.

Remark 28.2.2 By the assumption that $\varphi : G^{op} \to \mathrm{Aut}(X)$ is a group homomorphism, we obtain that $1x = x$ and $(gh)x = g(hx)$ for any $x : X$.

Example 28.2.3 The underlying set of any group G is automatically a G-set, in the following way. Given an element $g : G$, we obtain the automorphism $x \mapsto gx$ on G.This G-set is called the **universal** G**-bundle**, and we will simply denote it by G.

28.3 The classifying type of a group

Our goal in this section is to construct the classifying type of an arbitrary group.

Definition 28.3.1 Consider a group G, and let X be a pointed 1-type. We say that X is the **classifying type** of G if X is connected and comes equipped with a group isomorphism

$$G \cong \Omega(BG).$$

If X is the classifying type of G, we will also write BG for X.

Remark 28.3.2 We speak of *the* classifying type of a group G, suggesting that it is unique. In Theorem 28.3.5 we will show that the classifying type of a group indeed exists and is unique.

The concept of Eilenberg-Mac Lane spaces is closely related.

Definition 28.3.3 Consider a group G and a pointed 1-type X that comes equipped with a group homomorphism

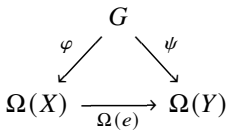
$$\varphi : \mathsf{Grp}(G, \Omega(X)).$$

We say that X satisfies the universal property of the Eilenberg-Mac Lane space $K(G, 1)$ if the map

$$f \mapsto \Omega(f) \circ \varphi : (X \to_* Y) \to \mathsf{Grp}(G, \Omega(Y))$$

is an equivalence for every pointed 1-type Y.

Remark 28.3.4 Any two pointed 1-types X and Y equipped with group homomorphisms $\varphi : \mathsf{Grp}(G, \Omega(X))$ and $\psi : \mathsf{Grp}(G, \Omega(Y))$ that both satisfy the universal property of the Eilenberg-Mac Lane space $K(G,1)$ are uniquely equivalent. Indeed, by their universal properties, we obtain that the type of pointed equivalences $e : X \simeq_* Y$ such that the triangle of group homomorphisms



commutes, is contractible.

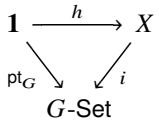
Since the pointed 1-types that satisfy the universal property of the Eilenberg-Mac Lane space $K(G,1)$ are uniquely unique in the above sense, we will simply write $K(G,1)$ for any such space.

Theorem 28.3.5 *Consider a group G and a pointed 1-type X. Then the following are equivalent:*

(i) *The type X is connected and it comes equipped with a group isomorphism*

$$\varphi : G \cong \Omega(X).$$

(ii) *The type X is connected and fits in a commuting triangle*



where the map pt_G points at the universal G-set, and the map i is an embedding.

(iii) *The type X satisfies the universal property of the Eileberg-Mac Lane space $K(G,1)$.*

We conclude that for every group G there is a unique pointed connected 1-type BG equipped with group isomorphism

$$G \cong \Omega(BG).$$

Proof We first show that (i) implies (ii). Suppose that X is connected and comes equipped with an isomorphism

$$\varphi : G \cong \Omega(X).$$

First we have construct a map $i : X \to G\text{-}\mathsf{Set}$. Given $x : X$, we define the G-Set

$i(x)$ to be $x_0 = x$. This is indeed a G-set, because for each $g : G$ we have the automorphism

$$\mathsf{concat}(\varphi(g)) : (x_0 = x) \simeq (x_0 = x),$$

and since φ is assumed to be a group isomorphism it follows that

$$\varphi(gh) \cdot p = (\varphi(g) \cdot \varphi(h)) \cdot p = \varphi(g) \cdot (\varphi(h) \cdot p)$$

and $\varphi(1) \cdot p = \mathsf{refl} \cdot p = p$.

Next, we have to show that the triangle in Item (ii) commutes. In other words, we have to show that $i(x_0)$ is the universal G-set. This is immediate from the isomorphism $\varphi : G \cong (x_0 = x_0)$.

It remains to show that the map i is an embedding. Since X is assumed to be connected, it suffices to show that

$$\mathsf{ap}_i : (x_0 = x) \to (i(x_0) = i(x))$$

is an equivalence, for any $x : X$. Equivalently, we have to show that $\sum_{(x:X)} i(x_0) = i(x)$ is contractible. Computing the identity type $i(x_0) = i(x)$, we see that it is equivalent to show that the type

$$\textstyle\sum_{(x:X)} \sum_{(e:(x_0=x_0)\simeq(x_0=x))} \prod_{(g:x_0=x_0)} e(g) = g \cdot e(\mathsf{refl})$$

is contractible. Since the type $\sum_{(p:x_0=x)} e(\mathsf{refl}) = p$ is contractible, it follows that the type above is equivalent to the type

$$\textstyle\sum_{(e:(x_0=x_0)\simeq(x_0=x_0))} \sum_{(q:e(\mathsf{refl})=\mathsf{refl})} \prod_{(g:x_0=x_0)} e(g) = g$$

Any e in the above type must be the identity equivalence. Therefore it follows—using the assumption that X is a 1-type, and therefore that the type $e(\mathsf{refl}) = \mathsf{refl}$ is a proposition—that the above type is contractible. This completes the proof that (i) implies (ii).

Next, we show that (ii) implies (iii). Suppose that X satisfies the condition in (ii). Then the map $\mathbf{1} \to X$ is surjective, so it follows from Theorem 15.2.5 that the embedding $i : X \to G\text{-}\mathsf{Set}$ satisfies the universal property of the image inclusion of $\mathsf{pt}_G : \mathbf{1} \to G\text{-}\mathsf{Set}$.

Now let Y be a pointed 1-type equipped with a group homomorphism

$$\psi : G \to \Omega Y.$$

Our goal is to show that there is a unique □

Example 28.3.6 The circle is the classifying type of $\mathbb{Z}$, i.e. we have

$$\mathbf{S}^1 = B\mathbb{Z} = K(\mathbb{Z}, 1).$$

Example 28.3.7 For any set A, the type

$$\textstyle\sum_{(X:\mathsf{Set})} \|A = X\|$$

is the classifying type of the automorphism group $\mathrm{Aut}(A)$.

Exercises

28.1 Using the classifying type BG of a group G, we say that a G-type is simply a map

$$BG \to \mathcal{U}.$$

If $X : BG \to \mathcal{U}$ is a G-type such that $X(*) = A$, we also say that A has the structure of a G-type, and we write

$$A /\!\!/ G :\equiv \textstyle\sum_{(x:BG)} X(x).$$

(a) Show that a G-set is equivalently defined as a map

$$BG \to \mathsf{Set}.$$

(b) For any group G, show that the type of (unpointed) equivalences $BG \simeq BG$ can be given the structure of a *transitive* $\mathrm{Aut}(G)$-type, i.e., an $\mathrm{Aut}(G)$-type such that the type

$$(BG \simeq BG) /\!\!/ \mathrm{Aut}(G)$$

is connected.

28.2 (Unsolved in HoTT, see [**HarpeDuff**]) Show that the type

$$\Sigma\mathsf{BAut}(\mathbb{N})$$

is contractible.

28.3 Show that for $n \neq 2, 6$, the type of groups merely equal to $\mathcal{S}_n$ is equivalent to $B\mathcal{S}_n$.

28.4 Show that the following are equivalent, for a group G:

(i) The group G is abelian.

(ii) There is an equivalence

$$(x = x) \simeq G$$

for every $x : BG$.

(iii) There is a homotopy witnessing that the square

$$\begin{array}{ccc} BG \times G & \xrightarrow{\mathsf{pr}_1} & BG \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle \delta} \\ BG & \xrightarrow[\delta]{} & BG \times BG \end{array}$$

commutes, such that it is also a pullback square.

28.5 A map $f : A \to B$ is called **weakly** 1**-constant** if it comes equipped with a homotopy $H : \prod_{(x,y:A)} f(x) = f(y)$ such that

$$\mu : \prod_{(x,y,z:A)} H(x,y) \cdot H(y,z) = H(x,z)$$
$$\nu : \prod_{(x:A)} H(x,x) = \mathsf{refl}$$

We will write $\mathsf{is\text{-}weakly\text{-}constant}_1(f)$ for the type of such triples (H, μ, ν).

(a) Show that every map $g : \|A\| \to B$ is weakly 1-constant. Use this to obtain a map

$$\alpha : \Big(\|A\| \to B\Big) \to \Big(\textstyle\sum_{(f:A\to B)} \mathsf{is\text{-}weakly\text{-}constant}_1(f)\Big).$$

(b) Show that if B is a 1-type, then the map α is an equivalence. In other words, show that every weakly 1-constant map $f : A \to B$ into a 1-type B has a unique extension

$$\begin{array}{ccc} A & \xrightarrow{f} & B. \\ {\scriptstyle \eta}\downarrow & \nearrow & \\ \|A\| & & \end{array}$$

28.6 Construct an equivalence

$$\Big(\textstyle\sum_{(X:\mathbb{F}_2)} \mathbb{F}_2^X\Big) \simeq BD_4,$$

where D_4 is the dihedral group of order 8.

29 The Hopf fibration

Our goal in this section is to construct the **Hopf fibration**. The Hopf fibration is a fiber sequence

$$\mathbf{S}^1 \hookrightarrow \mathbf{S}^3 \twoheadrightarrow \mathbf{S}^2.$$

More generally, we show that for any type A equipped with a multiplicative operation $\mu : A \to (A \to A)$ for which $\mu(x, -)$ and $\mu(-, x)$ are equivalences, there is a fiber sequence

$$A \hookrightarrow A * A \twoheadrightarrow \Sigma A.$$

The construction of this fiber sequence is known as the **Hopf construction**. We then get the Hopf fibration from the Hopf construction by using the multiplication on $\mathbf{S}^1$ constructed in Section 20.3 after we show that $\mathbf{S}^1 * \mathbf{S}^1 \simeq \mathbf{S}^3$.

We then introduce the long exact sequence of homotopy groups. The long exact sequence is an important tool to compute homotopy groups which applies to any fiber sequence

$$F \hookrightarrow E \twoheadrightarrow B.$$

In the case of the Hopf fibration, we will use the long exact sequence to show that

$$\pi_k(\mathbf{S}^3) = \pi_k(\mathbf{S}^2)$$

for any $k \geq 3$.

Since the Hopf fibration is closely related to the multiplication operation of the complex numbers on the unit circle, the Hopf fibration is sometimes also called the *complex* Hopf fibration. Indeed, there is also a *real* Hopf fibration

$$\mathbf{S}^0 \hookrightarrow \mathbf{S}^1 \twoheadrightarrow \mathbf{S}^1.$$

This is just the double cover of the circle. There is even a *quaternionic* Hopf fibration

$$\mathbf{S}^3 \hookrightarrow \mathbf{S}^7 \twoheadrightarrow \mathbf{S}^4,$$

which uses the multiplication of the quaternionic numbers on the unit sphere. The main difficulty in defining the quaternionic Hopf fibration in homotopy type theory is to define the quaternionic multiplication

$$\mathsf{mul}_{\mathbf{S}^3} : \mathbf{S}^3 \to (\mathbf{S}^3 \to \mathbf{S}^3).$$

The construction of the octonionic Hopf fibration

$$\mathbf{S}^7 \hookrightarrow \mathbf{S}^{15} \twoheadrightarrow \mathbf{S}^8$$

in homotopy type theory is still an open problem. Another open problem is to formalize Adams' theorem [1] in homotopy type theory, that there are *no* further fiber sequences of the form

$$\mathbf{S}^k \hookrightarrow \mathbf{S}^l \twoheadrightarrow \mathbf{S}^m,$$

for $k, l, m \geq 0$.

29.1 Fiber sequences

Definition 29.1.1 A **short sequence** of maps into a pointed type B with base point b consists of maps

$$F \xrightarrow{i} E \xrightarrow{p} B$$

equipped with a homotopy $p \circ i \sim \mathsf{const}_b$. We say that a short sequence as above is an **unpointed fiber sequence** if the commuting square

$$\begin{array}{ccc} F & \xrightarrow{i} & E \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle p} \\ \mathbf{1} & \xrightarrow[\mathsf{const}_b]{} & B \end{array}$$

is a pullback square.

Definition 29.1.2 A **short sequence** of pointed maps into a pointed type B with base point b consists of pointed maps

$$F \xrightarrow{i} E \xrightarrow{p} B$$

equipped with a pointed homotopy $p \circ i \sim_* \mathsf{const}_b$. We say that a short sequence as above is an **fiber sequence** if the commuting square

$$\begin{array}{ccc} F & \xrightarrow{i} & E \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle p} \\ \mathbf{1} & \xrightarrow[\mathsf{const}_b]{} & B \end{array}$$

is a pullback square.

29.2 The Hopf construction

The Hopf construction is a general construction of a fiber sequence

$$A \hookrightarrow A * A \twoheadrightarrow \Sigma A,$$

that applies to any H-space A. Our definition of an H-space is chosen such that it provides only the necessary structure to apply the Hopf construction. We give an unpointed and a pointed variant, and moreover we give a coherent variant that is more closely related to the traditional definition of an H-space.

Definition 29.2.1

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

(i) An **unpointed H-space** structure on a type A consists of a multiplicative operation

$$\mu : A \to (A \to A)$$

such that $\mu(x, -)$ and $\mu(-, x)$ are equivalences, for each $x : A$.

(ii) If A is a pointed type with base point $e : A$, then an **H-space** structure on A is an unpointed H-space structure on A equipped with an identification $\mu(e, e) = e$.

(iii) A **coherent H-space** structure on a pointed type A with base point $e : A$ consists of an unpointed H-space structure μ on A that satisfies the unit laws, i.e., μ comes equipped with identifications

$$\begin{aligned}\mathsf{left\text{-}unit}_\mu &: \mu(e, a) = a\\ \mathsf{right\text{-}unit}_\mu &: \mu(a, e) = a\\ \mathsf{coh\text{-}unit}_\mu &: \mathsf{left\text{-}unit}_\mu(e) = \mathsf{right\text{-}unit}_\mu(e).\end{aligned}$$

Example 29.2.2 The loop space $\Omega(A)$ of any pointed type is a coherent H-space, where the multiplication is given by path concatenation.

By an unpointed fiber sequence, we mean a sequence

$$F \xhookrightarrow{i} E \xrightarrow{p} B$$

where only the type B is assumed to be pointed (with base point b), and the square

$$\begin{array}{ccc} F & \xrightarrow{i} & E \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle p} \\ \mathbf{1} & \xrightarrow[\mathsf{const}_b]{} & B \end{array}$$

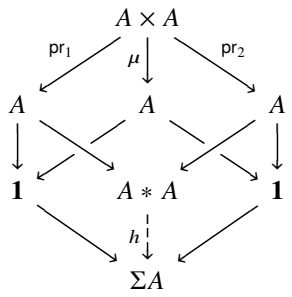
is a pullback square.

Theorem 29.2.3 (The Hopf construction) *Consider a type A equipped with an H-space structure μ. Then there is an unpointed fiber sequence*

$$A \hookrightarrow A * A \twoheadrightarrow \Sigma A.$$

If A and the H-space structure are pointed, then this unpointed fiber sequence is an fiber sequence.

Proof Note that there is a unique map $h : A * A \to \Sigma A$ such that the cube



commutes. Thus we see that we obtain a fiber sequence $A \hookrightarrow A * A \twoheadrightarrow \Sigma A$ if we show that the front two squares are pullback squares. By the descent theorem, Theorem 25.4.1, it suffices to show that the two squares in the back

$$\begin{array}{ccc} A \times A & \xrightarrow{\mu} & A \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow \\ A & \longrightarrow & \mathbf{1} \end{array} \qquad \begin{array}{ccc} A \times A & \xrightarrow{\mu} & A \\ {\scriptstyle \mathsf{pr}_2}\downarrow & & \downarrow \\ A & \longrightarrow & \mathbf{1} \end{array}$$

are pullback squares. We claim that in both squares, the multiplicative operation μ induces equivalences on the fibers, and hence both squares are pullbacks by Theorem 22.5.3. To see this, note that the induced map on fibers fit in commuting squares

$$\begin{array}{ccc} \mathsf{fib}_{\mathsf{pr}_1}(x) & \dashrightarrow & \mathsf{fib}_{\mathsf{const}_\star}(\star) \\ {\scriptstyle \simeq}\downarrow & & \downarrow{\scriptstyle \simeq} \\ A & \xrightarrow[\mu(x,-)]{} & A \end{array} \qquad \begin{array}{ccc} \mathsf{fib}_{\mathsf{pr}_2}(x) & \dashrightarrow & \mathsf{fib}_{\mathsf{const}_\star}(\star) \\ {\scriptstyle \simeq}\downarrow & & \downarrow{\scriptstyle \simeq} \\ A & \xrightarrow[\mu(-,x)]{} & A. \end{array}$$

The claim now follows, since we have assumed that $\mu(x, -)$ and $\mu(-, x)$ are equivalences for each $x : X$. □

Remark 29.2.4 The Hopf map h constructed in Theorem 29.2.3 is the unique map $A * A \to \Sigma A$ equipped with identifications

$$p : \mathsf{N} = h(\mathsf{inl}(x))$$
$$p' : \mathsf{S} = h(\mathsf{inr}(x'))$$

and an identification q witnessing that the square

$$\begin{array}{ccc} \mathsf{N} & \xrightarrow{p} & h(\mathsf{inl}(x)) \\ {\scriptstyle \mathsf{merid}(\mu(x,x'))}\Big\| & & \Big\|{\scriptstyle \mathsf{ap}_h(\mathsf{glue}(x,x'))} \\ \mathsf{S} & \xrightarrow[p']{} & h(\mathsf{inr}(x')) \end{array}$$

commutes.

Corollary 29.2.5 *There is a fiber sequence*

$$\mathbf{S}^1 \hookrightarrow \mathbf{S}^1 * \mathbf{S}^1 \twoheadrightarrow \mathbf{S}^2.$$

Proof By Theorem 29.2.3 it suffices to construct an H-space structure on $\mathbf{S}^1$. This H-space structure $\mathbf{S}^1 \times \mathbf{S}^1 \to \mathbf{S}^1$ is determined by the complex multiplication operation constructed in Definition 20.3.1. □

Lemma 29.2.6 *The join operation is associative*

Proof

$$\begin{array}{ccccc} A & \longleftarrow & A \times C & \longrightarrow & A \times C \\ \uparrow & & \uparrow & & \uparrow \\ A \times B & \longleftarrow & A \times B \times C & \longrightarrow & A \times C \\ \downarrow & & \downarrow & & \downarrow \\ B & \longleftarrow & B \times C & \longrightarrow & C \end{array}$$

□

Corollary 29.2.7 *There is an equivalence* $\mathbf{S}^1 * \mathbf{S}^1 \simeq \mathbf{S}^3$.

Theorem 29.2.8 *There is a fiber sequence* $\mathbf{S}^1 \hookrightarrow \mathbf{S}^3 \twoheadrightarrow \mathbf{S}^2$.

Lemma 29.2.9 *Suppose $f : G \to H$ is a group homomorphism, such that the sequence*

$$0 \longrightarrow G \xrightarrow{f} H \longrightarrow 0$$

is exact at G and H, where we write 0 for the trivial group consisting of just the unit element. Then f is a group isomorphism.

Corollary 29.2.10 *We have $\pi_2(\mathbf{S}^2) = \mathbb{Z}$, and for $k > 2$ we have $\pi_k(\mathbf{S}^2) = \pi_k(\mathbf{S}^3)$.*

29.3 The long exact sequence

Definition 29.3.1 A fiber sequence $F \hookrightarrow E \twoheadrightarrow B$ consists of:

(i) Pointed types F, E, and B, with base points x_0, y_0, and b_0 respectively,
(ii) Base point preserving maps $i : F \to_* E$ and $p : E \to_* B$, with $\alpha : i(x_0) = y_0$ and $\beta : p(y_0) = b_0$,
(iii) A pointed homotopy $H : \mathsf{const}_{b_0} \sim_* p \circ_* i$ witnessing that the square

$$\begin{array}{ccc} F & \xrightarrow{i} & E \\ \downarrow & & \downarrow{\scriptstyle p} \\ \mathbf{1} & \xrightarrow[\mathsf{const}_{b_0}]{} & B, \end{array}$$

commutes and is a pullback square.

Lemma 29.3.2 *Any fiber sequence $F \hookrightarrow E \twoheadrightarrow B$ induces a sequence of pointed maps*

$$\Omega(F) \xrightarrow{\Omega(i)} \Omega(E) \xrightarrow{\Omega(p)} \Omega(B) \xrightarrow{\partial} F \xrightarrow{i} E \xrightarrow{p} B,$$

in which every two consecutive maps form a fiber sequence.

Proof By taking pullback squares repeatedly, we obtain the diagram

$$\begin{array}{ccccc} \Omega(F) & \longrightarrow & \mathbf{1} & & \\ {\scriptstyle \Omega(i)}\downarrow & & \downarrow{\scriptstyle \mathsf{const}_{\mathsf{refl}_{b_0}}} & & \\ \Omega(E) & \xrightarrow{\Omega(p)} & \Omega(B) & \longrightarrow & \mathbf{1} \\ \downarrow & & {\scriptstyle \partial}\downarrow & & \downarrow{\scriptstyle \mathsf{const}_{y_0}} \\ \mathbf{1} & \xrightarrow[\mathsf{const}_{x_0}]{} & F & \xrightarrow{i} & E \\ & & \downarrow & & \downarrow{\scriptstyle p} \\ & & \mathbf{1} & \xrightarrow[\mathsf{const}_{b_0}]{} & B. \end{array}$$

□

Definition 29.3.3 We say that a consecutive pair of pointed maps between pointed sets

$$A \xrightarrow{f} B \xrightarrow{g} C$$

is **exact** at B if we have

$$\left(\exists_{(a:A)}\, f(a) = b\right) \leftrightarrow (g(b) = c)$$

for any $b : B$.

Remark 29.3.4 If a pair of consecutive pointed maps between pointed sets

$$A \xrightarrow{f} B \xrightarrow{g} C$$

is exact at B, it directly that $\mathrm{im}(f) = \mathsf{fib}_g(c)$. Indeed, such a pair of pointed maps is exact at B if and only if there is an equivalence $e : \mathrm{im}(f) \simeq \mathsf{fib}_g(c)$ such that the triangle

$$\begin{array}{ccc} \mathrm{im}(f) & \xrightarrow{e} & \mathsf{fib}_g(c) \\ & \searrow \quad \swarrow & \\ & B & \end{array}$$

commutes. In other words, $\mathrm{im}(f)$ and $\mathsf{fib}_g(c)$ are equal *as subsets of* B.

Lemma 29.3.5 *Suppose $F \hookrightarrow E \twoheadrightarrow B$ is a fiber sequence. Then the sequence*

$$\|F\|_0 \xrightarrow{\|i\|_0} \|E\|_0 \xrightarrow{\|p\|_0} \|B\|_0$$

is exact at $\|E\|_0$.

Proof To show that the image $\mathrm{im}\,\|i\|_0$ is the fiber $\mathsf{fib}_{\|p\|_0}(|b_0|_0)$, it suffices to construct a fiberwise equivalence

$$\textstyle\prod_{(x:\|E\|_0)} \|\mathsf{fib}_{\|i\|_0}(x)\|_{-1} \simeq \|p\|_0(x) = |b_0|_0.$$

By the universal property of 0-truncation it suffices to show that

$$\textstyle\prod_{(x:E)} \|\mathsf{fib}_{\|i\|_0}(|x|_0)\|_{-1} \simeq \|p\|_0(|x|_0) = |b_0|_0.$$

First we note that

$$\begin{aligned} \|p\|_0(|x|_0) = |b_0|_0 &\simeq |p(x)|_0 = |b_0|_0 \\ &\simeq \|p(x) = b_0\|_{-1}. \end{aligned}$$

Next, we note that

$$\begin{aligned} \mathsf{fib}_{\|i\|_0}(|x|_0) &\simeq \textstyle\sum_{(y:\|F\|_0)} \|i\|_0(y) = |x|_0 \\ &\simeq \|\textstyle\sum_{(y:F)} \|i\|_0(|y|_0) = |x|_0\|_0 \\ &\simeq \|\textstyle\sum_{(y:F)} |i(y)|_0 = |x|_0\|_0 \\ &\simeq \|\textstyle\sum_{(y:F)} \|i(y) = x\|_{-1}\|_0. \end{aligned}$$

Therefore it follows that

$$\begin{aligned} \|\mathsf{fib}_{\|i\|_0}(|x|_0)\|_{-1} &\simeq \|\textstyle\sum_{(y:F)} \|i(y) = x\|_{-1}\|_{-1} \\ &\simeq \|\textstyle\sum_{(y:F)} i(y) = x\|_{-1} \end{aligned}$$

Now it suffices to show that $\left(\sum_{(y:F)} i(y) = x\right) \simeq p(x) = b_0$. This follows by the pasting lemma of pullbacks

$$\begin{array}{ccc} (p(x) = b_0) & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow \\ F & \longrightarrow & E \\ \downarrow & & \downarrow \\ \mathbf{1} & \longrightarrow & B \end{array}$$

□

Theorem 29.3.6 *Any fiber sequence $F \hookrightarrow E \twoheadrightarrow B$ induces a long exact sequence on homotopy groups*

$$\cdots \to \pi_n(F) \xrightarrow{\pi_n(i)} \pi_n(E) \xrightarrow{\pi_n(p)} \pi_n(B) \dashrightarrow \pi_1(F) \xrightarrow{\pi_1(i)} \pi_1(E) \xrightarrow{\pi_1(p)} \pi_1(B) \to \pi_0(F) \xrightarrow{\pi_0(i)} \pi_0(E) \xrightarrow{\pi_0(p)} \pi_0(B)$$

29.4 The universal complex line bundle

Definition 29.4.1 A **coherently associative unpointed H-space** structure on a type X consists of

29.5 The finite dimensional complex projective spaces

Remark 29.5.1 The universe of types that are merely equal to the circle does not classify complex line bundles.

Exercises

29.1 Consider an unpointed H-space X of which the multiplication is associative, and consider $x : X$. Construct a unit for the multiplication, and show that it satisfies the coherent unit laws.

29.2 (a) Show that the type of associative unpointed H-space structures on bool is equivalent to bool.
(b) Show that the type of associative (pointed) H-space structures on $(\mathsf{bool}, \mathsf{true})$ is contractible.

29.3 Show that any fiber sequence

$$F \hookrightarrow E \twoheadrightarrow B$$

where the base points are $x_0 : B$, $y_0 : F$, and $z_0 : E$ induces a fiber sequence of connected components

$$\mathsf{BAut}(y_0) \hookrightarrow \mathsf{BAut}(z_0) \twoheadrightarrow \mathsf{BAut}(x_0).$$

29.4 Show that there is a fiber sequence

$$\mathbf{S}^3 \hookrightarrow \mathbf{S}^2 \twoheadrightarrow \|\mathbf{S}^2\|_2,$$

where the map $\mathbf{S}^2 \to \|\mathbf{S}^2\|_2$ is the unit of the 2-truncation.

29.5 Show that $\mathbb{C}\mathsf{P}^\infty$ is a coherent H-space. Note: the 2-sphere is not an H-space, and yet its 2-truncation is!

29.6 Construct for every group G of order $n+1$ a fiber sequence

$$G \hookrightarrow \bigvee_{(i:\mathsf{Fin}_{n^2})} \mathbf{S}^1 \twoheadrightarrow \bigvee_{(i:\mathsf{Fin}_n)} \mathbf{S}^1$$

29.7 Show that there is a fiber sequence

$$\mathbb{R}\mathsf{P}^\infty \hookrightarrow \mathbb{C}\mathsf{P}^\infty \twoheadrightarrow \mathbb{C}\mathsf{P}^\infty.$$

29.8 Show that the type of (small) fiber sequences is equivalent to the type of quadruples (B, P, b_0, x_0), consisting of

$$B : \mathcal{U}$$
$$P : B \to \mathcal{U}$$
$$b_0 : B$$
$$x_0 : P(b_0).$$

30 The real projective spaces

30.1 The type of 2-element sets

Theorem 30.1.1 *The type*

$$\textstyle\sum_{(X:\mathcal{U}_{\mathsf{bool}})} X$$

is contractible.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

Corollary 30.1.2 *For any 2-element type X, the map*

$$(\mathsf{bool} = X) \to X$$

given by $p \mapsto \mathsf{tr}_{\mathcal{T}}(p, \mathsf{true})$ is an equivalence.

30.2 Classifying real line bundles

30.3 The finite dimensional real projective spaces

31 Truncations

Truncation is a universal way of turning an arbitrary type into a k-truncated type. We have already seen the propositional truncation of a type X in Section 14, which is the proposition that X is merely inhabited, and the set truncation of X in Section 27.3, which is the set of connected components of X. The k-truncation is a generalization of the propositional truncation and the set truncation to an arbitrary truncation level k.

We construct the truncations by recursion on k. The base case $k \doteq -2$ is just the operation that sends a type X to the unit type $\mathbf{1}$, because up to equivalence there is only one contractible type. For the inductive step, we need to construct the $(k+1)$-truncation assuming that the k-truncation of an arbitrary type in a fixed universe $\mathcal{U}$ exists. Our construction of the $(k+1)$-truncation is a direct generalization of the construction of the set truncation as a set quotient, where we quotient out the equivalence relation

$$(x \sim y) :\equiv \|x = y\|_{-1}.$$

The idea is simple: if $\|X\|_{k+1}$ is to be the universal $(k+1)$-truncated type equipped with a map $|-|_{k+1} : X \to \|X\|_{k+1}$, then it has to be the case that

$$(|x|_{k+1} = |x'|_{k+1}) \simeq \|x = y\|_k.$$

We prove that this is indeed the case in Corollary 31.2.7.

The construction of the $(k+1)$-truncation as a quotient is different than the construction of the $(k+1)$-truncation that appears in [5] as a higher inductive type. This construction is based on the observation that a type X is k-truncated if and only if every map $\mathbf{S}^{k+1} \to X$ is constant. In other words, for every map $f : \mathbf{S}^{k+1} \to X$ into a k-type X, there is a point $x : X$ and a family of paths $p(t) : x = f(t)$. If we think of f as a 'wheel' in X, then x is the hub at the center of the wheel, and the paths $p(t)$ are the spokes. This leads to defining the k-truncation of a type X by the *hubs-and-spokes method*. In Section 31.3 we show that the k-truncation of a type is such a higher inductive type.

31.1 The universal property of the truncations

Definition 31.1.1 Let X be a type. A map $f : X \to Y$ into an k-type Y is said to satisfy the **universal property of the k-truncation of** X if the precomposition map

$$- \circ f : (Y \to Z) \to (X \to Z)$$

is an equivalence for every k-type Z.

Remark 31.1.2 A map $f : X \to Y$ into an k-type Y satisfies the universal property of k-truncation if of for every $g : X \to Z$ the type of extensions

$$\begin{array}{ccc} X & & \\ {\scriptstyle f}\downarrow & \searrow^{g} & \\ Y & \dashrightarrow & Z \end{array}$$

is contractible. Indeed, the type of such extensions is the type

$$\textstyle\sum_{(h:Y\to Z)} h \circ f \sim g,$$

which is equivalent to the fiber of the precomposition map $- \circ f$ at g.

In the following proposition we show that if a map $f : X \to Y$ into a k-type Y satisfies the universal property of the k-truncation of X, then f also satisfies the dependent elimination property.

Proposition 31.1.3 *Suppose the map $f : X \to Y$ into an k-type Y. The following are equivalent:*

(i) *The map f satisfies the universal property of k-truncation.*
(ii) *For any family P of k-types over Y, the precomposition map*

$$- \circ f : \left(\textstyle\prod_{(y:Y)} P(y)\right) \to \left(\textstyle\prod_{(x:X)} P(f(x))\right)$$

*is an equivalence. This property is also called the **dependent universal property** of the k-truncation.*

Proof The fact that (ii) implies (i) is immediate, so we only have to prove the converse.

Suppose P is a family of k-truncated types over Y. Then we have a commuting square

$$\begin{array}{ccc} \left(Y \to \textstyle\sum_{(y:Y)} P(y)\right) & \xrightarrow{-\circ f} & \left(X \to \textstyle\sum_{(y:Y)} P(y)\right) \\ {\scriptstyle \mathsf{pr}_1 \circ -}\downarrow & & \downarrow{\scriptstyle \mathsf{pr}_1 \circ -} \\ \left(Y \to Y\right) & \xrightarrow[-\circ f]{} & \left(X \to Y\right) \end{array}$$

Since the total space $\sum_{(y:Y)} P(y)$ is again k-truncated by Exercise 12.6, it follows by the universal property of the k-truncation that the top map is an equivalence, and by the universal property the bottom map is an equivalence too. It follows from Corollary 22.5.5 that this square is a pullback square, so it induces equivalences on the fibers by Theorem 22.5.3. In particular we have a commuting square

$$\begin{array}{ccc} \left(\prod_{(y:Y)} P(y)\right) & \longrightarrow & \left(\prod_{(x:X)} P(f(x))\right) \\ \downarrow & & \downarrow \\ \mathsf{fib}_{(\mathsf{pr}_1 \circ -)}(\mathsf{id}_Y) & \longrightarrow & \mathsf{fib}_{(\mathsf{pr}_1 \circ -)}(f) \end{array}$$

in which the left and right maps are equivalences by Corollary 13.2.3, and the bottom map is an equivalence as we have just established. Therefore the top map is an equivalence, so we conclude that f satisfies the dependent universal property. □

Just as for pullbacks, pushouts, and the many other types characterized by a universal property, the k-truncation of a type is unique once it exists. We prove this in the following proposition and its corollary.

Proposition 31.1.4 *Consider a commuting triangle*

$$\begin{array}{ccc} & X & \\ {\scriptstyle f}\swarrow & & \searrow{\scriptstyle f'} \\ Y & \xrightarrow[h]{} & Y' \end{array}$$

where Y and Y' are assumed to be k-types. If any two of the following three properties hold, so does the third:

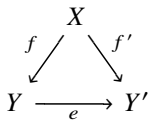
(i) *The map $f : X \to Y$ satisfies the universal property of the k-truncation of X.*
(ii) *The map $f' : X \to X'$ satisfies the universal property of the k-truncation of X.*
(iii) *The map h is an equivalence.*

Proof The claim follows by the 3-for-2 property of equivalences, since we have a commuting triangle

$$\begin{array}{ccc} Z^{Y'} & \xrightarrow{-\circ h} & Z^Y \\ & {\scriptstyle -\circ f'}\searrow \quad \swarrow{\scriptstyle -\circ f} & \\ & Z^X & \end{array}$$

for any k-type Z. □

Corollary 31.1.5 *Consider two maps $f : X \to Y$ and $f' : X \to Y'$ into k-types Y and Y', and suppose that both f and f' satisfy the universal property of the k-truncation of X. Then the type of equivalences $e : Y \simeq Y'$ equipped with a homotopy witnessing that the triangle*



commutes is contractible.

31.2 The construction of the $(k+1)$-truncation as a quotient

Definition 31.2.1 Consider a universe $\mathcal{U}$. We say that $\mathcal{U}$ **has k-truncations** if for every type $X : \mathcal{U}$ there is a map $f : X \to Y$ into a k-type $Y : \mathcal{U}$ that satisfies the universal property of the k-truncation of X.

Remark 31.2.2 Note that the universal property of k-truncations is formulated with respect to all k-types, and not only with respect to the k-types in $\mathcal{U}$.

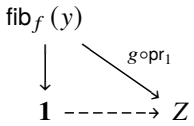
We will use the following proposition to prove the universal property of the $(k+1)$-truncation. In fact, the converse of the following proposition also holds, and we prove it below in Theorem 31.2.6.

Proposition 31.2.3 *Consider a map $f : X \to Y$ into a $(k+1)$-type Y. If f is surjective, and its action on paths*

$$\mathsf{ap}_f : (x = x') \to (f(x) = f(x'))$$

satisfies the universal property of the k-truncation of $x = x'$, then f satisfies the universal property of the $(k+1)$-truncaton of X.

Proof Consider a map $g : X \to Z$ into a $(k+1)$-type Z. Our goal is to show that g extends uniquely along f to a map $h : Y \to Z$. We claim that for any $y : Y$, the type of extensions



is contractible. In other words, on each of the fibers of f, the map g is constant in a unique way. Since f is assumed to be surjective, it follows by Proposition 15.2.3

that it suffices to prove the above extension property for $y \doteq f(x)$, for each $x : X$. In other words, we have to show that the type

$$\textstyle\sum_{(z:Z)} \prod_{(x':X)} (f(x) = f(x')) \to (z = g(x'))$$

is contractible for each $x : X$.

Note that the type $z = g(x')$ is k-truncated, and that the map ap_f is assumed to satisfy the universal property of the k-truncation of $x = x'$. Therefore it is equivalent to show that the type

$$\textstyle\sum_{(z:Z)} \prod_{(x':X)} (x = x') \to (z = g(x'))$$

is contractible. This is immediate by the universal property of the identity type (Theorem 13.3.3), and the fact that $\sum_{(z:Z)} z = g(x')$ is contractible (Corollary 10.4.7).

It follows by Theorem 13.1.2 that the product

$$\textstyle\prod_{(y:Y)} \sum_{(z:Z)} \prod_{(x:X)} (y = f(x)) \to (z = g(x))$$

is contractible. Since Π distributes over Σ by Theorem 13.2.1, we obtain that the type of functions $h : Y \to Z$ equipped with a homotopy $h \circ f \sim g$ is contractible. □

Before we show that any universe has k-truncations for arbitrary k, we prove a truncated version of the type theoretic Yoneda lemma under the assumption that $\mathcal{U}$ has k-truncations for a given k.

Lemma 31.2.4 *Suppose $\mathcal{U}$ is a universe that has k-truncations*

$$|-|_k : X \to \|X\|_k$$

for a given $k \geq -2$, and consider a family P of types over X. We make two claims:

(i) *The evaluation function*

$$\left(\textstyle\prod_{(y:X)} \|x = y\|_k \to \|P(y)\|_k\right) \to \|P(x)\|_k$$

given by $h \mapsto h_x(|\mathsf{refl}_x|_k)$, is an equivalence.

(ii) *If the total space of P is contractible, then the evaluation function*

$$\left(\textstyle\prod_{(y:X)} \|x = y\|_k \simeq \|P(y)\|_k\right) \to \|P(x)\|_k$$

given by $e \mapsto e_x(|\mathsf{refl}_x|_k)$, is an equivalence.

Proof The first claim follows immediately by the universal property of the k-truncation of $x = y$ and the type theoretical Yoneda lemma (Theorem 13.3.3).

To prove the second claim, we first observe that the inclusion of equivalences into all maps induces an embedding that fits in a commuting triangle

$$\begin{array}{ccc} \left(\prod_{(y:X)} \|x = y\|_k \simeq \|P(y)\|_k\right) & \hookrightarrow & \left(\prod_{(y:X)} \|x = y\|_k \to \|P(y)\|_k\right) \\ & {\scriptstyle \mathsf{ev}_{|\mathsf{refl}_x|_k}} \searrow \quad \swarrow {\scriptstyle \mathsf{ev}_{|\mathsf{refl}_x|_k}} & \\ & \|P(x)\|_k. & \end{array}$$

The evaluation map on the right is an equivalence, and we have to show that if the total space $\sum_{(y:X)} P(y)$ is contractible, then the evaluation map on the left is an equivalence. We do this by showing that the top map is an equivalence.

To see this, note that we have a commuting diagram

$$\begin{array}{ccccc} \left(\prod_{(y:X)} (x = y) \simeq P(y)\right) & \hookrightarrow & \left(\prod_{(y:X)} (x = y) \to P(y)\right) & \xrightarrow{\text{ev-refl}} & P(x) \\ \downarrow{\scriptstyle e \mapsto \lambda y.\, \|e_y\|_k} & & \downarrow{\scriptstyle h \mapsto \lambda y.\, \|h_y\|_k} & & \downarrow \\ \left(\prod_{(y:X)} \|x = y\|_k \simeq \|P(y)\|_k\right) & \hookrightarrow & \left(\prod_{(y:X)} \|x = y\|_k \to \|P(y)\|_k\right) & \xrightarrow[\mathsf{ev}_{|\mathsf{refl}_x|_k}]{} & \|P(x)\|_k \end{array}$$

In the top row of this diagram we have a concatenation of equivalences: the first map is an equivalence by the fundamental theorem of identity types, and the second map is an equivalence by Theorem 13.3.3. The second map in the bottom row is an equivalence by the first claim of this lemma. Therefore it follows that the vertical map in the middle satisfies the universal property of the k-truncation. Since the type at the bottom left is k-truncated, we obtain by the universal property of the k-truncation a section of the embedding in the bottom row, which proves the claim. □

Theorem 31.2.5 *Any univalent universe $\mathcal{U}$ that is closed under pushouts has k-truncations, for every truncation level k. We will write*

$$|-|_k : X \to \|X\|_k$$

for the k-truncation of X.

Proof It is easy to see that the terminal projection $X \to \mathbf{1}$ is a (-2)-truncation, for any type X. Thus, any universe has (-2)-truncations.

We will proceed by induction on k. Our inductive hypothesis is that $\mathcal{U}$ has k-truncations, and our goal is to show that $\mathcal{U}$ has $(k+1)$-truncations. The idea of the construction is very similar to the construction of the set quotient by an

equivalence relation. Consider the type-valued relation $R_k : X \to (X \to \mathcal{U})$ given by

$$R_k(x, x') :\equiv \|x = x'\|_k.$$

Analogous to the definition of set quotients, we define

$$\|X\|_{k+1} :\equiv \mathrm{im}(R_k),$$

which comes equipped with a surjective map $q : X \to \|X\|_{k+1}$. Note that the image of $R : X \to (X \to \mathcal{U})$ is (essentially) small by Theorem 26.6.11. To see that $q : X \to \|X\|_{k+1}$ satisfies the universal property of the $(k+1)$-truncation of X, we apply Proposition 31.2.3. Therefore it remains to show that the action on paths

$$\mathsf{ap}_q : (x = x') \to (q(x) = q(x'))$$

satisfies the universal property of the k-truncation of $x = x'$. Since q is the surjective map in the image factorization of R_k, it is equivalent to show that the action on paths

$$\mathsf{ap}_R : (x = x') \to (R_k(x) = R_k(x'))$$

satisfies the universal property of the k-truncation of $x = x'$. Note that we have a commuting triangle

$$\begin{array}{ccc} & (x = x') & \\ {\scriptstyle \mathsf{ap}_{R_k}}\swarrow & & \searrow \\ \Big(R_k(x) = R_k(x')\Big) & \xrightarrow{\simeq} & \Big(\prod_{(y:X)} \|x = y\|_k \simeq \|x' = y\|_k\Big), \end{array}$$

where the bottom map is an equivalence by function extensionality and the univalence axiom. Therefore it suffices to show that the map on the right of this triangle, which is the unique map that sends refl_x to the family of identity equivalences, satisfies the universal property of the k-truncation of $x = x'$.

This map fits in a commuting square

$$\begin{array}{ccc} (x = x') & \xrightarrow{|-|_k} & \|x = x'\|_k \\ \downarrow & & \downarrow{\scriptstyle \|\mathsf{inv}\|_k} \\ \Big(\prod_{(y:X)} \|x = y\|_k \simeq \|x' = y\|_k\Big) & \xrightarrow[\mathsf{ev}_{|\mathsf{refl}_x|_k}]{} & \|x' = x\|_k. \end{array}$$

The map on the right is an equivalence because $\mathsf{inv} : (x = x') \to (x' = x)$ is an equivalence. The bottom map is an equivalence by Lemma 31.2.4. The top map

satisfies the universal property of the k-truncation of $x = x'$, hence so does the map on the left, which completes the proof. □

Theorem 31.2.6 *Consider a map $f : X \to Y$ into a $(k+1)$-truncated type Y. Then the following are equivalent:*

(i) *The map f satisfies the universal property of the $(k+1)$-truncation of X.*
(ii) *The map f is surjective, and for each $x, x' : X$ the map*

$$\mathsf{ap}_f : (x = x') \to (f(x) = f(x'))$$

satisfies the universal property of the k-truncation of $x = x'$.

Proof The fact that (ii) implies (i) was established in Proposition 31.2.3, so it suffices to show that (i) implies (ii).

Suppose first that the map f satisfies the universal property of the $(k+1)$-truncation, and let $x : X$. Recall from Exercise 17.1 that the universe of k-truncated types is itself $(k+1)$-truncated. Therefore it follows that the map $x' \mapsto \|x = x'\|_k$ has a unique extension

$$\begin{array}{ccc} X & & \\ {\scriptstyle f}\downarrow & \searrow^{x' \mapsto \|x = x'\|_k} & \\ Y & \xrightarrow[P]{} & \mathcal{U}_{\leq k}. \end{array}$$

In other words, we obtain a unique family P of k-types over Y equipped with equivalences

$$e_{x'} : P(f(x')) \simeq \|x = x'\|_k$$

indexed by $x' : X$. In particular, P comes equipped with a point $p_0 : P(f(x))$ such that $e_x(p_0) = |\mathsf{refl}_x|_k$. Hence we obtain a family of maps

$$\textstyle\prod_{(y:Y)} (f(x) = y) \to P(y).$$

We claim that this is a family of equivalences. By the fundamental theorem of identity types, Theorem 11.2.2, it suffices to show that the total space

$$\textstyle\sum_{(y:Y)} P(y)$$

is contractible. We have $(f(x), p_0)$ at the center of contraction, so we have to construct a contraction

$$\textstyle\prod_{(y:Y)} \prod_{(p:P(y))} (f(x), p_0) = (y, p).$$

Now we observe that the type $\sum_{(y:Y)} P(y)$ is $(k+1)$-truncated, using the fact that any Σ-type of a family of k-types over a $(k+1)$-type is again $(k+1)$-truncated (Exercise 12.6). It follows that the type $(f(x), p_0) = (y, p)$ is k-truncated for

each $y : Y$ and each $p : P(y)$. Now we use the dependent universal property of the k-truncation of X, which was proven in Proposition 31.1.3, so it suffices to show that

$$\prod_{(x':X)} \prod_{(p:P(f(x')))} (f(x), p_0) = (f(x'), p).$$

Since we have an equivalence $e_{x'} : P(f(x')) \simeq \|x = x'\|_k$ for each $x' : X$, it is equivalent to show that

$$\prod_{(x':X)} \prod_{(p:\|x=x'\|_k)} (f(x), p_0) = (f(x'), e_{x'}^{-1}(p)).$$

Again, we use that the type of paths $(f(x), p_0) = (f(x'), e_{x'}^{-1}(p))$ is a k-type, so we use Proposition 31.1.3 to conclude that it suffices to show that

$$\prod_{(x':X)} \prod_{(p:x=x')} (f(x), p_0) = (f(x'), e_{x'}^{-1}(p)).$$

This is immediate by path induction on $p : x = x'$. This proves the claim that the canonical map

$$h_y : (f(x) = y) \to P(y)$$

is an equivalence for each $y : Y$. Now observe that we have a commuting triangle

$$\begin{array}{ccc} & (x = x') & \\ {\scriptstyle \mathsf{ap}_f}\swarrow & & \searrow{\scriptstyle |-|_k} \\ (f(x) = f(x')) & \xrightarrow[h_{f(x')}]{} & \|x = x'\|_k \end{array}$$

for each $x' : X$. Therefore it follows that $\mathsf{ap}_f : (x = x') \to (f(x) = f(x'))$ satisfies the universal property of the k-truncation of $x = x'$. □

Corollary 31.2.7 *For any $x, y : X$, there is an equivalence*

$$(|x|_{k+1} = |y|_{k+1}) \simeq \|x = y\|_k.$$

31.3 The truncations as recursive higher inductive types

Recall from Theorem 12.4.7 that a map $f : A \to B$ is $(k+1)$-truncated if and only if the action on paths

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

is a k-truncated map, for each $x, y : A$. Moreover, in Exercise 22.2 we established that the fibers of the diagonal map $\delta_f : A \to A \times_B A$ are equivalent to the fibers of the maps ap_f, so it is also the case that f is $(k+1)$-truncated if and only if the diagonal δ_f is k-truncated.

In the following theorem, we add yet another equivalent characterization to

the truncatedness of a map. We will use this theorem in two ways. First, a simple corollary gives a useful characterization of k-truncated types. Second, we will use this theorem to derive an elimination principle of the $(k+1)$-sphere that can be applied to families of k-types

Theorem 31.3.1 *Consider a map $f : A \to B$. Then the following are equivalent:*

(i) *The map f is k-truncated.*
(ii) *The commuting square*

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle \lambda x.\,\mathsf{const}_x}\downarrow & & \downarrow{\scriptstyle \lambda y.\,\mathsf{const}_y} \\ A^{\mathbf{S}^{k+1}} & \xrightarrow[f^{\mathbf{S}^{k+1}}]{} & B^{\mathbf{S}^{k+1}} \end{array}$$

is a pullback square.

Proof We prove the claim by induction on $k \geq -2$. The base case is clear, because the map $A^{\mathbf{S}^{-1}} \to B^{\mathbf{S}^{-1}}$ is a map between contractible types, hence an equivalence. Therefore the square

$$\begin{array}{ccc} A & \longrightarrow & B \\ \downarrow & & \downarrow \\ A^{\mathbf{S}^{-1}} & \longrightarrow & B^{\mathbf{S}^{-1}} \end{array}$$

is a pullback square if and only if $A \to B$ is an equivalence.

For the inductive step, assume that for any map $g : X \to Y$, the map g is k-truncated if and only if the square

$$\begin{array}{ccc} X & \longrightarrow & Y \\ \downarrow & & \downarrow \\ X^{\mathbf{S}^{k+1}} & \longrightarrow & Y^{\mathbf{S}^{k+1}} \end{array}$$

is a pullback square, and consider a map $f : A \to B$. Then f is $(k+1)$-truncated if and only if $\mathsf{ap}_f : (x = y) \to (f(x) = f(y))$ is k-truncated for each $x, y : A$. By the inductive hypothesis this happens if and only if the square

$$\begin{array}{ccc} (x = y) & \longrightarrow & (f(x) = f(y)) \\ \downarrow & & \downarrow \\ (x = y)^{\mathbf{S}^{k+1}} & \longrightarrow & (f(x) = f(y))^{\mathbf{S}^{k+1}} \end{array}$$

is a pullback square, for each $x, y : A$. Now we observe that this is the case if and only if the square on the left in the diagram

$$\begin{array}{ccccc}
\sum_{(x,y:A)} x = y & \longrightarrow & \sum_{(x,y:A)} (f(x) = f(y)) & \longrightarrow & \sum_{(x,y:B)} x = y \\
\downarrow & & \downarrow & & \downarrow \\
\sum_{(x,y:A)} (x = y)^{\mathbf{S}^{k+1}} & \longrightarrow & \sum_{(x,y:A)} (f(x) = f(y))^{\mathbf{S}^{k+1}} & \longrightarrow & \sum_{(x,y:B)} (x = y)^{\mathbf{S}^{k+1}}
\end{array}$$

is a pullback square. The square on the right is a pullback square, so the square on the left is a pullback if and only if the outer rectangle is a pullback. By the universal property of $\mathbf{S}^{k+2}$ it follows that the outer rectangle is a pullback if and only if the square

$$\begin{array}{ccc}
A & \longrightarrow & B \\
\downarrow & & \downarrow \\
A^{\mathbf{S}^{k+2}} & \longrightarrow & B^{\mathbf{S}^{k+2}}
\end{array}$$

is a pullback. □

Theorem 31.3.2 *Consider a type A. Then the following are equivalent:*

(i) *The type A is k-truncated.*
(ii) *The map*

$$\lambda x.\, \mathsf{const}_x : A \to (\mathbf{S}^{k+1} \to A)$$

is an equivalence.

Proof We prove the claim by induction on $k \geq -2$. The base case is clear, because the map $A^{\mathbf{S}^{-1}}$ is contractible.

For the inductive step, assume that any type X is k-truncated if and only if the map

$$\lambda x.\, \mathsf{const}_x : X \to (\mathbf{S}^{k+1} \to X)$$

is an equivalence. Then A is $(k+1)$-truncated if and only if its identity types $x = y$ are k-truncated, for each $x, y : A$. By the inductive hypothesis this happens if and only if

$$(x = y) \to (\mathbf{S}^{k+1} \to (x = y))$$

is a family of equivalences indexed by $x, y : A$. This is a family of equivalences if and only if the induced map on total spaces

$$\left(\sum_{(x,y:A)} x = y\right) \to \left(\sum_{(x,y:A)} (x = y)^{\mathbf{S}^{k+1}}\right)$$

is an equivalence. Note that we have a commuting square

$$\begin{array}{ccc} A & \longrightarrow & A^{\mathbf{S}^{k+2}} \\ \downarrow & & \downarrow \\ \left(\sum_{(x,y:A)} x = y\right) & \longrightarrow & \left(\sum_{(x,y:A)} (x = y)^{\mathbf{S}^{k+1}}\right) \end{array}$$

in which both vertical maps are equivalences. Therefore the top map is an equivalence if and only if the bottom map is an equivalence, which completes the proof. □

Proof Immediate from the fact that A is k-truncated if and only if the map $A \to \mathbf{1}$ is k-truncated. □

Definition 31.3.3 Consider a type X. A **k-truncation** of X consist of a k-type Y, and a map $f : X \to Y$ satisfying the **universal property of k-truncation**, that for every k-type Z the precomposition map

$$- \circ f : (Y \to Z) \to (X \to Z)$$

is an equivalence.

We define $\|X\|_k$ by the 'hubs-and-spokes' method, as a higher inductive type. The idea is to force any map $\mathbf{S}^k X \to \|X\|_k$ to be homotopic to a constant function by including enough points (the hubs) for the values of these constant functions, and enough paths (the spokes) for the homotopies to these constant functions.

Definition 31.3.4 For any type X we define a type $\|X\|_k$ as a higher inductive type, with constructors

$$\begin{aligned} \eta &: X \to \|X\|_k. \\ \mathsf{hub} &: (\mathbf{S}^{k+1} \to \|X\|_k) \to \|X\|_k \\ \mathsf{spoke} &: \textstyle\prod_{(f:\mathbf{S}^{k+1} \to \|X\|_k)} \prod_{(t:\mathbf{S}^{k+1})} f(t) = \mathsf{hub}(f). \end{aligned}$$

Remark 31.3.5 The induction principle for $\|X\|_k$ asserts that for any family P of types over $\|X\|_k$, if we have a dependent function $\alpha : \prod_{(x:X)} P(\eta(x))$ and a dependent function

$$\beta : \textstyle\prod_{(f:\mathbf{S}^{k+1} \to \|X\|_k)} \left(\prod_{(t:\mathbf{S}^{k+1})} P(f(t))\right) \to P(\mathsf{hub}(f))$$

equipped with an identification

$$\gamma(f, g, t) : \mathsf{tr}_P(\mathsf{spoke}(f, t), g(t)) = \beta(f, g),$$

for every $f : \mathbf{S}^{k+1} \to \|X\|_k$, $g : \prod_{(t:\mathbf{S}^{k+1})} P(f(t))$, and every $t : \mathbf{S}^{k+1}$, then we obtain a dependent function

$$h : \prod_{(x:\|X\|_k)} P(\eta(x))$$

equipped with an identification $H(x) : h(\eta(x)) = \alpha(x)$ for any $x : X$.

Proposition 31.3.6 *For any type X, the type $\|X\|_k$ is k-truncated.*

Proof By Theorem 31.3.2 it suffices to show that the map

$$\delta :\equiv \lambda x.\, \mathsf{const}_x : \|X\|_k \to (\mathbf{S}^{k+1} \to \|X\|_k)$$

is an equivalence. Note that the inverse of this map is simply the map

$$\mathsf{hub} : (\mathbf{S}^{k+1} \to \|X\|_k) \to \|X\|_k,$$

which is a section of δ by the homotopy spoke. Therefore it remains to show that

$$\mathsf{hub}(\mathsf{const}_x) = x.$$

for every $x : \|X\|_k$. Note that $\mathsf{spoke}(\mathsf{const}_x, \mathsf{hub}(\mathsf{const}_x))^{-1}$ is such an identification. □

Recall that the $(k+1)$-sphere is k-connected in the following sense.

Lemma 31.3.7 *For any family P of k-types over $\mathbf{S}^{k+1}$, the evaluation map at the base point*

$$\mathsf{ev}_* : \left(\prod_{(t:\mathbf{S}^{k+1})} P(t)\right) \to P(*)$$

is an equivalence.

Theorem 31.3.8 *For any family P of k-types over $\|X\|_k$, the function*

$$- \circ \eta : \left(\prod_{(x:\|X\|_k)} P(x)\right) \to \left(\prod_{(x:X)} P(\eta(x))\right)$$

is an equivalence.

Proof We first show that for any family P of k-types over $\|X\|_k$, the function

$$- \circ \eta : \left(\prod_{(x:\|X\|_k)} P(x)\right) \to \left(\prod_{(x:X)} P(\eta(x))\right)$$

has a section. To see this, we apply the induction principle of $\|X\|_k$. For any function $\alpha : \prod_{(x:X)} P(\eta(x))$ we need to construct a function $h : \prod_{(x:\|X\|_k)} P(x)$ such that $h \circ \eta \sim \alpha$, so it suffices to show that the k-truncatedness of the types in the family P imply the existence of the terms β and η of the induction principle

of $\|X\|_k$. In other words, we need to show that for every $f : \mathbf{S}^{k+1} \to \|X\|_k$ and every $g : \prod_{(t:\mathbf{S}^{k+1})} P(f(t))$ there are

$$\beta(f,g) : P(\mathsf{hub}(f))$$
$$\gamma(f,g) : \prod_{(t:\mathbf{S}^{k+1})} \mathsf{tr}_P(\mathsf{spoke}(f,t), g(t)) = \beta(f,g).$$

Since we have already shown that $\|X\|_k$ is k-truncated, it suffices to show the above for $f :\equiv \mathsf{const}_x$, for any $x : \|X\|_k$. Now the type of g is just the function type $\mathbf{S}^{k+1} \to P(x)$, so by the truncatedness of $P(x)$ it suffices to construct

$$\beta(\mathsf{const}_x, \mathsf{const}_y) : P(\mathsf{hub}(\mathsf{const}_x))$$
$$\gamma(\mathsf{const}_x, \mathsf{const}_y) : \prod_{(t:\mathbf{S}^{k+1})} \mathsf{tr}_P(\mathsf{spoke}(\mathsf{const}_x, t), y) = \beta(\mathsf{const}_x, \mathsf{const}_y)$$

for any $x : X$ and $y : P(x)$. Now we simply define

$$\beta(\mathsf{const}_x, \mathsf{const}_y) :\equiv \mathsf{tr}_P(\mathsf{spoke}(\mathsf{const}_x, *), y).$$

Then it remains to construct an identification

$$\mathsf{tr}_P(\mathsf{spoke}(\mathsf{const}_x, t), y) = \mathsf{tr}_P(\mathsf{spoke}(\mathsf{const}_x, *), y)$$

for any $t : \mathbf{S}^{k+1}$, but this follows at once from Lemma 31.3.7, because the identity types of a k-truncated type is again k-truncated. This completes the proof that the precomposition function

$$- \circ \eta : \left(\prod_{(x:\|X\|_k)} P(x)\right) \to \left(\prod_{(x:X)} P(\eta(x))\right)$$

has a section s for every family P of k-types over $\|X\|_k$.

To show that it is an equivalence, we have to show that s is also a retraction of the precomposition function $- \circ \eta$, i.e., we have to show that

$$s(h \circ \eta) = h$$

for any $h : \prod_{(x:\|X\|_k)} P(x)$. By function extensionality, it is equivalent to show that

$$\prod_{(x:\|X\|_k)} s(h \circ \eta)(x) = h(x).$$

Now we observe that the type $s(h \circ \eta)(x) = h(x)$ is a k-type, and therefore we already know that the function

$$- \circ \eta : \left(\prod_{(x:\|X\|_k)} s(h \circ \eta)(x) = h(x)\right) \to \left(\prod_{(x:X)} s(h \circ \eta)(\eta(x)) = h(\eta(x))\right)$$

has a section. In other words, it suffices to construct a dependent function of type

$$\prod_{(x:X)} s(h \circ \eta)(\eta(x)) = h(\eta(x)).$$

Here we simply use that s is a section $- \circ \eta$, and we are done. □

Corollary 31.3.9 *For any type X, the map $\eta : X \to \|X\|_k$ satisfies the universal property of k-truncation.*

31.4 Theorems not to forget

Theorem 31.4.1 *Consider a type X and a family P of $(k+n)$-truncated types over $\|X\|_k$. Then the precomposition map*

$$- \circ \eta : \left(\prod_{(y:\|X\|_k)} P(y)\right) \to \left(\prod_{(x:X)} P(\eta(x))\right)$$

is $(n-2)$-truncated.

Exercises

31.1 Consider an equivalence relation $R : A \to (A \to \mathsf{Prop})$. Show that the map $|-|_0 \circ \mathsf{inl} : A \to \|A \sqcup^R A\|_0$ satisfies the universal property of the quotient A/R, where $A \sqcup^R A$ is the canonical pushout

$$\begin{array}{ccc} \sum_{(x,y:A)} R(x,y) & \xrightarrow{\pi_2} & A \\ {\scriptstyle \pi_1}\downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ A & \xrightarrow[\mathsf{inl}]{} & A \sqcup^R A. \end{array}$$

31.2 Consider the trivial relation $\mathbf{1} :\equiv \lambda x.\, \lambda y.\, \mathbf{1} : A \to (A \to \mathsf{Prop})$. Show that the set quotient $A/\mathbf{1}$ is a proposition satisfying the universal property of the propositional truncation.

31.3 Show that the type of pointed 2-element sets

$$\sum_{(X:\mathcal{U}_{\mathsf{bool}})} X$$

is contractible.

31.4 Define the type $\mathbb{F}$ of finite sets by

$$\mathbb{F} :\equiv \mathrm{im}(\mathsf{Fin}),$$

where $\mathsf{Fin} : \mathbb{N} \to \mathcal{U}$ is defined in Definition 7.3.2.

(a) Show that $\mathbb{F} \simeq \sum_{(n:\mathbb{N})} \mathcal{U}_{\mathsf{Fin}(n)}$.
(b) Show that $\mathbb{F}$ is closed under Σ and Π.

31.5 (a) A type Y is called **k-separated** if for every type X the map

$$(\|X\|_k \to Y) \to (X \to Y)$$

is an embedding. Show that Y is k-separated if and only if it is $(k+1)$-truncated.

(b) A type Y is called **n-fold k-separated** if for every type X the map

$$(\|X\|_k \to Y) \to (X \to Y)$$

is $(n-2)$-truncated. Show that Y is n-fold k-separated if and only if it is $(k+n)$-truncated.

31.6 Consider a map $f : A \to B$. Show that the square

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle \lambda x.\,\mathsf{const}_x}\downarrow & & \downarrow{\scriptstyle \lambda y.\,\mathsf{const}_y} \\ A^{\mathbf{S}^{k+1}} & \xrightarrow[f^{\mathbf{S}^{k+1}}]{} & B^{\mathbf{S}^{k+1}} \end{array}$$

is a pullback square if and only if its gap map has a section.

31.7 Consider a map $f : X \to Y$ into a k-truncated type Y. Show that the following are equivalent:

(i) For any family P of k-types over Y, the precomposition map

$$- \circ f : \left(\textstyle\prod_{(y:Y)} P(y)\right) \to \left(\textstyle\prod_{(x:X)} P(f(x))\right)$$

is an equivalence.

(ii) For any family P of k-types over Y, the precomposition map

$$- \circ f : \left(\textstyle\prod_{(y:Y)} P(y)\right) \to \left(\textstyle\prod_{(x:X)} P(f(x))\right)$$

has a section.

31.8 Show that for each type X, the map

$$\|X\|_{k+1} \to \mathcal{U}^X$$

given by $y \mapsto \lambda x.\,(y = \eta'(x))$ is an embedding.

31.9 (Descent for truncations) Consider a commuting square

$$\begin{array}{ccc} A & \xrightarrow{i} & X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[j]{} & Y \end{array}$$

in which both X and Y are assumed to be n-truncated. Show that the following are equivalent:

(i) The square is a pullback square.

(ii) The induced square

$$\begin{array}{ccc} \|A\|_n & \dashrightarrow & X \\ {\scriptstyle \|f\|_n}\downarrow & & \downarrow{\scriptstyle g} \\ \|B\|_n & \dashrightarrow & Y \end{array}$$

is a pullback square.

Conclude that in each of these equivalent cases, the square on the left is also a pullback square.

32 Connected types and maps

In this section we introduce the concept of k-connected types and maps. We define k-connected types to be types with contractible k-truncation, and a k-connected map is just a map of which the fibers are k-connected. The idea is that a type is k-connected if and only if its homotopy groups $\pi_i(X)$ are trivial for all $i \leq k$.

One of the main theorems in this section is a characterization of k-connected maps in terms of their action on homotopy groups: A map $f : X \to Y$ is k-connected if and only if it induces isomorphisms

$$\pi_i(f, x) : \pi_i(X, x) \to \pi_i(Y, f(x))$$

of homotopy groups, for each $i \leq k$ and each $x : X$, and a *surjective* group homomorphism

$$\pi_{k+1}(f, x) : \pi_{k+1}(X, x) \to \pi_{k+1}(Y, f(x))$$

on the $(k+1)$-st homotopy group, for each $x : X$. If one drops the condition that f induces a surjective group homomorphism on the $(k+1)$-st homotopy group, then the map is only a k-equivalence, i.e., a map of which $\|f\|_k$ is an equivalence. We see from the above characterization that any k-connected map is a k-equivalence, and also that any $(k+1)$-equivalence is a k-connected map. Nevertheless, the difference between the classes of k-equivalences and k-connected maps is somewhat subtle.

We will study k-equivalences and k-connected maps synchronously, because understanding the subtle differences between the results about either of them will increase the understanding of both classes of maps. For instance, we will show that the k-connected maps enjoy a dependent elimination property, while the k-equivalences only satisfy a non-dependent elimination property. We will

see that the k-equivalences satisfy the 3-for-2 property, while one of the cases of the 3-for-2 property fails for k-connected maps.

The k-connected maps can be characterized as the class of maps that is left orthogonal to the class of k-truncated maps, where a map $f : A \to B$ is said to be left orthogonal to a map $g : X \to Y$ if the type of diagonal fillers of any commuting square of the form

$$\begin{array}{ccc} A & \longrightarrow & X \\ {\scriptstyle f}\downarrow & \nearrow & \downarrow{\scriptstyle g} \\ B & \longrightarrow & Y \end{array}$$

is contractible. Similarly, the class of k-equivalences is the class of maps that is left orthogonal to any map between k-truncated types. However, this result is not entirely sharp, because there are more maps that the k-equivalences are left orthogonal to. It turns out that a map is a k-equivalence if and only if it is left orthogonal to any map $g : X \to Y$ for which the naturality square

$$\begin{array}{ccc} X & \xrightarrow{g} & Y \\ {\scriptstyle \eta}\downarrow & & \downarrow \\ \|X\|_k & \xrightarrow[\|g\|_k]{} & \|Y\|_k \end{array}$$

is a pullback square. Such maps are called k-étale, and they induce isomorphisms

$$\pi_i(g, x) : \pi_i(X, x) \to \pi_i(Y, g(x))$$

on homotopy groups for $i > k$.

In the final part of this section we will use the results about k-equivalences to show that the n-sphere is $(n-1)$-connected, for each $n : \mathbb{N}$, and that the join $A * B$ is $(k + l + 2)$-connected if A is k-connected and B is l-connected.

32.1 Connected types

Definition 32.1.1 A type X is said to be **k-connected** if its k-truncation $\|X\|_k$ is contractible. We define

$$\mathsf{is\text{-}conn}_k(X) :\equiv \mathsf{is\text{-}contr}\|X\|_k.$$

Remark 32.1.2 Since the (-2)-truncation of any type is just $\mathbf{1}$, it follows that every type is (-2)-connected. Furthermore, since any proposition is contractible as soon as it comes equipped with an element, it follows that any type is (-1)-connected as soon as it is inhabited.

In Theorem 32.1.4 below, we will see that a type X is 0-connected if and only

if it is inhabited and every two points are connected by an unspecified path. In this sense 0-connected types are also called **path connected**, or just **connected**. Thus, it is immediate that the circle is an example of a connected type.

Similarly, in the case where $k \doteq 0$ the theorem states that a type X is 1-conneced if and only if it is inhabited and for every $x, y : X$ the identity type $x = y$ is path connected. In other words, a type is **simply connected** if it is 1-connected! The 2-sphere is an example of a simply connected type. This fact is shown in Corollary 32.4.4 below, where we will show more generally that the n-sphere is $(n-1)$-connected, for each $n : \mathbb{N}$.

Lemma 32.1.3 *If a type is $(k+1)$-connected, then it is also k-connected.*

Proof This follows from the fact that $\|\|X\|_{k+1}\|_k \simeq \|X\|_k$. Indeed, if $\|X\|_{k+1}$ is contractible, then its k-truncation is also contractible, so it follows that $\|X\|_k$ is contractible. □

For the following theorem, recall that a type X is said to be inhabited if it comes equipped with an element of type $\|X\|_{-1}$.

Theorem 32.1.4 *Consider a type X. Then the following are equivalent:*

(i) *The type X is $(k+1)$-connected.*
(ii) *The type X is inhabited, and the type $x = y$ is k-connected for each $x, y : X$.*

Proof Suppose first that X is $(k+1)$-connected. It is immediate that X is inhabited in this case. Moreover, since we have equivalences

$$(\eta(x) = \eta(y)) \simeq \|x = y\|_k$$

for each $x, y : X$, it follows from the assumption that $\|X\|_{k+1}$ is contractible that the type $\|x = y\|_k$ is equivalent to a contractible type. This proves that (i) implies (ii).

To see that (ii) implies (i), suppose that X is inhabited and that its identity types are k-connected. Our goal is to construct an element of type

$$\mathsf{is\text{-}contr}\|X\|_{k+1},$$

which is a proposition, so we may eliminate the assumption that X is inhabited and assume to have $x : X$. Now we simply take $\eta(x)$ for the center of contraction of $\|X\|_{k+1}$. To construct the contraction, note that by the dependent universal property of $(k+1)$-truncation we have an equivalence

$$\left(\prod_{(y:\|X\|_{k+1})} \eta(x) = y\right) \simeq \left(\prod_{(y:X)} \eta(x) = \eta(y)\right).$$

Therefore it suffices to construct an identification $\eta(x) = \eta(y)$ for every $y : X$.

However, this type is contractible, since it is equivalent to the contractible type $\|x = y\|_k$. This completes the proof of (ii) implies (i). □

In the case where $k \geq -1$ we can improve Theorem 32.1.4 and characterize a high degree of connectedness entirely in terms of the triviality of homotopy groups. This is what connectedness is all about.

Theorem 32.1.5 *Consider a type X, and suppose that $k \geq 0$. Then the following are equivalent:*

(i) *The type X is k-connected.*
(ii) *The type X is connected, and for every $x : X$ the loop space*

$$\Omega(X, x)$$

is $(k-1)$-connected.
(iii) *For each $i \leq k$ and each $x : X$, the i-th homotopy group $\pi_i(X, x)$ is trivial.*

Proof If X is k-connected for $k \geq 0$, then it is certainly connected, and $\Omega(X, x)$ is $(k-1)$-connected by Theorem 32.1.4. Thus, the fact that (i) implies (ii) is immediate.

To see that (ii) implies (i), note that if X is connected and its loop spaces are $(k-1)$-connected, then all its identity types are $(k-1)$-connected, since we have

$$\begin{aligned}\textstyle\prod_{(x,y:X)} \mathsf{is\text{-}contr}(\|x = y\|_{k-1}) &\simeq \textstyle\prod_{(x,y:X)} \|x = y\|_{-1} \to \mathsf{is\text{-}contr}(\|x = y\|_{k-1}) \\ &\simeq \textstyle\prod_{(x,y:X)} (x = y) \to \mathsf{is\text{-}contr}(\|x = y\|_{k-1}) \\ &\simeq \textstyle\prod_{(x:X)} \mathsf{is\text{-}contr}(\|x = x\|_{k-1}).\end{aligned}$$

In the first step of this calculation we use that X is connected, so $\|x = y\|_{-1}$ is contractible; then we use that $\mathsf{is\text{-}contr}(\|x = y\|_{k-1})$ is a proposition; and finally we use the universal property of identity types to arrive at our assumption that the loop spaces of X are $(k-1)$-connected. Since we have shown that the identity types are $(k-1)$-connected, it follows by Theorem 32.1.4 that X is k-connected, which concludes the proof that (ii) implies (i).

It is easy to see by induction on $k \geq 0$ that (ii) holds if and only if (iii) holds, since we have

$$\pi_{i+1}(X, x) = \pi_i(\Omega(X, x)).$$ □

Remark 32.1.6 If X is assumed to be a pointed type in Theorem 32.1.5, then conditions (ii) and (iii) only have to be checked at the base point.

32.2 k-Equivalences and k-connected maps

We now study two classes of maps that differ only slightly: the k-equivalences and the k-connected maps.

Definition 32.2.1

(i) A map $f : X \to Y$ is said to be **k-connected** if its fibers are k-connected. We will write

$$\mathsf{is\text{-}conn}_k(f) :\equiv \prod_{(y:Y)} \mathsf{is\text{-}conn}_k(\mathsf{fib}_f(y)).$$

(ii) A map $f : X \to Y$ is said to be a **k-equivalence** if

$$\|f\|_k : \|X\|_k \to \|Y\|_k$$

is an equivalence. We will write

$$\mathsf{is\text{-}equiv}_k(f) :\equiv \mathsf{is\text{-}equiv}(\|f\|_k).$$

Example 32.2.2 Any equivalence is a k-connected map, as well as a k-equivalence. Moreover, for any k-connected type X the map $\mathsf{const}_\star : X \to \mathbf{1}$ is k-connected. It is also immediate that *any* map between k-connected types is a k-equivalence.

Example 32.2.3 A (-1)-connected map is a map $f : X \to Y$ for which the propositionally truncated fibers

$$\|\mathsf{fib}_f(y)\|_{-1}$$

are contractible. Since propositions are contractible as soon as they are inhabited, we see that a map is (-1)-connected if and only if it is surjective.

A (-1)-equivalence, on the other hand, is just a map $f : X \to Y$ that induces an equivalence $\|X\|_{-1} \simeq \|Y\|_{-1}$. The map $\mathsf{const}_{\mathsf{true}} : \mathbf{1} \to \mathsf{bool}$ is an example of such a map, showing that (-1)-equivalences don't need to be surjective.

However, it is the case that every surjective map $f : X \to Y$ is in fact (-1)-equivalence. To see this, we need to show that

$$\|Y\|_{-1} \to \|X\|_{-1}.$$

Such a map is constructed by the universal property of (-1)-truncation. Thus, it suffices to construct a function $Y \to \|X\|_{-1}$. Since we have assumed that f is surjective, we have for every $y : Y$ the element

$$s(y) : \|\mathsf{fib}_f(y)\|_{-1}.$$

Thus, we define a function $Y \to \|X\|_{-1}$ by

$$y \mapsto \|\mathsf{pr}_1\|_{-1}(s(y)).$$

This concludes the proof that f is a (-1)-equivalence, since we have shown that $\|X\|_{-1} \leftrightarrow \|Y\|_{-1}$.

Remark 32.2.4 An immediate difference between the classes of k-equivalences and k-connected maps is that the k-connected maps are stable under base change, while the k-equivalences are not. By this, we mean that for any pullback square

$$\begin{array}{ccc} E' & \xrightarrow{g} & E \\ {\scriptstyle p'}\downarrow & & \downarrow{\scriptstyle p} \\ B' & \xrightarrow[f]{} & B, \end{array}$$

if the map p is k-connected, then the map p' is also k-connected. In such a pullback diagram, the map p' is sometimes called the **base change** of p along f. By Theorem 22.5.3 we have an equivalence

$$\mathsf{fib}_{p'}(b') \simeq \mathsf{fib}_p(f(b'))$$

for any $b' : B'$, so it is indeed the case that if the fibers of p are k-connected, then so are the fibers of p'.

An example showing that the k-equivalences are not stable under base change is given by the pullback square

$$\begin{array}{ccc} \Omega(\mathbf{S}^{k+1}) & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow \\ \mathbf{1} & \longrightarrow & \mathbf{S}^{k+1} \end{array}$$

We will show in Corollary 32.4.4 that the $(k+1)$-sphere is k-connected, so the map $\mathbf{1} \to \mathbf{S}^{k+1}$ is a k-equivalence. However, its loop space is only $(k-1)$-connected, and indeed we will show in **??** that $\pi_{k+1}(\mathbf{S}^{k+1}) = \mathbb{Z}$ for $k \geq 0$, showing that $\Omega(\mathbf{S}^{k+1})$ is *not* k-connected. Thus, the map $\Omega(\mathbf{S}^{k+1}) \to \mathbf{1}$ is not a k-equivalence.

Elimination properties

We will show that a map $f : X \to Y$ is a k-equivalence if and only if the precomposition function

$$- \circ f : (Y \to Z) \to (X \to Z)$$

is an equivalence for every k-type Z. On the other hand, we will show that f is k-connected if and only if the precomposition function

$$- \circ f : \left(\prod_{(y:Y)} P(y)\right) \to \left(\prod_{(x:X)} P(f(x))\right)$$

is an equivalence for every family P of k-types over Y. In other words, the k-connected maps satisfy a *dependent* unique elimination property, while the k-equivalences only satisfy a *non-dependent* unique elimination property.

Theorem 32.2.5 *Consider a function $f : X \to Y$. Then the following are equivalent*

(i) *The map f is a k-equivalence.*
(ii) *For every k-type Z, the precomposition function*

$$- \circ f : (Y \to Z) \to (X \to Z)$$

is an equivalence.

Theorem 32.2.6 *Let $f : X \to Y$ be a map. The following are equivalent:*

(i) *The map f is k-connected.*
(ii) *For every family P of k-truncated types over Y, the precomposition map*

$$- \circ f : \left(\prod_{(y:Y)} P(y)\right) \to \left(\prod_{(x:X)} P(f(x))\right)$$

is an equivalence.

Proof Suppose f is k-connected and let P be a family of k-types over Y. Now we may consider the following commuting diagram

$$\begin{array}{ccccc}
\prod_{(y:Y)} P(y) & & \xrightarrow{\quad - \circ f \quad} & & \prod_{(x:X)} P(f(x)) \\
& \searrow\!\swarrow & & \nwarrow\!\nearrow & \\
\prod_{(y:Y)} \|\mathsf{fib}_f(y)\|_k \to P(y) & & & & \prod_{(x:X)} \prod_{(y:Y)} \prod_{(p:f(x)=y)} P(y) \\
& \searrow & & \nearrow & \\
& \prod_{(y:Y)} \mathsf{fib}_f(y) \to P(y) & \longrightarrow & \prod_{(y:Y)} \prod_{(x:X)} \prod_{(p:f(x)=y)} P(y) &
\end{array}$$

which commutes by refl-htpy. In this diagram, the five maps going around counter clockwise are all equivalences for obvious reasons, so it follows that the top map is an equivalence.

Now suppose that f satisfies the dependent elimination property stated in (ii). In order to construct a center of contraction of $\|\mathsf{fib}_f(y)\|_k$ for every $y : Y$, we use the dependent elimination property with respect to the family P given by $P(y) :\equiv \|\mathsf{fib}_f(y)\|_k$. □

Corollary 32.2.7 *For any type X, the unit $\eta : X \to \|X\|_k$ of the k-truncation is a k-connected map.*

The inclusions

We will prove the following implications

$$\mathsf{is\text{-}equiv}_{k+1}(f) \xrightarrow{\text{Proposition 32.2.9}} \mathsf{is\text{-}conn}_k(f) \xrightarrow{\text{Proposition 32.2.8}} \mathsf{is\text{-}equiv}_k(f)$$

showing that the class of k-connected maps is contained in the class of k-equivalences, and that the class of $(k+1)$-equivalences is contained in the class of k-connected maps. Neither of these implications reverses.

Proposition 32.2.8 *Any k-connected map is a k-equivalence.*

Proposition 32.2.9 *Any $(k+1)$-equivalence is k-connected.*

Proof Consider a $(k+1)$-equivalence $f : X \to Y$. Recall that the map $\|f\|_{k+1}$ comes equipped with a homotopy $H : \|f\|_{k+1} \circ \eta \sim \eta \circ f$ witnessing that the square

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle\eta}\downarrow & & \downarrow{\scriptstyle\eta} \\ \|X\|_{k+1} & \xrightarrow[\|f\|_{k+1}]{} & \|Y\|_{k+1} \end{array}$$

commutes. We be using this homotopy, and we will use Theorem 32.2.6 to show that f is k-connected. Thus, our goal is to show that

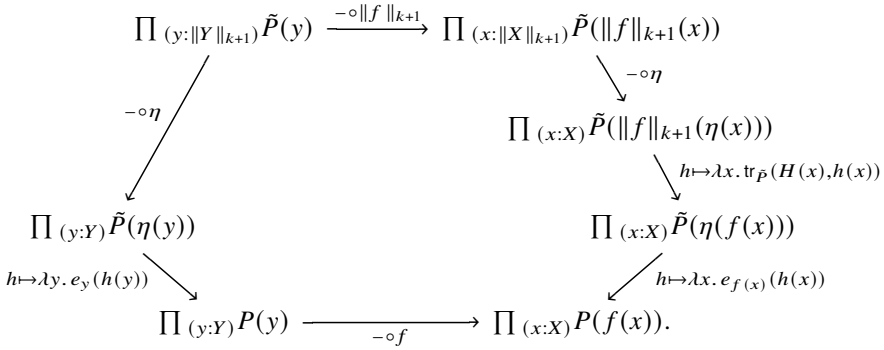
$$- \circ f : \left(\textstyle\prod_{(y:Y)} P(y)\right) \to \left(\textstyle\prod_{(x:X)} P(f(x))\right)$$

is an equivalence for any family P of k-types over Y.

Note that any family P of k-types over Y extends to a family $\tilde{P}$ of k-types over $\|Y\|_{k+1}$, since any univalent universe of k-types that contains P is itself a $(k+1)$-type by Exercise 17.1. The extended family $\tilde{P}$ of k-types over $\|Y\|_{k+1}$ comes equipped with a family of equivalences

$$e : \textstyle\prod_{(y:Y)} \tilde{P}(\eta(y)) \simeq P(y).$$

Now consider the commuting diagram



This diagram commutes by the homotopy

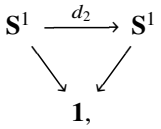
$$\lambda h.\,\mathsf{eq\text{-}htpy}(\lambda x.\,\mathsf{ap}_{e(f(x))}(\mathsf{apd}_h(H(x)))^{-1}).$$

In this diagrams all the maps pointing downwards are equivalences for obvious reasons: the two maps $-\circ\eta$ are equivalences since $\tilde{P}$ is a family of k-types, and the remaining three maps pointing downwards are all postcomposing with an equivalence. The top map is an equivalence since $\|f\|_{k+1}$ is assumed to be an equivalence. Thus we conclude that the bottom map $-\circ f$ is an equivalence. □

The 3-for-2 property

An important distinction between the class of k-equivalences and the class of k-connected maps is that the k-equivalences satisfy the 3-for-2 property, while the k-connected maps do not.

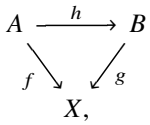
Remark 32.2.10 It is not hard to see that the k-connected maps don't satisfy the 3-for-2 property. For example, consider the following commuting triangle



where $d_2 : \mathbf{S}^1 \to \mathbf{S}^1$ is the degree 2 map. Since the circle is a 0-connected type, it follows that the maps $\mathbf{S}^1 \to \mathbf{1}$ are 0-connected. However, the fiber of d_2 at the base point is equivalent to the booleans, which is a non-contractible set so it is certainly not 0-connected.

Lemma 32.2.11 *The k-equivalences satisfy the 3-for-2 property, i.e., for any*

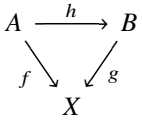
commuting triangle



if any two of the three maps are k-equivalences, then so is the third.

Proof This follows immediately from the fact that equivalences satisfy the 3-for-2 property. □

Proposition 32.2.12 *Consider a commuting triangle*



with $H : f \sim g \circ h$. The following three statements hold:

(i) *If f and h are k-connected, then g is k-connected.*
(ii) *If g and h are k-connected, then f is k-connected.*
(iii) *If f and g are k-connected, then h is a k-equivalence.*

Proof The first two statements combined assert that if h is k-connected, then f is k-connected if and only if g is k-connected. To see that this equivalence holds, consider for any family P of k-truncated types over X the commuting square

$$\begin{array}{ccc} \prod_{(x:X)} P(x) & \xrightarrow{-\circ g} & \prod_{(b:B)} P(g(b)) \\ {\scriptstyle -\circ f}\downarrow & & \downarrow{\scriptstyle -\circ h} \\ \prod_{(a:A)} P(f(a)) & \xrightarrow[\lambda s.\, \lambda a.\, \mathrm{tr}_P(H(a), s(a))]{} & \prod_{(a:A)} P(g(h(a))) \end{array}$$

In this square, the bottom map is given by postcomposing with the family of equivalences $\mathrm{tr}_P(H(a))$ indexed by $a : A$, so it is an equivalence. The map on the right is an equivalence by Theorem 32.2.6, using the assumption that h is a k-connected map. The square commutes by the homotopy

$$\lambda s.\, \mathsf{eq\text{-}htpy}\big(\lambda a.\, \mathsf{apd}_s(H(a))\big).$$

Therefore it follows that the precomposition map $- \circ f$ is an equivalence if and only if the precomposition map $- \circ g$ is. By Theorem 32.2.6 we conclude that f is connected if and only if g is. This proves statements (i) and (ii).

Statement (iii) follows from the facts that any k-connected map is a k-equivalence by **??** and that the k-equivalences satisfy the 3-for-2 property **??**. □

The action on homotopy groups

Theorem 32.2.13 *Consider a map $f : X \to Y$, and suppose that $k \geq -1$. The following are equivalent:*

(i) *The map f is a k-equivalence.*
(ii) *The map f is a (-1)-equivalence, and for every $0 \leq i \leq k$ and every $x : X$, the induced group homomorphism*

$$\pi_i(f, x) : \pi_i(X, x) \to \pi_i(Y, f(x))$$

is an isomorphism.

Definition 32.2.14 A map $f : X \to Y$ is said to be a **weak equivalence** if it is a 0-equivalence, and it induces an isomorphism

$$\pi_i(f, x) : \pi_i(X, x) \cong \pi_i(Y, f(x))$$

on homotopy groups, for every $x : X$ and every $i \geq 1$.

The following corollary is an instance of Whitehead's principle, which asserts that a map between any two spaces is a homotopy equivalence if and only if it is a weak equivalence. Thus, by the following corollary, Whitehead's principle holds for k-types.

Corollary 32.2.15 *Consider two k-types X and Y, and consider a map $f : X \to Y$ between them. Then the following are equivalent:*

(i) *The map f is an equivalence.*
(ii) *The map f is a weak equivalence.*

Theorem 32.2.16 *Consider a map $f : X \to Y$. The following are equivalent:*

(i) *The map f is $(k+1)$-connected.*
(ii) *The map f is surjective, and for each $x, x' : X$ the action on paths*

$$\mathsf{ap}_f : (x = x') \to (f(x) = f(x'))$$

is k-connected.

Theorem 32.2.17 *Consider a surjective map $f : X \to Y$. The following are equivalent:*

(i) *The map f is k-connected.*

(ii) *The induced maps on loop spaces*

$$\Omega(f,x) : \Omega(X,x) \to \Omega(Y,f(x))$$

is $(k-1)$*-connected for every* $x : X$.

(iii) *The induced maps on homotopy groups*

$$\pi_i(f,x) : \pi_i(X,x) \to \pi_i(Y,f(x))$$

are isomorphisms for $0 \leq i \leq k$*, and it is surjective for* $i = k+1$.

Remark 32.2.18 If $f : X \to Y$ is a pointed map between connected types, then conditions (ii) and (iii) in **??** only have to be checked at the base point.

32.3 Orthogonality

The idea of orthogonality is that a map $f : A \to B$ is left orthogonal to a map $g : X \to Y$ if for every commuting square of the form

$$\begin{array}{ccc} A & \xrightarrow{h} & X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[i]{} & Y, \end{array}$$

with $H : (i \circ f) \sim (g \circ h)$, the type of diagonal fillers is contractible. The type of diagonal fillers is the type of maps $j : B \to X$ equipped with homotopies

$$K : j \circ f \sim h$$

$$L : g \circ j \sim i$$

and a homotopy M witnessing that the triangle

$$\begin{array}{ccc} g \circ j \circ f & \xrightarrow{g \cdot K} & h \circ g \\ & {\scriptstyle L \cdot f} \searrow \quad \nearrow {\scriptstyle H} & \\ & i \circ f & \end{array}$$

commutes. A slicker way to express this condition is to assert that the map

$$(B \to X) \to \textstyle\sum_{(h:A\to X)} \sum_{(i:B\to Y)} i \circ f \sim g \circ h$$

given by $j \mapsto (j \circ f, g \circ j, \mathsf{refl\text{-}htpy})$ is an equivalence. Indeed, the type of triples (h,i,H) in the codomain is the type of commuting squares with respect to which we stated the orthogonality condition. Now we may even recognize the above map as a gap map of a commuting square, and we arrive at our actual definition of orthogonality.

Definition 32.3.1 A map $f : A \to B$ is said to be **left orthogonal** to a map $g : X \to Y$, or equivalently the map g is said to be **right orthogonal** to f, if the commuting square

$$\begin{array}{ccc} X^B & \xrightarrow{-\circ f} & X^A \\ {\scriptstyle g\circ -}\downarrow & & \downarrow{\scriptstyle g\circ -} \\ Y^B & \xrightarrow[-\circ f]{} & Y^A \end{array}$$

is a pullback square.

Theorem 32.3.2 *Let $f : A \to B$ be a map. The following are equivalent:*

(i) *The map f is k-connected.*
(ii) *The map f is left orthogonal to every k-truncated map. is a pullback square.*

Theorem 32.3.3 *Let $f : A \to B$ be a map. The following are equivalent:*

(i) *The map f is a k-equivalence.*
(ii) *The map f is left orthogonal to every map between k-truncated types.*
(iii) *The map f is left orthogonal to every map $g : X \to Y$ for which the naturality square*

$$\begin{array}{ccc} X & \xrightarrow{g} & Y \\ {\scriptstyle \eta}\downarrow & & \downarrow{\scriptstyle \eta} \\ \|X\|_k & \xrightarrow[\|g\|_k]{} & \|Y\|_k \end{array}$$

*is a pullback square. Such maps are called k-**étale**.*

32.4 The connectedness of suspensions

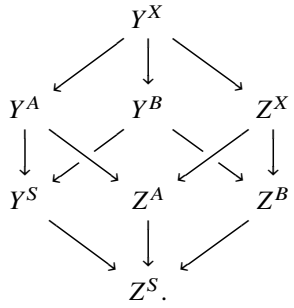
We will use connected maps to prove the connectedness of suspensions.

Proposition 32.4.1 *Consider a pushout square*

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

If the map $f : S \to A$ is k-connected, then so is the map $j : B \to X$.

Proof We claim that the map $j : B \to X$ is left orthogonal to any k-truncated map $p : Y \to Z$, which is equivalent to the property that j is k-connected. To see that j is left orthogonal to p, consider the commuting cube



In this cube, the front left square is a pullback square because the map $f : S \to A$ is assumed to be k-connected, and therefore it is left orthogonal to the k-truncated map p. The back left and front right squares are pullback squares by the pullback property of pushouts. Therefore it follows that the back right square is a pullback square. This shows that j is left orthogonal to p. □

Lemma 32.4.2 *A pointed type X is $(k+1)$-connected if and only if the point inclusion*

$$\mathbf{1} \to X$$

is a k-connected map.

Proof Since X is assumed to have a base point $x_0 : X$, it follows that X is $(k+1)$-connected if and only if its identity types $(x = y)$ are k-connected. Now the claim follows from the fact that there is an equivalence

$$\mathsf{fib}_{\mathsf{const}_{x_0}}(y) \simeq (x_0 = y).$$ □

Theorem 32.4.3 *If X is an k-connected type, then its suspension ΣX is $(k+1)$-connected.*

Proof The type X is k-connected if and only if the map $\mathsf{const}_\star : X \to \mathbf{1}$ is a k-connected map. Recall that the suspension of X is a pushout

$$\begin{array}{ccc} X & \xrightarrow{\mathsf{const}_\star} & \mathbf{1} \\ {\scriptstyle \mathsf{const}_\star}\downarrow & & \downarrow{\scriptstyle \mathsf{S}} \\ \mathbf{1} & \xrightarrow[\mathsf{N}]{} & \Sigma X. \end{array}$$

Therefore we see by Proposition 32.4.1 that the point inclusions $\mathsf{N}, \mathsf{S} : \mathbf{1} \to \Sigma X$

are both k-connected maps. By Lemma 32.4.2 it follows that ΣX is a $(k+1)$-connected type. □

Corollary 32.4.4 *The n-sphere is $(n-1)$-connected.*

Proof The 0-sphere is (-1)-connected, since it contains a point. Thus the claim follows by induction on $n : \mathbb{N}$, using Theorem 32.4.3. □

32.5 The join connectivity theorem

Theorem 32.5.1 *If X is k-connected and Y is l-connected, then their join $X * Y$ is $(k+l+2)$-connected.*

Theorem 32.5.2 *Consider a pullback square*

$$\begin{array}{ccc} C & \longrightarrow & B \\ \downarrow & & \downarrow \\ A & \longrightarrow & X. \end{array}$$

If the maps $A \to X$ and $B \to X$ are k- and l-connected, respectively, then the map $A \sqcup^C B \to X$ is $(k+l+2)$-connected.

Theorem 32.5.3 *The connected maps contain the equivalences, are closed under coproducts, pushouts, retracts, and transfinite compositions.*

32.6 Universal covers

Theorem 32.6.1 *Consider a type X with base point x_0, and consider a family Y of sets over X equipped with a base point $y_0 : Y(x_0)$. Then the following are equivalent:*

(i) *The canonical map*

$$\|x_0 = x\|_0 \to Y(x)$$

is an equivalence for each $x : X$.

(ii) *The total space*

$$\textstyle\sum_{(x:X)} Y(x)$$

is simply connected.

If either of these equivalent conditions holds, then we call Y the ***universal cover*** *of X at x_0, and moreover we obtain an equivalence*

$$\pi_1(X, x_0) \simeq Y(x_0).$$

In fact, we will prove a more general version of the above theorem.

Theorem 32.6.2 *Consider a type X with base point x_0, and consider a family Y of k-types over X equipped with a base point $y_0 : Y(x_0)$. Then the following are equivalent:*

(i) *The canonical map*
$$\|x_0 = x\|_k \to Y(x)$$
is an equivalence for each $x : X$.

(ii) *The total space*
$$\textstyle\sum_{(x:X)} Y(x)$$
is $(k+1)$-connected.

If either of these equivalent conditions holds, then we call Y the ***universal k-cover*** *of X at x_0.*

Proof Since Y is assumed to be a family of k-types over X, we find a unique family $\tilde{Y}$ of k-types over $\|(\|_{k+1}X)$ such that the square

$$\begin{array}{ccc} \sum_{(x:X)} Y(x) & \dashrightarrow & \sum_{(x:\|X\|_{k+1})} \tilde{Y}(x) \\ {\scriptstyle \mathsf{pr}_1}\downarrow & & \downarrow{\scriptstyle \mathsf{pr}_1} \\ X & \xrightarrow[\eta]{} & \|X\|_{k+1} \end{array}$$

is a pullback square. □

Exercises

32.1 Show that every type is equivalent to a disjoint union of connected components, i.e., show that for every type X there is a family of connected types B_i by a set I, with an equivalence
$$X \simeq \textstyle\sum_{(i:I)} B_i.$$

32.2 Let $f : A \to_* B$ be a pointed map between pointed n-connected types, for $n \geq -1$. Show that the following are equivalent:
(i) f is an equivalence.
(ii) $\Omega^{n+1}(f)$ is an equivalence.

32.3 Show that if
$$\begin{array}{ccc} A & \longrightarrow & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ X & \longrightarrow & Y \end{array}$$

is k**-cocartesian** in the sense that the cogap map is k-connected, then the map $\mathsf{cofib}(f) \to \mathsf{cofib}(g)$ is k-connected.

32.4 Show that if $f : X \to Y$ is a k-connected map, then so is

$$\|f\|_l : \|X\|_l \to \|Y\|_l$$

for any $l \geq -2$.

32.5 Consider a commuting square

$$\begin{array}{ccc} A & \longrightarrow & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ X & \longrightarrow & Y \end{array}$$

(a) Show that if the square is k-cartesian and g is k-connected, then so is f.

(b) Show that if f is k-connected and g is $(k+1)$-connected, then the square is k-cartesian.

32.6 (a) Show that any sequential colimit of k-connected types is again k-connected.

(b) Show that if every map in a type sequence

$$A_0 \longrightarrow A_1 \longrightarrow A_2 \longrightarrow \cdots$$

is k-connected, then so is the transfinite composition $A_0 \to A_\infty$.

32.7 Recall that a commuting square is called k-cartesian, if its gap map is k-connected. Show that $(k+1)$-truncation preserves l-cartesian squares for any $l \leq k$, i.e., show that for any $l \leq k$, if a square

$$\begin{array}{ccc} C & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & X. \end{array}$$

is l-cartesian, then the square

$$\begin{array}{ccc} \|C\|_{k+1} & \xrightarrow{\|q\|_{k+1}} & \|B\|_{k+1} \\ {\scriptstyle \|p\|_{k+1}}\downarrow & & \downarrow{\scriptstyle \|g\|_{k+1}} \\ \|A\|_{k+1} & \xrightarrow[\|f\|_{k+1}]{} & \|X\|_{k+1} \end{array}$$

is l-cartesian.

32.8 Generalize Remark 32.2.10 to show that for every $k \geq -1$, the k-connected maps do not satisfy the 3-for-2 property.

32.9 Consider a commuting square

$$\begin{array}{ccc} A & \longrightarrow & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ X & \longrightarrow & Y \end{array}$$

Show that the following are equivalent:

(i) The map $A \to X \times_Y B$ is n-connected. In this case the square is called **n-cartesian**.

(ii) For each $x : X$ the map

$$\mathsf{fib}_f(x) \to \mathsf{fib}_g(f(x))$$

is n-connected.

32.10 Consider a map $f : A \to B$. Show that the following are equivalent:

(i) The map f is a weak equivalence.

(ii) The map f is ∞-connected, in the sense that f is k-connected for each k.

(iii) The map f is left orthogonal to any map between truncated types of any truncation level.

(iv) The map f is left orthogonal to any truncated map, for any truncation level.

Thus we see that, while the classes of k-connected maps and k-equivalences differ for finite $k \geq -1$, they come to agree at ∞.

32.11 Consider a pointed $(k+1)$-connected type X. Show that every k-truncated map $f : A \to X$ trivializes, in the sense that there is a k-type B and an equivalence $e : A \simeq X \times B$ for which the triangle

$$\begin{array}{ccc} A & \xrightarrow{e} & X \times B \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle \mathsf{pr}_1} & \\ & X & \end{array}$$

commutes.

32.12 Show that any commuting square

$$\begin{array}{ccc} E' & \xrightarrow{g} & E \\ {\scriptstyle p'}\downarrow & & \downarrow{\scriptstyle p} \\ B' & \xrightarrow[f]{} & B, \end{array}$$

in which p and p' are k-étale and f and g are k-equivalences, is a pullback square.

32.13 Consider a k-equivalence $f : B' \to B$. Show that the base-change functor induces an equivalence

$$\left(\sum_{(E:\mathcal{U})}\sum_{(p:E\to B)}\mathsf{is\text{-}etale}_k(p)\right) \simeq \left(\sum_{(E':\mathcal{U})}\sum_{(p':E'\to B')}\mathsf{is\text{-}etale}_k(p')\right).$$

In other words, for every k-étale map $p' : E' \to B'$ there is a unique k-étale map $p : E \to B$ equipped with a map $q : E' \to E$ such that the square

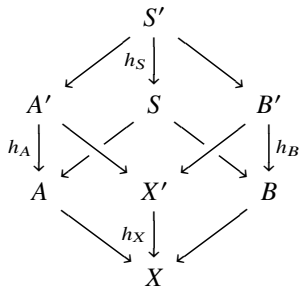
$$\begin{array}{ccc} E' & \overset{q}{\dashrightarrow} & E \\ {\scriptstyle p'}\downarrow & & \downarrow{\scriptstyle p} \\ B' & \xrightarrow[f]{} & B \end{array}$$

commutes and is a pullback square. In this sense k-étale maps descend along k-equivalences.

32.14 Consider two types X and Y. Show that the following are equivalent:

(i) Both X and Y are k-connected.
(ii) The type $X \times Y$ is k-connected.

32.15 Consider a commuting cube



in which h_A and h_B are 1-equivalences and h_S is a 0-equivalence. Show that h_X is a 1-equivalence.

33 The Blakers-Massey theorem

The Blakers-Massey theorem is a connectivity theorem which can be used to prove the Freudenthal suspension theorem, giving rise to the field of *stable homotopy theory*. It was proven in the setting of homotopy type theory by Lumsdaine et al, and their proof was the first that was given entirely in an elementary way, using only constructions that are invariant under homotopy equivalence.

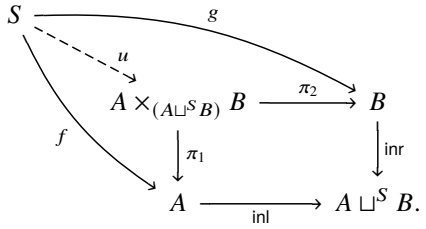
The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

33.1 The Blakers-Massey theorem

Consider a span $A \leftarrow S \rightarrow B$, consisting of an m-connected map $f : S \to A$ and an n-connected map $g : S \to B$. We take the pushout of this span, and subsequently the pullback of the resulting cospan, as indicated in the diagram



(33.1.1)

The universal property of the pullback determines a unique map $u : S \to A \times_{(A \sqcup^S B)} B$ as indicated.

Theorem 33.1.1 (Blakers-Massey) *The map $u : S \to A \times_{(A \sqcup^S B)} B$ of Eq. (33.1.1) is $(n+m)$-connected.*

33.2 The Freudenthal suspension theorem

Theorem 33.2.1 *If X is a k-connected pointed type, then the canonical map*

$$X \to \Omega(\Sigma X)$$

is $2k$-connected.

Theorem 33.2.2 *$\pi_n(\mathbf{S}^n) = \mathbb{Z}$ for $n \geq 1$.*

33.3 Higher groups

Recall that types in HoTT may be viewed as ∞-groupoids: elements are objects, paths are morphisms, higher paths are higher morphisms, etc.

It follows that *pointed connected* types B may be viewed as higher groups, with **carrier** ΩB. The neutral element is the identity path, the group operation is given by path composition, and higher paths witness the unit and associativity laws. Of course, these higher paths are themselves subject to further laws, etc., but the beauty of the type-theoretic definition is that we don't have to worry about that: all the (higher) laws follow from the rules of the identity types. Writing G for the carrier ΩB, it is common to write BG for the pointed connected type B, which comes equipped with an identification $G = \Omega BG$. We call BG the **delooping** of G.

The type of pointed types is $\mathcal{U}_{\mathsf{pt}} :\equiv \sum_{(A:\mathcal{U})} A$. The type of n-truncated

types is $\mathcal{U}^{\leq n} :\equiv \sum_{(A:\mathcal{U})} \mathsf{is\text{-}trunc}_n A$ and for n-connected types it is $\mathcal{U}^{>n} :\equiv \sum_{(A:\mathcal{U})} \mathsf{is\text{-}conn}_n(A)$. We will combine these notations as needed.

Definition 33.3.1 We define the type of **higher groups**, or **∞-groups**, to be

$$\infty\mathsf{Grp} :\equiv \sum_{(G:\mathcal{U})} \sum_{(BG:\mathcal{U}_{\mathsf{pt}}^{>0})} G \simeq \Omega BG.$$

When G is an ∞-group, we also write G for its first projection, called the **carrier** of G.

Remark 33.3.2 Note that we have equivalences

$$\begin{aligned}\infty\mathsf{Grp} &\doteq \sum_{(G:\mathcal{U})} \sum_{(BG:\mathcal{U}_{\mathsf{pt}}^{>0})} G \simeq \Omega BG \\ &\simeq \sum_{(G:\mathcal{U}_{\mathsf{pt}})} \sum_{(BG:\mathcal{U}_{\mathsf{pt}}^{>0})} G \simeq_{\mathsf{pt}} \Omega BG \\ &\simeq \mathcal{U}_{\mathsf{pt}}^{>0}\end{aligned}$$

for the type of higher groups.

Automorphism groups form a major class of examples of ∞-groups. Given *any* type A and any object $a : A$, the automorphism group at a is defined as **automorphism group** $\mathrm{Aut}\, a :\equiv (a = a)$. This is indeed an ∞-group, because it is the loop space of the connected component of A at a, i.e. we define $\mathsf{BAut} a :\equiv \mathrm{im}(a : 1 \to A) = (x : A) \times \|a = x\|_{-1}$. From this definition it is immediate that $\mathrm{Aut}\, a = \Omega\mathsf{BAut} a$, so we see that $\mathrm{Aut}\, a$ is indeed an example of an ∞-group.

If we take $A = \mathsf{Set}$, we get the usual symmetric groups $S_n :\equiv \mathrm{Aut}(\mathsf{Fin}_n)$, where Fin_n is a set with n elements. (Note that $BS_n = \mathsf{BAut}(\mathsf{Fin}_n)$ is the type of all n-element sets.)

We recover the ordinary set-level groups by requiring that G is a 0-type, or equivalently, that BG is a 1-type. This leads us to introduce:

Definition 33.3.3 We define the type of **groupal $(n-1)$-gropuoids**, or **n-groups**, to be

$$n\mathsf{Grp} :\equiv \sum_{(G:\mathcal{U}_{\mathsf{pt}}^{\leq n})} \sum_{(BG:\mathcal{U}_{\mathsf{pt}}^{>0})} G \simeq_{\mathsf{pt}} \Omega BG.$$

We write Grp for the type of 1-groups.

The type of n-groups is therefore equivalent to the type of pointed connected $(n+1)$-types. Note that if A is an $(n+1)$-type, then $\mathrm{Aut}\, a$ is an $(n+1)$-group because $\mathrm{Aut}\, a$ is n-truncated.

For example, the integers $\mathbb{Z}$ as an additive group are from this perspective represented by their delooping $\mathrm{B}\mathbb{Z} = \mathbb{S}^1$, i.e., the circle. Indeed, any set-level group G is represented as its delooping $BG :\equiv K(G, 1)$.

Moving across the homotopy hypothesis, for every pointed type (X, x) we have the **fundamental ∞-group of** X, $\Pi_\infty(X,x) :\equiv \mathrm{Aut}\, x$. Its $(n-1)$-truncation (an instance of decategorification, see Section 33.4) is the **fundamental n-group of** X, $\Pi_n(X,x)$, with corresponding delooping $\mathrm{B}\Pi_n(X,x) = \|\mathsf{BAut}x\|_n$.

Double loop spaces are more well-behaved than mere loop spaces. For example, they are commutative up to homotopy by the Eckmann-Hilton argument [5, Theorem 2.1.6]. Triple loop spaces are even better behaved than double loop spaces, and so on.

Definition 33.3.4 A type G is said to be **k-tuply groupal** if it comes equipped with a **k-fold delooping**, i.e. a pointed k-connected $B^k G : \mathcal{U}_{\mathsf{pt}}^{\geq k}$ and an equivalence $G \simeq \Omega^k B^k G$.

Mixing the two directions, we also define

$$\begin{aligned}(n,k)\mathrm{GType} &:\equiv \textstyle\sum_{(G:\mathcal{U}_{\mathsf{pt}}^{\leq n})} \sum_{(B^k G:\mathcal{U}_{\mathsf{pt}}^{\geq k})} G \simeq_{\mathsf{pt}} \Omega^k B^k G \\ &\simeq \mathcal{U}_{\mathsf{pt}}^{\geq k, \leq n+k}\end{aligned}$$

for the type of **k-tuply groupal n-groupoids**[4]. We allow taking $n = \infty$, in which case the truncation requirement is simply dropped.

Note that $n\mathsf{Grp} = (n-1, 1)\mathrm{GType}$. This shift in indexing is slightly annoying, but we keep it to stay consistent with the literature.

Note that for each $k \geq 0$ there is a forgetful map

$$(n, k+1)\mathrm{GType} \to (n,k)\mathrm{GType},$$

given by $B^{k+1}G \mapsto \Omega B^{k+1} G$, defining a sequence

$$\cdots \longrightarrow (n,2)\mathrm{GType} \longrightarrow (n,1)\mathrm{GType} \longrightarrow (n,0)\mathrm{GType}.$$

Thus we define $(n,\infty)\mathrm{GType}$ as the limit of this sequence:

$$\begin{aligned}(n,\infty)\mathrm{GType} &:\equiv \mathsf{lim}_k\, (n,k)\mathrm{GType} \\ &\simeq \textstyle\sum_{(B^- G:\prod_{(k:\mathbb{N})} \mathcal{U}_{\mathsf{pt}}^{\geq k, \leq n+k})} \prod_{(k:\mathbb{N})} B^k G \simeq_{\mathsf{pt}} \Omega B^{k+1} G.\end{aligned}$$

In Section 33.4 we prove the stabilization theorem (Theorem 33.4.10), from which it follows that $(n,\infty)\mathrm{GType} = (n,k)\mathrm{GType}$ for $k \geq n+2$.

The type $(\infty,\infty)\mathrm{GType}$ is the type of **stably groupal ∞-groups**, also known as **connective spectra**. If we also relax the connectivity requirement, we get the type of all spectra, and we can think of a spectrum as a kind of ∞-groupoid with k-morphisms for all $k \in \mathbb{Z}$.

[4] This is called $n\mathcal{U}_k$ in [**BaezDolan1998**], but here we give equal billing to n and k, and we add the "G" to indicate group-structure.

Table III.1 *Periodic table of k-tuply groupal n-groupoids.*

$k \setminus n$	0	1	2	$\cdots$	∞
0	pointed set	pointed groupoid	pointed 2-groupoid	$\cdots$	pointed ∞-groupoid
1	group	2-group	3-group	$\cdots$	∞-group
2	abelian group	braided 2-group	braided 3-group	$\cdots$	braided ∞-group
3	— " —	symmetric 2-group	sylleptic 3-group	$\cdots$	sylleptic ∞-group
4	— " —	— " —	symmetric 3-group	$\cdots$	?? ∞-group
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
Ω	— " —	— " —	— " —	$\cdots$	connective spectrum

The double hierarchy of higher groups is summarized in Table III.1. We shall prove the correctness of the $n = 0$ column in **??**.

A homomorphism between higher groups is any function that can be suitably delooped.

Definition 33.3.5 For $G, H : (n, k)\mathrm{GType}$, we define

$$\begin{aligned}\mathsf{hom}_{(n,k)}(G, H) &:\equiv \textstyle\sum_{(h:G\to_{\mathsf{pt}}H)} \sum_{(B^k h : B^k G \to_{\mathsf{pt}} B^k H)} \Omega^k(B^k h) \sim_{\mathsf{pt}} h \\ &\simeq (B^k h : B^k G \to_{\mathsf{pt}} B^k H).\end{aligned}$$

For (connective) spectra we need pointed maps between all the deloopings and pointed homotopies showing they cohere.

Note that if $h, k : G \to H$ are homomorphisms between set-level groups, then h and k are **conjugate** if $Bh, Bk : BG \to_{\mathsf{pt}} BH$ are **freely** homotopic (i.e., equal as maps $BG \to BH$).

Also observe that

$$\begin{aligned}\pi_j(B^k G \to_{\mathsf{pt}} B^k H) &\simeq \|B^k G \to_{\mathsf{pt}} \Omega^j B^k H\|_0 \\ &\simeq \|\Sigma^j B^k G \to_{\mathsf{pt}} B^k H\|_0 \\ &\simeq 0\end{aligned}$$

for $j > n$, which suggests that $\mathsf{hom}_{(n,k)}(G, H)$ is n-truncated. To prove this, we deviate slightly from the approach in [**BuchholtzDoornRijke**] and use the following intermediate result.

33.4 The stabilization theorem for higher groups

Definition 33.4.1 The **decategorification** Decat G of a k-tuply groupal $(n+1)$-group is defined to be the k-tuply groupal n-group $\|G\|_{n-1}$, which has delooping $\|B^k G\|_{n+k-1}$. Thus, decategorification is an operation

$$\text{Decat} : (n,k)\text{GType} \to (n-1,k)\text{GType}.$$

The functorial action of Decat is defined in the expected way. We also define the **∞-decategorification** ∞-Decat$G$ of a k-tuply groupal ∞-group as the k-tuply groupal n-group $\|G\|_n$, which has delooping $\|B^k G\|_{n+k}$.

Definition 33.4.2 The **discrete categorification** Disc G of a k-tuply-groupal $(n+1)$-group is defined to be the same ∞-group G, now considered as a k-tuply groupal $(n+2)$-group. Thus, the discrete categorification is an operation

$$\text{Disc} : (n,k)\text{GType} \to (n+1,k)\text{GType}.$$

Similarly, the **discrete ∞-decategorification** ∞-Disc$G$ of a k-tuply groupal $(n+1)$-group is defined to be the same group, now considered as a k-tuply groupal ∞-group.

Remark 33.4.3 The decategorification and discrete categorification functors make the $(n+1)$-category (n,k)GType a reflective sub-$(\infty,1)$-category of $(n+1,k)$GType. That is, there is an adjunction Decat $\dashv$ Disc. These properties are straightforward consequences of the universal property of truncation. Similarly, we have ∞-Decat $\dashv$ ∞-Disc such that the counit induces an isomorphism ∞-Decat $\circ$ ∞-Disc $=$ id.

For the next constructions, we need the following properties.

Definition 33.4.4 For $A : \mathcal{U}_{\mathsf{pt}}$ we define the **n-connected cover** of A to be $A\langle n\rangle :\equiv \mathsf{fib}_{A\to\|A\|_n}$. We have the projection $p_1 : A\langle n\rangle \to_{\mathsf{pt}} A$.

Lemma 33.4.5 *The universal property of the n-connected cover states the following. For any n-connected pointed type B, the pointed map*

$$(B \to_{\mathsf{pt}} A\langle n\rangle) \to_{\mathsf{pt}} (B \to_{\mathsf{pt}} A),$$

given by postcomposition with p_1, is an equivalence.

Proof Given a map $f : B \to_{\mathsf{pt}} A$, we can form a map $\widetilde{f} : B \to A\langle n\rangle$. First note that for $b : B$ the type $|fb|_n =_{\|A\|_n} |\mathsf{pt}|_n$ is $(n-1)$-truncated and inhabited for $b = \mathsf{pt}$. Since B is n-connected, the universal property for connected types shows that we can construct a $qb : |fb|_n = |\mathsf{pt}|_n$ for all b such that $q_0 : qb_0 \cdot \mathsf{ap}_{|-|_n}(f_0) = 1$. Then we can define the map $\widetilde{f}(b) :\equiv (fb, qb)$. Now $\widetilde{f}$ is pointed, because $(f_0, q_0) : (fb_0, qb_0) = (a_0, 1)$.

Now we show that this is indeed an inverse to the given map. On the one hand, we need to show that if $f : B \to_{\mathsf{pt}} A$, then $\mathsf{pr}_1 \circ \widetilde{f} = f$. The underlying functions are equal because they both send b to $f(b)$. They respect points in the same way, because $\mathsf{ap}p_1(\widetilde{f_0}) = f_0$. The proof that the other composite is the identity follows from a computation using fibers and connectivity, which we omit here, but can be found in the formalization. □

The next reflective sub-$(\infty, 1)$-category is formed by looping and delooping.

looping $\Omega : (n, k)\mathrm{GType} \to (n-1, k+1)\mathrm{GType}$
$\langle G, B^k G\rangle \mapsto \langle \Omega G, B^k G\langle k\rangle\rangle$
delooping $\mathrm{B} : (n, k)\mathrm{GType} \to (n+1, k-1)\mathrm{GType}$
$\langle G, B^k G\rangle \mapsto \langle \Omega^{k-1} B^k G, B^k G\rangle$

We have $\mathrm{B} \dashv \Omega$, which follows from Lemma 33.4.5 and $\Omega \circ \mathrm{B} = \mathsf{id}$, which follows from the fact that $A\langle n\rangle = A$ if A is n-connected.

The last adjoint pair of functors is given by stabilization and forgetting. This does not form a reflective sub-$(\infty, 1)$-category.

forgetting $F : (n, k)\mathrm{GType} \to (n, k-1)\mathrm{GType}$
$\langle G, B^k G\rangle \mapsto \langle G, \Omega B^k G\rangle$
stabilization $S : (n, k)\mathrm{GType} \to (n, k+1)\mathrm{GType}$
$\langle G, B^k G\rangle \mapsto \langle SG, \|\Sigma B^k G\|_{n+k+1}\rangle$,
where $SG = \|\Omega^{k+1}\Sigma B^k G\|_n$

We have the adjunction $S \dashv F$ which follows from the suspension-loop adjunction $\Sigma \dashv \Omega$ on pointed types.

The next main goal in this section is the stabilization theorem, stating that the ditto marks in Table III.1 are justified.

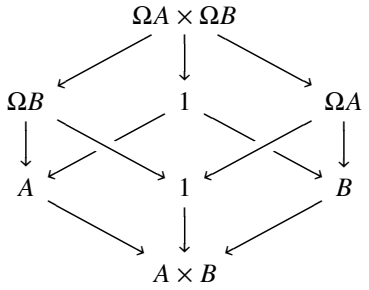
The following corollary is almost [5, Lemma 8.6.2], but proving this in Book HoTT is a bit tricky. See the formalization for details.

Lemma 33.4.6 (Wedge connectivity) *If $A : \mathcal{U}_{\mathsf{pt}}$ is n-connected and $B : \mathcal{U}_{\mathsf{pt}}$ is m-connected, then the map $A \vee B \to A \times B$ is $(n+m)$-connected.*

Let us mention that there is an alternative way to prove the wedge connectivity lemma: Recall that if A is n-connected and B is m-connected, then $A * B$ is $(n+m+2)$-connected [**joinconstruction**]. Hence the wedge connectivity lemma is also a direct consequence of the following lemma.

Lemma 33.4.7 *Let A and B be pointed types. The fiber of the wedge inclusion $A \vee B \to A \times B$ is equivalent to $\Omega A * \Omega B$.*

Proof Note that the fiber of $A \to A \times B$ is ΩB, the fiber of $B \to A \times B$ is ΩA, and of course the fiber of $1 \to A \times B$ is $\Omega A \times \Omega B$. We get a commuting cube



in which the vertical squares are pullback squares.

By the descent theorem for pushouts it now follows that $\Omega A * \Omega B$ is the fiber of the wedge inclusion. □

The second main tool we need for the stabilization theorem is:

Theorem 33.4.8 (Freudenthal) *If $A : \mathcal{U}_{\mathsf{pt}}^{\geq n}$ with $n \geq 0$, then the map $A \to \Omega\Sigma A$ is $2n$-connected.*

This is [5, Theorem 8.6.4].

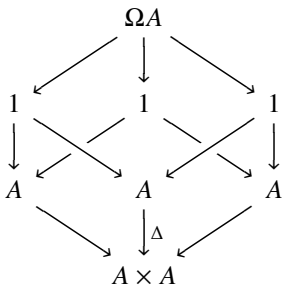
The final building block we need is:

Lemma 33.4.9 *There is a pullback square*

$$\begin{array}{ccc} \Sigma\Omega A & \longrightarrow & A \vee A \\ {\scriptstyle \varepsilon_A}\downarrow & & \downarrow \\ A & \xrightarrow[\Delta]{} & A \times A \end{array}$$

for any $A : \mathcal{U}_{\mathsf{pt}}$.

Proof Note that the pullback of $\Delta : A \to A \times A$ along either inclusion $A \to A \times A$ is contractible. So we have a cube



in which the vertical squares are all pullback squares. Therefore, if we pull back along the wedge inclusion, we obtain by the descent theorem for pushouts that the square in the statement is indeed a pullback square. □

Theorem 33.4.10 (Stabilization) *If $k \geq n+2$, then $S : (n,k)\mathrm{GType} \to (n,k+1)\mathrm{GType}$ is an equivalence, and any $G : (n,k)\mathrm{GType}$ is an infinite loop space.*

Proof We show that $F \circ S = \mathsf{id} = S \circ F : (n,k)\mathrm{GType} \to (n,k)\mathrm{GType}$ whenever $k \geq n+2$.

For the first, the unit map of the adjunction factors as

$$B^k G \to \Omega\Sigma B^k G \to \Omega\|\Sigma B^k G\|_{n+k+1}$$

where the first map is $2k-2$-connected by Freudenthal, and the second map is $n+k$-connected. Since the domain is $n+k$-truncated, the composite is an equivalence whenever $2k-2 \geq n+k$.

For the second, the counit map of the adjunction factors as

$$\|\Sigma\Omega B^k G\|_{n+k} \to \|B^k G\|_{n+k} \to B^k G,$$

where the second map is an equivalence. By the two lemmas above, the first map is $2k-2$-connected. □

For example, for $G : (0,2)\mathrm{GType}$ an abelian group, we have $B^n G = K(G,n)$, an Eilenberg-MacLane space.

The adjunction $S \dashv F$ implies that the free group on a pointed set X is $\Omega\|\Sigma X\|_1 = \pi_1(\Sigma X)$. If X has decidable equality, ΣX is already 1-truncated. It is an open problem whether this is true in general.

Also, the abelianization of a set-level group $G : 1\mathsf{Grp}$ is $\pi_2(\Sigma BG)$. If $G : (n,k)\mathrm{GType}$ is in the stable range ($k \geq n+2$), then $SFG = G$.

33.5 Eilenberg-Mac Lane spaces

Exercises

33.1 Show that if X is m-connected and $f : X \to Y$ is n-connected, then the map

$$X \to \mathsf{fib}_{m_f}(*)$$

where $m_f : Y \to M_f$ is the inclusion of Y into the cofiber of f, is $(m+n)$-connected.

33.2 Suppose that X is a connected type, and let $f : X \to Y$ be a map. Show that the following are equivalent:

(i) f is n-connected.

(ii) The mapping cone of f is $(n+1)$-connected.

33.3 Apply the Blakers-Massey theorem to the defining pushout square of the smash product to show that if A and B are m- and n-connected respectively, then there is a $(m+n+\min(m,n)+2)$-connected map

$$\Omega(A) * \Omega(B) \to \Omega(A \wedge B).$$

33.4 Show that the square

$$\begin{array}{ccc} \mathbf{1} & \longrightarrow & \mathsf{bool} \\ \downarrow & & \downarrow \\ X & \longrightarrow & X + \mathbf{1} \end{array}$$

is both a pullback and a pushout. Conclude that the result of the Blakers-Massey theorem is not always sharp.

33.5 Show that for every pointed type X, and any $n : \mathbb{N}$, there is a fiber sequence

$$K(\pi_{n+1}(X), n+1) \hookrightarrow \|X\|_{n+1} \twoheadrightarrow \|X\|_n.$$

33.6 Suppose that X is a pointed, n-connected type. Construct an equivalence

$$\|X\|_{n+1} \simeq K(\pi_{n+1}(X), n+1).$$

33.7 Define

$$\mathsf{EM}(G,n) :\equiv \textstyle\sum_{(X:\mathcal{U})} \|K(G,n) \simeq X\|$$
$$\mathsf{EM}_*(G,n) :\equiv \textstyle\sum_{(X:\mathcal{U})} \|K(G,n) \simeq X\| \times X.$$

(a) Show that $\mathsf{EM}_*(G,n) \simeq K(\mathrm{Aut}(G),1)$.

(b) (Lemma 2.7 in Scoccola!!!) For any pointed connected type X, and any $n \geq 1$, show that

$$(X \to_* \mathsf{EM}_*(G,n)) \simeq (\pi_1(X) \curvearrowright G).$$

33.8 Define

$$\mathsf{EM}_{**}(G,n) :\equiv \textstyle\sum_{(X:\mathcal{U})} \|K(G,n) \simeq X\| \times X \times X.$$

In other words, $\mathsf{EM}_{**}(G,n)$ is the type of doubly pointed Eilenberg-Mac Lane spaces of type (G,n). Construct an equivalence

$$\mathsf{EM}(G,n) \simeq \mathsf{EM}_{**}(G,n+1)$$

There are 306 exercises.

Ljubljana, November 10, 2021

34 Higher group theory

34.1 The category of pointed connected 1-types

Proposition 34.1.1 *Consider a k-connected map $f : X \to Y$, and a family P of $(k+n)$-truncated types over Y, where $n \geq 0$. Then the precomposition map*

$$- \circ f : \left(\prod_{(y:Y)} P(y)\right) \to \left(\prod_{(x:X)} P(f(x))\right)$$

is $(n-2)$-truncated.

Proposition 34.1.2 *Consider a pointed $(k+1)$-connected type X, and a family $Y : X \to \mathcal{U}^{\leq n+k}$ of $(n+k)$-truncated types over X. Then the map*

$$\mathsf{ev\text{-}pt} : \left(\prod_{(x:X)} Y(x)\right) \to Y(\mathsf{pt})$$

induced by the point inclusion $\mathbf{1} \to X$, is an $(n-2)$-truncated map.

Proof Note that we have a commuting triangle

$$\begin{array}{ccc} \left(\prod_{(x:X)} Y(x)\right) & & \\ {\scriptstyle -\circ \mathsf{const}_{\mathsf{pt}}}\swarrow & & \searrow{\scriptstyle \mathsf{ev\text{-}pt}} \\ \left(\prod_{(t:\mathbf{1})} Y(\mathsf{pt})\right) & \xrightarrow[\mathsf{ev\text{-}pt}]{\simeq} & Y(\mathsf{pt}), \end{array}$$

so the map on the left is an $(n-2)$-truncated map if and only if the map on the right is. For the map on the left, the claim follows immediately from Proposition 34.1.1, since the point inclusion $\mathsf{const}_{\mathsf{pt}} : \mathbf{1} \to X$ is a k-connected map by **??**. □

Definition 34.1.3 If $X : \mathcal{U}_{\mathsf{pt}}$ and $Y : X \to \mathcal{U}_{\mathsf{pt}}$, then we introduce the type of **pointed sections**,

$$\textstyle\prod^*_{(x:X)} Y(x) :\equiv \sum_{(s:\prod_{(x:X)} Y(x))} s(\mathsf{pt}) = \mathsf{pt}$$

This type is itself pointed by the trivial section $\lambda x.\, \mathsf{pt}$.

Corollary 34.1.4 *Consider a pointed k-connected type X, and a family $Y : X \to \mathcal{U}^{\leq n+k}_{\mathsf{pt}}$ of pointed $(n+k)$-truncated types over X. Then the type $\prod^*_{(x:X)} Y(x)$ is $(n-1)$-truncated.*

Proof Note that we have a pullback square

$$\begin{array}{ccc} \prod^*_{(x:X)} Y(x) & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow \\ \prod_{(x:X)} Y(x) & \xrightarrow[\mathsf{ev\text{-}pt}]{} & Y(*), \end{array}$$

so the claim follows from the fact that ev-pt is an $(n-1)$-truncated map. □

Theorem 34.1.5 *The type* $\mathsf{hom}_{(n,k)}(G,H)$ *is an* n*-type for any* $G, H : (n,k)\mathrm{GType}$.

Proof If X is $(k-1)$-connected, and Y is $(n+k)$-truncated, then the type of pointed maps $X \to_{\mathsf{pt}} Y$ is n-truncated. □

Corollary 34.1.6 *The type* $(n,k)\mathrm{GType}$ *is* $(n+1)$*-truncated.*

Proof This follows immediately from the preceding corollary, as the type of equivalences $G \simeq H$ is a subtype of the homomorphisms from G to H. □

If $k \geq n+2$ (so we're in the stable range), then $\mathsf{hom}_{(n,k)}(G,H)$ becomes a stably groupal n-groupoid. This generalizes the fact that the homomorphisms between abelian groups form an abelian group.

Corollary 34.1.7 *The automorphism group* $\mathrm{Aut}\, G$ *of a higher group* $G : (n,k)\mathrm{GType}$ *is a* 1*-groupal* $(n+1)$*-group, equivalent to the automorphism group of the pointed type* $B^k G$.

Proposition 34.1.8 *For any two pointed* n*-connected* $(n+k+1)$*-truncated types* X *and* Y*, the type of pointed maps*

$$X \to_* Y$$

is k*-truncated.*

Corollary 34.1.9 *For any two pointed* n*-connected* $(n+1)$*-truncated types* X *and* Y*, the type of pointed maps*

$$X \to_* Y$$

is a set.

Theorem 34.1.10 *The pre-category of* n*-connected* $(n+1)$*-truncated types in a universe* $\mathcal{U}$ *is Rezk complete.*

34.2 Equivalences of categories

Definition 34.2.1 A **functor** is...

Definition 34.2.2 A functor $F : \mathcal{C} \to \mathcal{D}$ is an equivalence if ...

34.3 The equivalence of groups and pointed connected 1-types

Theorem 34.3.1 *The loop space functor*

$$\mathsf{Type}_0^1 \to \mathsf{Group}$$

is an equivalence of categories.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

Appendix A

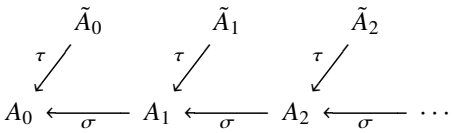
Algebraic dependent type theory

In this appendix we present type theory algebraically, since it might illuminate the intended meaning of the rules of type theory presented in Section 1. The algebras in this algebraic presentation are called **B-systems**, and the category of B-systems is equivalent to the category of contextual categories introduced by Cartmell [**cartmell**].

A.1 The structure underlying a dependent type theory

We begin by defining the underlying structure of a B-system.

Definition A.1.1 A **B-structure A** is a diagram of the form



In other words, a B-structure consists of two families $(A_n)_{n\in\mathbb{N}}$ and $(\tilde{A}_n)_{n\in\mathbb{N}}$ of sets, and two families of maps

$$\sigma : A_{n+1} \to A_n$$
$$\tau : \tilde{A}_n \to A_n$$

indexed by $n \in \mathbb{N}$.

Given a B-structure **A**, we think of the sets A_n as the classes of types in a context of length n: The set A_0 is the set of closed types of the B-structure **A**, the set A_1 is the set of types in a context of length one, and so forth. Similarly, the sets $\tilde{A}_n$ are the sets of elements of types in a context of length n. In the

following definition we give the precise interpretations of contexts and the four basic judgments of dependent type theory in a B-structure.

Definition A.1.2

(i) A **context of length** n in a B-structure $\mathbf{A}$ is a list of elements $\Gamma := (X_0, \ldots, X_{n-1})$ where $X_i \in A_i$ for $i < n$, such that $\sigma(X_{i+1}) = X_i$ for all $i < n - 1$. The **empty context** is the context of length 0.

(ii) A **type** in a context $\Gamma := (X_0, \ldots, X_{n-1})$ of length n in a B-structure $\mathbf{A}$ is an element $X \in A_n$ such that the equation $\sigma(X) = X_{n-1}$ holds if $n > 0$. When X is a type in context Γ, we write (Γ, X) for the context $(X_0, \ldots, X_{n-1}, X)$.

(iii) Two types X and X' in a context Γ of length n are said to be **judgmentally equal** if $X = X'$ in A_n.

(iv) An **element** of type X in context Γ of length n in a B-structure $\mathbf{A}$ is an element $x \in \tilde{A}_n$ such that $\tau(x) = X$.

(v) Two elements x and x' of type X in a context Γ of length n in a B-structure $\mathbf{A}$ are said to be **judgmentally equal** if $x = x'$ in $\tilde{A}_n$.

Remark A.1.3 Since equality on the sets A_n and $\tilde{A}_n$ are equivalence relations, it automatically follows that the interpretations of judgmental equality for types and for elements satisfies the rules that stipulate that judgmental equality is an equivalence relation. Moreover, the variable conversion rules are satisfied by the fact that contexts are defined up to equality.

Definition A.1.4 Consider a context $\Gamma := (X_0, \ldots, X_{n-1})$ in a B-structure $\mathbf{A}$. Then we define the **slice** B-structure $\mathbf{A}/\Gamma$ by

$$(A/\Gamma)_m := \{Y \in A_{m+n+1} \mid \sigma^{m+1}(Y) = X_{n-1}\}$$
$$(\tilde{A}/\Gamma)_m := \{y \in \tilde{A}_{m+n+1} \mid \tau(y) \in (A/\Gamma)_m\}.$$

The families of maps σ and τ on $\mathbf{A}/\Gamma$ are defined by restriction.

B-structures form a category with the following notion of morphism.

Definition A.1.5 Let $\mathbf{A}$ and $\mathbf{B}$ be B-structures. A **morphism** $f : \mathbf{A} \to \mathbf{B}$ consists of two families of maps

$$f : A_n \to B_n$$
$$\tilde{f} : \tilde{A}_n \to \tilde{B}_n$$

indexed by $n \in \mathbb{N}$, such that the squares

$$\begin{array}{ccc} A_{n+1} & \xrightarrow{f_{n+1}} & B_{n+1} \\ {\scriptstyle\sigma}\downarrow & & \downarrow{\scriptstyle\sigma} \\ A_n & \xrightarrow[f_n]{} & B_n \end{array} \qquad \begin{array}{ccc} \tilde{A}_n & \xrightarrow{\tilde{f}_n} & \tilde{B}_n \\ {\scriptstyle\tau}\downarrow & & \downarrow{\scriptstyle\tau} \\ A_n & \xrightarrow[f_n]{} & B_n \end{array}$$

commute. The identity morphism $\mathsf{id} : \mathbf{A} \to \mathbf{A}$ and the composite $g \circ f$ of morphisms of B-structures are defined in the obvious way, so that we obtain a category of B-structures.

Remark A.1.6 Morphisms of B-structures also act on contexts: Given a morphism $f : \mathbf{A} \to \mathbf{B}$ and a context $\Gamma := (X_0, \ldots, X_{n-1})$ in a B-structure $\mathbf{A}$, we define the context $f(\Gamma) := (f(X_0), \ldots, f(X_{n-1}))$. To see that $f(\Gamma)$ is again a context, note that

$$\sigma(f(X_i)) = f(\sigma(X_i)) = f(X_{i-1})$$

for each $i < n$. Furthermore, for any morphism $f : \mathbf{A} \to \mathbf{B}$ of B-structures, we define the morphism $f/\Gamma : \mathbf{A}/\Gamma \to \mathbf{B}/f(\Gamma)$ by restriction.

A.2 B-systems

Now that we have identified the basic structure underlying a dependent type theory, i.e., a B-structure, we have to interpret substitution, weakening, and the generic elements. Recall that the rule for substitution

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

asserts that when we have an element $a : X$ in context Γ, then we can substitute a for x in any type or element in context $\Gamma, x : X, \Delta$ to obtain a type or element in context $\Gamma, \Delta[a/x]$.

We will interpret substitution by operations on the types and the elements on a B-structure. Similarly, weakening and the generic elements are interpreted as operations on a B-structure. When a B-structure comes equippped with such substitution, weakening, and generic element structure we will call it a *pre-B-system*. For the full algebraic interpretation of dependent type theory, and hence for the full definition of a B-system, we will impose nine sets of laws on the substitution, weakening, and generic element structure.

Definition A.2.1 A **pre-B-system** consists of a B-structure $\mathbf{A}$ equipped with:

(i) A **substitution structure**: For every element a of type X in context Γ a morphism of B-structures

$$S_a : \mathbf{A}/(\Gamma, X) \to \mathbf{A}/\Gamma.$$

(ii) A **weakening structure**: For every type X in context Γ in $\mathbf{A}$ a morphism of B-structures

$$W_X : \mathbf{A}/\Gamma \to \mathbf{A}/(\Gamma, X).$$

(iii) A **generic element structure**: A family of maps

$$\delta : A_n \to \tilde{A}_{n+1}$$

indexed by $n \in \mathbb{N}$, such that $\tau(\delta(X)) = W_X(X)$ for each $X \in A_n$.

The algebraic definition of a dependent type theory is not yet complete. There are laws to be satisfied that ensure that substitution, weakening and the generic elements behave as expected.

Note that if we substitute twice in the syntactic presentation of type theory of Section 1, then it doesn't matter in which order we do that. That is, if we have an element $a : X$ in context Γ, and an element $b : Y$ in context $\Gamma, x : X, \Delta$, then we may substitute any type or element in context $\Gamma, x : X, \Delta, y : Y, E$ first by b and then by a, or alternatively we may substitute first by a and then by $b[a/x]$. The result will be the same in either case. Such laws are automatic in the syntax of dependent type theory. In the algebraic presentation of dependent type theory, however, we have to impose them. There are nine sets of laws that need to be imposed to complete the algebraic definition of a dependent type theory, i.e., of a B-system.

Definition A.2.2 A **B-system** is a pre-B-system $\mathbf{A}$ satisfying the **laws of a B-system**:

(i) **Substitution preserves substitution**: The equations

$$S_a(S_b(Z)) = S_{\tilde{S}_a(b)}(S_a(Z))$$
$$\tilde{S}_a(\tilde{S}_b(c)) = \tilde{S}_{\tilde{S}_a(b)}(\tilde{S}_a(c))$$

hold for every element a of type X in context $(X_0, \ldots, X_{n-1})$, every element b of type Y in context $(X_0, \ldots, X_{n-1}, X, Y_0, \ldots, Y_{m-1})$, and every element c of type Z in context $(X_0, \ldots, Y_{m-1}, Y, Z_0, \ldots, Z_{k-1})$ in $\mathbf{A}$.

(ii) **Substitution preserves weakening**: The equations

$$S_a(W_Y(Z)) = W_{S_a(Y)}(S_a(Z))$$
$$\tilde{S}_a(\tilde{W}_Y(c)) = \tilde{W}_{S_a(Y)}(\tilde{S}_a(c))$$

hold for every element a of type X in context $(X_0, \ldots, X_{n-1})$, every type Y in context $(X_0, \ldots, X_{n-1}, X, Y_0, \ldots, Y_{m-1})$, and every element c of type Z in context $(X_0, \ldots, Y_{m-1}, Z_0, \ldots, Z_{k-1})$ in $\mathbf{A}$.

(iii) **Substitution preserves the generic elements**: The equation

$$S_a(\delta(Y)) = \delta(S_a(Y))$$

holds for every element a of type X in context $(X_0, \ldots, X_{n-1})$ and every type Y in context $(X_0, \ldots, X_{n-1}, X, Y_0, \ldots, Y_{m-1})$ in $\mathbf{A}$.

(iv) **Weakening preserves subsitution**: The equations

$$W_X(S_b(Z)) = S_{\tilde{W}_X(b)}(W_X(Z))$$
$$\tilde{W}_X(\tilde{S}_b(c)) = \tilde{S}_{\tilde{W}_X(b)}(\tilde{W}_X(c))$$

hold for every type X in context $(X_0, \ldots, X_{n-1})$, every element b of type Y in context $(X_0, \ldots, X_{n-1}, Y_0, \ldots, Y_{m-1})$, and every element c of type Z in context $(X_0, \ldots, Y_{m-1}, Y, Z_0, \ldots, Z_{k-1})$ in $\mathbf{A}$.

(v) **Weakening preserves weakening**: The equations

$$W_X(W_Y(Z)) = W_{W_X(Y)}(W_X(Z))$$
$$\tilde{W}_X(\tilde{W}_Y(c)) = \tilde{W}_{W_X(Y)}(\tilde{W}_X(c))$$

hold for every type X in context $(X_0, \ldots, X_{n-1})$, every type Y in context $(X_0, \ldots, X_{n-1}, Y_0, \ldots, Y_{m-1})$, and every element c of type Z in context $(X_0, \ldots, Y_{m-1}, Z_0, \ldots, Z_{k-1})$ in $\mathbf{A}$.

(vi) **Weakening preserves the generic elements**: The equation

$$W_X(\delta(Y)) = \delta(W_X(Y))$$

holds for every type X in context $(X_0, \ldots, X_{n-1})$ and every type Y in context $(X_0, \ldots, X_{n-1}, Y_0, \ldots, Y_{m-1})$ in $\mathbf{A}$.

(vii) **Substitution cancels weakening**: The equations

$$S_a(W_X(Y)) = Y$$
$$\tilde{S}_a(\tilde{W}_X(b)) = b$$

hold for every element a of type X in context $(X_0, \ldots, X_{n-1})$ and every element b of type Y in context $(X_0, \ldots, X_{n-1}, Y_0, \ldots, Y_{m-1})$ in $\mathbf{A}$.

(viii) **Generic elements elements satisfy left unit law**: The equation

$$S_a(\delta(X)) = a$$

holds for every element a of type X in context $(X_0, \ldots, X_{n-1})$ in $\mathbf{A}$.

(ix) **Generic elements satisfy right unit law**: The equations

$$S_{\delta(X)}(W_X(Y)) = Y$$
$$\tilde{S}_{\delta(X)}(\tilde{W}_X(b)) = b$$

hold for every type X in context $(X_0, \ldots, X_{n-1})$ and every element b of type Y in context $(X_0, \ldots, X_{n-1}, X, Y_0, \ldots, Y_{m-1})$ in $\mathbf{A}$.

Remark A.2.3 Given a context $\Gamma := (X_0, \ldots, X_{n-1})$ in a B-system $\mathbf{A}$, we obtain another B-system $\mathbf{A}/\Gamma$ where

$$(A/\Gamma)_m := \{Y \in A_{n+m} \mid \sigma^{m+1}(Y) = X_{n-1}\}$$
$$(\tilde{A}/\Gamma)_m := \{y \in A_{n+m} \mid \tau(y) \in (A/\Gamma)_m\}.$$

The substitution, weakening, and generic element structures are defined by restriction, and satisfy the laws of a B-system.

Definition A.2.4 Given two B-systems $\mathbf{A}$ and $\mathbf{B}$, a morphism of B-systems $f : \mathbf{A} \to \mathbf{B}$ consists of a morphism of B-structures between their underlying B-structures such that

(i) The morphism f **preserves substitution**: For every element $a : X$ in context Γ in $\mathbf{A}$ the square

$$\begin{array}{ccc} \mathbf{A}/(\Gamma, X) & \xrightarrow{f/(\Gamma,X)} & \mathbf{B}/(f(\Gamma), f(X)) \\ {\scriptstyle S_a}\downarrow & & \downarrow{\scriptstyle S_{\tilde{f}(a)}} \\ \mathbf{A}/\Gamma & \xrightarrow[f/\Gamma]{} & \mathbf{B}/f(\Gamma) \end{array}$$

of morphisms of B-structures commutes.

(ii) The morphism f **preserves weakening**: For every type X in context Γ in $\mathbf{A}$ the square

$$\begin{array}{ccc} \mathbf{A}/\Gamma & \longrightarrow & \mathbf{B}/f(\Gamma) \\ \downarrow & & \downarrow \\ \mathbf{A}/(\Gamma, X) & \longrightarrow & \mathbf{B}/(f(\Gamma), f(X)) \end{array}$$

of morphisms of B-structures commutes.

(iii) The morphism f **preserves the generic elements**: For every type X in context Γ in $\mathbf{A}$ the equation

$$f(\delta(X)) = \delta(f(X))$$

holds.

Remark A.2.5 Note that, given a B-system $\mathbf{A}$, each S_a is a morphism of B-systems by conditions (i)-(iii), and each W_X is a morphism of B-systems by conditions (iv)-(vi).

A.3 From B-systems to contextual categories

The collection of contexts in a B-systems can be given the structure of a category. This category has the further structure of a contextual category, a notion of model of type theory due to Cartmell [**CartmellPhD**]. Contextual categories were later called C-systems by Voevodsky because the set-level nature of this notion of model of type theory, whereas (1-)categories are objects at the level of (1-)groupoids. In the language of the Univalent Foundations of mathematics, contextual categories have an underlying precategory of which the type of objects is a set. However, since we are working in set theory in this appendix, we will use the traditional vocabulary and call the underlying precategory a category.

In the following defintion we introduce contextual categories, and in the remainder of this section we show that every B-system gives rise to a contextual category.

Definition A.3.1 A **contextual category (C-system)** consists of

(i) A category C.
(ii) A length function $l : \mathrm{Ob}(C) \to \mathbb{N}$.
(iii) An chosen object $1 \in C$.
(iv) A function $\sigma : \mathrm{Ob}(C) \to \mathrm{Ob}(C)$.
(v) For any object $\Gamma \in \mathrm{Ob}(C)$ such that $l(\Gamma) > 0$, a morphism $p_\Gamma : \Gamma \to \sigma(\Gamma)$.
(vi) For any object $\Gamma \in \mathrm{Ob}(C)$ such that $l(\Gamma) > 0$ and any $f : \Delta \to \sigma(\Gamma)$ an object $f^*(\Gamma)$ and a morphism $q_f(\Gamma) : f^*(\Gamma) \to \Gamma$.

We divide this section in two subsections. In Appendix A.3.1 we construct the category of contexts of a B-system. In Appendix A.3.2 we show that this category of contexts has the structure of a contextual category.

A.3.1 The category of contexts of a B-system

Any B-system $\mathbf{A}$ gives rise to a set of contexts, for each context Γ a set of dependent contexts over Γ, and for each dependent context Δ over Γ a set of context elements of Δ in context Γ.

Definition A.3.1 Consider a context $\Gamma := (X_0, \ldots, X_{n-1})$ in a B-system $\mathbf{A}$.

(i) A **context element of** Γ is a list of elements $(x_0, \ldots, x_{n-1})$ of A_0 such that

$$\tau(x_i) = S_{x_{i-1}} \circ \cdots \circ S_{x_0}(X_i).$$

Given a context element $(x_0, \ldots, x_{n-1})$ of Γ, we will write $S_{(x_0,\ldots,x_{n-1})}$ for the composite $S_{x_{n-1}} \circ \cdots \circ S_{x_0}$.

(ii) A **dependent context over** Γ is a context in the B-system $\mathbf{A}/\Gamma$.

(iii) A **dependent context element** of a dependent context Δ over Γ is a context element of Δ in the B-system $\mathbf{A}/\Gamma$.

Definition A.3.2 Consider a B-system $\mathbf{A}$.

(i) For any context element $a := (a_0, \ldots, a_{m-1}) : \Gamma$, any dependent context Δ over Γ, and any dependent context element b of Δ over Γ we define

$$S_a(\Delta) := S_{a_{m-1}} \circ \cdots \circ S_{a_0}(\Delta)$$
$$S_a(b) := \tilde{S}_{a_{m-1}} \circ \cdots \circ \tilde{S}_{a_0}(b).$$

(ii) For any context $\Gamma := (X_0, \ldots, X_{n-1})$, any context Δ, and any context element b of Δ, we define

$$W_\Gamma(\Delta) := W_{X_{n-1}} \circ \cdots \circ W_{X_0}(\Delta)$$
$$\tilde{W}_\Gamma(b) := W_{X_{n-1}} \circ \cdots \circ W_{X_0}(b).$$

(iii) For any context $\Gamma := (X_0, \ldots, X_{n-1})$ we define the **generic context element** $\delta(\Gamma)$ of $W_\Gamma(\Gamma)$ over Γ by

$$(W_{(X_1,\ldots,X_{n-1})}(\delta(X_0)), \ldots, \delta(X_{n-1})).$$

Remark A.3.3 Substitution by context elements, weakening by contexts, and generic context elements satisfy laws analogous to the laws of a B-system.

Definition A.3.4 Consider two contexts Γ and Δ in B-system $\mathbf{A}$. The set of **context morphisms** $\mathsf{hom}(\Gamma, \Delta)$ is the set of dependent context elements of $W_\Gamma(\Delta)$ over Γ.

Definition A.3.5 The identity morphism on Γ is the generic context element on Γ.

Remark A.3.6 To verify that the generic context element $\delta(\Gamma)$ is indeed an element of the set $\mathsf{hom}(\Gamma, \Gamma)$ we need to show that the equation

$$\tau(W_{(X_{n-1},\ldots,X_{i+1})}(\delta(X_i))) = S_{(f_{i-1},\ldots,f_0)}(W_{(X_{n-1},\ldots,X_0)}(W_{(X_{n-1},\ldots,X_i)}(X_i)))$$

holds for each $i < n$.

Definition A.3.7 Given a context morphism $f : \mathsf{hom}(\Gamma, \Delta)$, and a context morphism $g : \mathsf{hom}(\Delta, \mathrm{E})$, we define the context morphism $g \circ f : \mathsf{hom}(\Gamma, \mathrm{E})$ by

$$g \circ f := \tilde{S}_f\,(\tilde{W}_\Gamma(g)).$$

Remark A.3.8 To veryfiy that the composite $g \circ f$ of context morphisms is again a context morphism, we need to show that...

Lemma A.3.9 *Composition of context morphisms is associative and satisfies the left and right unit laws.*

A.3.2 The structure of a C-system on the category of contexts

Definition A.3.1 For any context $\Gamma := (X_0, \ldots, X_n)$ of length $n+1$, we define

$$\sigma(\Gamma) := (X_0, \ldots, X_{n-1}).$$

Definition A.3.2 For any context $\Gamma := (X_0, \ldots, X_n)$ of length $n+1$, we define

$$p_\Gamma := W_{X_n}(\delta(\sigma(\Gamma))).$$

Definition A.3.3 For any context $\Gamma := (X_0, \ldots, X_n)$ of length $n+1$ and any context morphism $f : \Delta \to \sigma(\Gamma)$, we define the context

$$f^*(\Gamma) := (\Delta, S_f\,(W_\Delta(X_n))).$$

Furthermore, we define the context morphism $q(f, \Gamma) : \mathsf{hom}(f^*(\Gamma), \Gamma)$ by

$$q(f, \Gamma) :=$$

With the following theorem we conclude that the category of contexts of a B-system is a contextual category.

Theorem A.3.4 *For any context Γ of length $n+1$ and any context morphism $f : \Delta \to \sigma(\Gamma)$, the square*

$$\begin{array}{ccc} f^*(\Gamma) & \xrightarrow{q(f,\Gamma)} & \Gamma \\ {\scriptstyle p_{f^*(\Gamma)}}\downarrow & & \downarrow{\scriptstyle p_\Gamma} \\ \Delta & \xrightarrow[f]{} & \sigma(\Gamma) \end{array}$$

commutes and is a pullback square.

A.4 B-systems with dependent function types

Definition A.4.1 A **Π-structure** on a B-system $\mathbf{A}$ consists of a morphism of B-systems

$$\Pi_X : \mathbf{A}/(\Gamma, X) \to \mathbf{A}/\Gamma,$$

where we will write $\lambda_X := \tilde{\Pi}_X$, such that

(i) **Substitution preserves Π-types**: The equations

$$S_a(\Pi_Y(Z)) = \Pi_{S_a(Y)}(S_a(Z))$$
$$\tilde{S}_a(\lambda_Y(c)) = \lambda_{S_a(Y)}(\tilde{S}_a(c))$$

hold for every element a of type X in context Γ, every type Y in context (Γ, X, Δ), and every type Z and every element c of type Z in context $(\Gamma, X, \Delta, Y, \mathrm{E})$ in $\mathbf{A}$.

(ii) **Weakening preserves Π-types**: The equations

$$W_X(\Pi_Y(Z)) = \Pi_{W_X(Y)}(W_X(Z))$$
$$\tilde{W}_X(\lambda_Y(c)) = \lambda_{W_X(Y)}(\tilde{W}_X(c))$$

hold for every type X in context Γ, every type Y in context (Γ, Δ), and every type Z and every element c of type Z in context $(\Gamma, \Delta, Y, \mathrm{E})$ in $\mathbf{A}$.

(iii) For every type X in context Γ and every type Y in context (Γ, X, Δ) in $\mathbf{A}$, the map λ_X restricts to a bijection

$$\{y \in \tilde{A}_{n+m+1} \mid \tau(y) = Y\} \cong \{f \in \tilde{A}_{n+m} \mid \tau(f) = \Pi_X(Y)\}$$

A.5 B-systems with inductive types

Appendix B

General inductive types

Most inductive types we have seen in this book have a finite number of constructors with finite arities. For example, the type $\mathbb{N}$ has two constructors: one constant $0_{\mathbb{N}}$ and one unary constructor $\mathsf{succ}_{\mathbb{N}}$. However, there is no objection to having an nonfinite amount of constructors, possibly with nonfinite arities. W-types are general inductive types that have a (possibly nonfinite) *type* of constructors, whose arities are (possibly nonfinite) *types*. W-types are therefore specified by a type A of *symbols* for the constructors, and a type family B over A specifying the arities of the constructors that the symbols represent.

An example of a W-type is the type of finitely branching trees. This inductive type has a constructor with arity n, for each $n : \mathbb{N}$. In other words, a finitely branching tree is obtained by attaching any finite number of finitely branching trees to a root. The root itself is therefore a finitely branching tree, obtained from the 0-ary constructor, and if we have any finite number of finitely branching trees, we can combine them all into one finitely branching tree by attaching them to a new root.

In homotopy type theory, however, there is also the possibity to use types of higher truncation levels to specify inductive types. For example, we can construct the type of finitely branching trees with symmetries as the W-type with an X-ary constructor for each finite type X.

The most general form of inductive types is, however, the even more general notion of *indexed* W-type. Examples of indexed W-types include the ordinary W-types, but they also include inductive type families such as the identity type of an arbitrary type. In fact, the identity type of a W-type can be characterized as an indexed W-type. And indeed we will show that a similar fact holds for indexed W-types.

B.1 The type of well-founded trees

Definition B.1.1 Consider a type family B over A. The **W-type** $\mathsf{W}(A, B)$ is defined as the inductive type with constructor

$$\mathsf{tree} : \prod_{(x:A)} (B(x) \to \mathsf{W}(A, B)) \to \mathsf{W}(A, B).$$

The induction principle of the W-type $\mathsf{W}(A, B)$ asserts that, for any type family P over $\mathsf{W}(A, B)$, any dependent function

$$h : \prod_{(x:A)} \prod_{(\alpha:B(x)\to\mathsf{W}(A,B))} \left(\prod_{(y:B(x))} P(\alpha(x)) \right) \to P(\mathsf{tree}(x, \alpha))$$

determines a dependent function

$$\mathsf{ind}_{\mathsf{W}}(h) : \prod_{(x:\mathsf{W}(A,B))} P(x)$$

that satisfies the judgmental equality

$$\mathsf{ind}_{\mathsf{W}}(h, \mathsf{tree}(x, \alpha)) \doteq h(x, \alpha, \lambda y.\, \mathsf{ind}_{\mathsf{W}}(h, \alpha(y))).$$

We introduce some terminology about W-types.

Definition B.1.2 Given a W-type $\mathsf{W}(A, B)$, we say that

(i) The elements of $\mathsf{W}(A, B)$ are called **trees**.
(ii) the type A is the type of **symbols** of $\mathsf{W}(A, B)$,
(iii) the type $B(x)$ is the **arity** of the **tree-forming operation**

$$\mathsf{tree}(x) : (B(x) \to \mathsf{W}(A, B)) \to \mathsf{W}(A, B).$$

(iv) Given a map $\alpha : B(x) \to \mathsf{W}(A, B)$, the
(v) the element $\mathsf{tree}(x, \alpha)$ is the **($B(x)$-ary) tree** built out of the family

$$\alpha(y) : \mathsf{W}(A, B)$$

of elements in $\mathsf{W}(A, B)$ indexed by $x : B(x)$

Since its elements are (well-founded) trees, the type $\mathsf{W}(A, B)$ is also referred to as the type of **well-founded trees**.

Remark B.1.3 If our goal is to define a dependent function

$$f : \prod_{(x:\mathsf{W}(A,B))} P(x)$$

via the induction principle of W-types, we will often display that definition by pattern matching. Such definitions are then displayed as

$$f(\mathsf{tree}(x, \alpha)) :\equiv h(x, \alpha, \lambda y.\, f(\alpha(y))),$$

which contains all the information to carry out the construction via the induction

principle of W-types, and which directly displays the defining judgmental equality that the function f satisfies.

Remark B.1.4 For any $x : A$, the function

$$\mathsf{tree}(x) : (B(x) \to \mathsf{W}(A,B)) \to \mathsf{W}(A,B)$$

takes a family of elements $\alpha(y) : \mathsf{W}(A,B)$ indexed by $y : B(x)$ and builds out of them the element $\mathsf{tree}(x,\alpha) : \mathsf{W}(A,B)$. Since the element $\mathsf{tree}(x,\alpha)$ has been constructed out of a family $\alpha(y)$ of elements of $\mathsf{W}(A,B)$ indexed by $y : B(x)$, we say that the type $B(x)$ is the **arity** of $\mathsf{tree}(x,\alpha)$. In other words, there is a function

$$\mathsf{arity} : \mathsf{W}(A,B) \to \mathcal{U}$$

given by $\mathsf{arity}(\mathsf{tree}(x,\alpha)) :\equiv B(x)$. The element $x : A$ is the **symbol** of the operation $\mathsf{tree}(x) : (B(x) \to \mathsf{W}(A,B)) \to \mathsf{W}(A,B)$. Note that there might be many different symbols $x, y : A$ for which the operations $\mathsf{tree}(x)$ and $\mathsf{tree}(y)$ have equivalent arities, i.e., for which $B(x) \simeq B(y)$.

Furthermore, the **components** of $\mathsf{tree}(x,\alpha)$ are the elements $\alpha(y) : \mathsf{W}(A,B)$ indexed by $y : B(x)$. In other words, we have

$$\mathsf{comp} : \textstyle\prod_{(w:\mathsf{W}(A,B))} \mathsf{arity}(w) \to \mathsf{W}(A,B),$$

given by $\mathsf{comp}(\mathsf{tree}(x,\alpha)) :\equiv \alpha$.

In the special case where $B(x)$ is empty, there is exactly one family of elements $\alpha(y) : \mathsf{W}(A,B)$ indexed by $y : B(x)$. Therefore, it follows that any $x : A$ such that $B(x)$ is empty induces a constant in the W-type $\mathsf{W}(A,B)$. More precisely, if we are given a map $h : B(x) \to \emptyset$, then we can define the constant

$$c_x(h) :\equiv \mathsf{tree}(x, \mathsf{ex\text{-}falso} \circ h).$$

The elements of $w : \mathsf{W}(A,B)$ for which the type $B(\mathsf{arity}(w))$ is empty are called the **constants** of $\mathsf{W}(A,B)$. In other words, the predicate

$$\mathsf{is\text{-}constant}_{\mathsf{W}} : \mathsf{W}(A,B) \to \mathsf{Prop}_{\mathcal{U}}$$

is defined by $\mathsf{is\text{-}constant}_{\mathsf{W}}(w) :\equiv \mathsf{is\text{-}empty}(B(\mathsf{arity}(w)))$.

On the other hand, if each type $B(x)$ is inhabited, then there are no such constants and we will see in the following proposition that the W-type $\mathsf{W}(A,B)$ is empty in this case.

Proposition B.1.5 *Consider a family B of types over A, and suppose that each $B(x)$ is inhabited. Then the W-type $\mathsf{W}(A,B)$ is empty.*

Proof Assume that the proposition $\|B(x)\|$ holds for each $x : A$. We will construct a function $f : \mathsf{W}(A, B) \to \emptyset$ directly via the induction principle of W-types. Thus, we show that there is a function of type

$$\prod_{(x:A)} \prod_{(\alpha:B(x)\to \mathsf{W}(A,B))} \left(\prod_{(y:B(x))} \emptyset \right) \to \emptyset.$$

By the universal property of propositional truncations, this type is equivalent to the type

$$\prod_{(x:A)} \prod_{(\alpha:B(x)\to \mathsf{W}(A,B))} \neg\neg \|B(x)\|.$$

Given $x : A$ and $\alpha : B(x) \to \mathsf{W}(A, B)$, we obtain an element of type $\neg\neg\|B(x)\|$ via double negation introduction (see Exercise 4.3 (b)), and the hypothesis that $\|B(x)\|$ holds. □

Example B.1.6 Consider the type family P over bool given by

$$P(\mathsf{false}) :\equiv \emptyset \qquad \text{and} \qquad P(\mathsf{true}) :\equiv \mathbf{1}.$$

We claim that the W-type $N :\equiv \mathsf{W}(\mathsf{bool}, P)$ is equivalent to $\mathbb{N}$. The idea is that the constructor tree of $\mathsf{W}(\mathsf{bool}, P)$ splits into one nullary constructor corresponding the type $P(\mathsf{false})$, and one unary constructor corresponding to the type $P(\mathsf{true})$.

More formally, we define the zero element $z : N$ and the successor function $s : N \to N$ by

$$z :\equiv \mathsf{tree}(\mathsf{false}, \mathsf{ex\text{-}falso}) \qquad \text{and} \qquad s(x) :\equiv \mathsf{tree}(\mathsf{true}, \mathsf{const}_x).$$

Thus, we obtain a function $f : \mathbb{N} \to N$ that satisfies $f(0_{\mathbb{N}}) \doteq z$ and $f(\mathsf{succ}_{\mathbb{N}}(n)) \doteq s(f(n))$. It's inverse $g : N \to \mathbb{N}$ is defined via the induction principle of W-types by

$$g(\mathsf{tree}(\mathsf{false}, \alpha)) :\equiv 0_{\mathbb{N}}$$
$$g(\mathsf{tree}(\mathsf{true}, \alpha)) :\equiv \mathsf{succ}_{\mathbb{N}}(g(\alpha(\star))).$$

It is immediate from these definitions that $g(f(n)) = n$ for all $n : \mathbb{N}$. It remains to construct a path $p(x) : f(g(x)) = x$ for all $x : N$. Such a path is constructed inductively. First, there is a path

$$p(\mathsf{tree}(\mathsf{false}, \alpha)) : \mathsf{tree}(\mathsf{false}, \mathsf{ex\text{-}falso}) = \mathsf{tree}(\mathsf{false}, \alpha)$$

by the fact that $\mathsf{ex\text{-}falso} = \alpha$ for any $\alpha : \emptyset \to N$. Second, there is a path

$$p(\mathsf{tree}(\mathsf{true}, \alpha)) : \mathsf{tree}(\mathsf{true}, \mathsf{const}_{\alpha(\star)}) = \mathsf{tree}(\mathsf{true}, \alpha)$$

by the fact that $\mathsf{const}_{\alpha(\star)} = \alpha$ for any map $\alpha : \mathbf{1} \to N$.

Example B.1.7 Consider the type family B over bool given by

$$B(\mathsf{false}) :\equiv \emptyset \qquad \text{and} \qquad B(\mathsf{true}) :\equiv \mathsf{bool}.$$

We claim that the W-type $\mathsf{W}(\mathsf{bool}, B)$ is the type of *planar* binary trees. We can also present the type of planar binary trees as an inductive type with the following constructors:

$$\begin{aligned}\mathsf{leaf} &: \mathsf{planar}\text{-}\mathbb{T}_2 \\ [-,-] &: \mathsf{planar}\text{-}\mathbb{T}_2 \to (\mathsf{planar}\text{-}\mathbb{T}_2 \to \mathsf{planar}\text{-}\mathbb{T}_2).\end{aligned}$$

Example B.1.8 Consider the type $A :\equiv \mathbf{1}+\mathbb{F}_2$, where $\mathbb{F}_2$ is the type of 2-element types. We define the family B over A by pattern matching:

$$\begin{aligned}B(\mathsf{inl}(x)) &:\equiv \emptyset \\ B(\mathsf{inr}(X)) &:\equiv X.\end{aligned}$$

The type of **binary trees** is the W-type $\mathsf{W}(A, B)$ for this choice of A and B. We can also present the type of binary trees as an inductive type with the following constructors:

$$\begin{aligned}\mathsf{leaf} &: \mathbb{T}_2 \\ \mathsf{collect}\text{-}\mathbb{T}_2 &: \textstyle\prod_{(X:\mathbb{F}_2)} \mathbb{T}_2^X \to \mathbb{T}_2.\end{aligned}$$

There is an important qualitative difference between the type of planar binary trees and the type of binary trees. Given two distinct planar binary trees T_1 and T_2, the two planar binary trees $[T_1, T_2]$ and $[T_2, T_1]$ will also be distinct. On the other hand, given two binary trees T_1 and T_2, the binary trees

$$\begin{aligned}&\mathsf{collect}\text{-}\mathbb{T}_2(\mathsf{bool}, \mathsf{ind}\text{-}\mathsf{bool}(T_1, T_2)) \\ &\mathsf{collect}\text{-}\mathbb{T}_2(\mathsf{bool}, \mathsf{ind}\text{-}\mathsf{bool}(T_2, T_1))\end{aligned}$$

can always be identified.

Example B.1.9 The W-type $\mathsf{W}(\mathbb{N}, \mathsf{Fin})$ is the type of *ordered* finitely branching trees.

Example B.1.10 The W-type $\mathsf{W}(\mathbb{F}, \mathcal{T})$ is the type of (unordered) finitely branching trees.

B.2 Observational equality of W-types

Each element $x : \mathsf{W}(A, B)$ has a pre-arity $\mathsf{symbol}(x) : A$ and a family of components $\mathsf{comp}(x) : B(\mathsf{symbol}(x)) \to \mathsf{W}(A, B)$. Therefore, we have a map

$$\eta : \mathsf{W}(A, B) \to \textstyle\sum_{(x:A)} (B(x) \to \mathsf{W}(A, B))$$

given by $\eta(x) :\equiv (\mathsf{symbol}(x), \mathsf{comp}(x))$.

Proposition B.2.1 *The map $\eta : \mathsf{W}(A, B) \to \sum_{(x:A)} (B(x) \to \mathsf{W}(A, B))$ is an equivalence.*

Proof We define

$$\varepsilon : \left(\textstyle\sum_{(x:A)} (B(x) \to \mathsf{W}(A, B))\right) \to \mathsf{W}(A, B)$$

by $\varepsilon(x, \alpha) :\equiv \mathsf{tree}(x, \alpha)$. The fact that ε is an inverse of η follows easily. □

The fact that we have an equivalence

$$\mathsf{W}(A, B) \simeq \textstyle\sum_{(x:A)} (B(x) \to \mathsf{W}(A, B)),$$

suggests a way to characterize the identity type of $\mathsf{W}(A, B)$. Indeed, any equivalence is an embedding, and therefore we also have

$$(x = y) \simeq (\eta(x) = \eta(y)).$$

The latter is an identity type in a Σ-type, which can be characterized as a Σ-type of identity types. We therefore define the following observational equality relation on $\mathsf{W}(A, B)$.

Definition B.2.2 Suppose A and each $B(x)$ are in $\mathcal{U}$. We define a binary relation

$$\mathsf{Eq}_{\mathsf{W}} : \mathsf{W}(A, B) \to \mathsf{W}(A, B) \to \mathcal{U}$$

recursively by

$$\mathsf{Eq}_{\mathsf{W}}(\mathsf{tree}(x, \alpha), \mathsf{tree}(y, \beta)) :\equiv \textstyle\sum_{(p:x=y)} \prod_{(z:B(x))} \alpha(z) = \beta(\mathsf{tr}_B(p, z))$$

Theorem B.2.3 *The observational equality relation Eq_{W} on $\mathsf{W}(A, B)$ is reflexive, and the canonical map*

$$(x = y) \to \mathsf{Eq}_{\mathsf{W}}(x, y)$$

is an equivalence for each $x, y : \mathsf{W}(A, B)$.

Proof The element $\mathsf{refl\text{-}Eq_W}(x) : \mathsf{Eq_W}(x, x)$ is defined recursively as

$$\mathsf{refl\text{-}Eq_W}(\mathsf{tree}(x, \alpha)) :\equiv (\mathsf{refl}_x, \mathsf{refl\text{-}htpy}_\alpha).$$

This proof of reflexivity induces the canonical map $(x = y) \to \mathsf{Eq_W}(x, y)$. To show that it is an equivalence for each $x, y : \mathsf{W}(A, B)$, we apply the fundamental theorem of identity types, by which it suffices to show that the type

$$\textstyle\sum_{(y:\mathsf{W}(A,B))} \mathsf{Eq_W}(x, y)$$

is contractible for each $x : \mathsf{W}(A, B)$. The center of contraction is the pair $(x, \mathsf{refl\text{-}Eq_W}(x))$. For the contraction, we have to construct a function

$$h : \textstyle\prod_{(y:\mathsf{W}(A,B))} \prod_{(p:\mathsf{Eq_W}(x,y))} (x, \mathsf{refl\text{-}Eq_W}(x)) = (y, p).$$

By the induction principle of W-types, it suffices to define

$$h(\mathsf{tree}(y, \beta), (p, H)) :\equiv (x, (\mathsf{refl}, \mathsf{refl\text{-}htpy})) = (y, (p, H)).$$

Here we proceed by path induction on $p : x = y$, followed by homotopy induction on the homotopy $h : \alpha \sim \beta$. Thus, it suffices to construct an identification

$$(x, (\mathsf{refl}, \mathsf{refl\text{-}htpy})) = (x, (\mathsf{refl}, \mathsf{refl\text{-}htpy})),$$

which we have by reflexivity. □

Theorem B.2.4 *Consider a type family B over a type A, and let $k : \mathbb{T}$ be a truncation level. If A is a $(k + 1)$-type, then so is $\mathsf{W}(A, B)$.*

Proof Suppose that A is a $(k + 1)$-type. In order to show that $\mathsf{W}(A, B)$ is a $(k + 1)$-type, we have to show that its identity types are k-types. The proof is by induction on $x, y : \mathsf{W}(A, B)$. For $x \doteq \mathsf{tree}(a, \alpha)$ and $y \doteq \mathsf{tree}(b, \beta)$, we have the equivalence

$$(\mathsf{tree}(a, \alpha) = \mathsf{tree}(b, \beta)) \simeq \textstyle\sum_{(p:a=b)} \prod_{(z:B(a))} \alpha(z) = \beta(\mathsf{tr}_B(p, z))$$

Note that the type $a = b$ is a k-type by the assumption that A is a $(k + 1)$-type. Furthermore, the type $\alpha(z) = \beta(\mathsf{tr}_B(p, z))$ is a k-type by the induction hypothesis. Therefore it follows that the type on the right-hand side of the displayed equivalence is a k-type, and this completes the proof. □

B.3 Functoriality of W-types

Definition B.3.1 Consider a type family B over A, and a type family B' over A'. Furthermore, consider a map $f : A' \to A$ and a family of equivalences

$$e_x : B'(x) \simeq B(f(x))$$

indexed by $x : A'$. Then we define the **morphism** $\mathsf{W}(f,e) : \mathsf{W}(A',B') \to \mathsf{W}(A,B)$ of W-types inductively by

$$\mathsf{W}(f,e)(\mathsf{tree}(x,\alpha)) :\equiv \mathsf{tree}(f(x), \mathsf{W}(f,g) \circ \alpha \circ e_x^{-1}).$$

Lemma B.3.2 *For any morphism* $\mathsf{W}(f,e) : \mathsf{W}(A',B') \to \mathsf{W}(A,B)$ *of W-types and any* $\mathsf{tree}(x,\alpha) : \mathsf{W}(A,B)$*, there is an equivalence*

$$\mathsf{fib}_{\mathsf{W}(f,e)}(\mathsf{tree}(x,\alpha)) \simeq \mathsf{fib}_f(x) \times \textstyle\prod_{(b:B(x))} \mathsf{fib}_{\mathsf{W}(f,e)}(\alpha(b)).$$

Proof First, note that by the characterization in Theorem B.2.3 of the identity type of $\mathsf{W}(A,B)$, there is an equivalence between the fiber $\mathsf{fib}_{\mathsf{W}(f,e)}(\mathsf{tree}(x,\alpha))$ and the type

$$\textstyle\sum_{(x':A')} \sum_{(\alpha':B'(x')\to\mathsf{W}(A',B'))} \sum_{(p:f(x')=x)} \prod_{(b:B(f(x')))} \mathsf{W}(f,e)(\alpha'(e_{x'}^{-1}(b))) = \alpha(\mathsf{tr}_B(p,b)).$$

By rearranging the Σ-type, we see that this type is equivalent to the type

$$\textstyle\sum_{((x',p):\mathsf{fib}_f(x))} \sum_{(\alpha':B'(x')\to\mathsf{W}(A',B'))} \prod_{(b:B(f(x')))} \mathsf{W}(f,e)(\alpha'(e_{x'}^{-1}(b))) = \alpha(\mathsf{tr}_B(p,b)).$$

Therefore, it suffices to show for each $(x',p) : \mathsf{fib}_f(x)$, that the type

$$\textstyle\sum_{(\alpha':B'(x')\to\mathsf{W}(A',B'))} \prod_{(b:B(f(x')))} \mathsf{W}(f,e)(\alpha'(e_{x'}^{-1}(b))) = \alpha(\mathsf{tr}_B(p,b))$$

is equivalent to the type $\prod_{(b:B(x))} \mathsf{fib}_{\mathsf{W}(f,e)}(\alpha(b))$. Since we have an identification $p : f(x') = x$ and an equivalence $e_{x'} : B'(x') \simeq B(f(x'))$, it follows that the type above is equivalent to the type

$$\textstyle\sum_{(\alpha':B(x)\to\mathsf{W}(A',B'))} \prod_{(b:B(x))} \mathsf{W}(f,e)(\alpha'(b)) = \alpha(b).$$

By distributitivity of Π over Σ, i.e., by Theorem 13.2.1, this type is equivalent to the type

$$\textstyle\prod_{(b:B(x))} \sum_{(w:\mathsf{W}(A',B'))} \mathsf{W}(f,e)(w) = \alpha(b),$$

completing the proof. □

Theorem B.3.3 *Consider a morphism* $\mathsf{W}(f,e) : \mathsf{W}(A,B) \to \mathsf{W}(A',B')$ *of W-types. If the map* $f : A \to A'$ *is* k*-truncated, then so is the map* $\mathsf{W}(f,e)$*. In particular, if* f *is an equivalence or an embedding, then so is* $\mathsf{W}(f,e)$*.*

Proof Suppose that the map f is k-truncated. We will prove recursively that the fibers of the morphism $\mathsf{W}(f,e)$ on W-types is k-truncated. We saw in Lemma B.3.2 that there is an equivalence

$$\mathsf{fib}_{\mathsf{W}(f,e)}(\mathsf{tree}(x,\alpha)) \simeq \mathsf{fib}_f(x) \times \textstyle\prod_{(b:B(x))} \mathsf{fib}_{\mathsf{W}(f,e)}(\alpha(b)).$$

The type $\mathsf{fib}_f(x)$ is k-truncated by assumption, and each of the types $\mathsf{fib}_{\mathsf{W}(f,e)}(\alpha(b))$ are k-truncated by the inductive hypothesis, so the claim follows. □

B.4 Extensional W-types

The elements of a W-type $\mathsf{W}(A,B)$ are constructed out of families of elements of $\mathsf{W}(A,B)$ indexed by a type $B(x)$ for some $x : A$. More precisely, for each $\mathsf{tree}(x,\alpha) : \mathsf{W}(A,B)$ we have a family of elements

$$\alpha(y) : \mathsf{W}(A,B)$$

indexed by $y : B(x)$. Thus, we could say that $\alpha(y)$ is in $\mathsf{tree}(x,\alpha)$, for each $y : B(x)$. More abstractly, we can define an elementhood relation on $\mathsf{W}(A,B)$.

Definition B.4.1 Given a W-type $\mathsf{W}(A,B)$ and a universe $\mathcal{U}$ containing both A and each type in the family B, we define a type-valued relation

$$\in : \mathsf{W}(A,B) \to \mathsf{W}(A,B) \to \mathcal{U}$$

by $(x \in \mathsf{tree}(a,\alpha)) :\equiv \sum_{(y:B(a))} \alpha(y) = x$.

Using the elementhood relation on $\mathsf{W}(A,B)$, we can reformulate the induction principle to, perhaps, a more recognizable form:

Theorem B.4.2 *For any family P of types over $\mathsf{W}(A,B)$, there is a function*

$$i : \left(\prod_{(x:\mathsf{W}(A,B))} \left(\prod_{(y:\mathsf{W}(A,B))} (y \in x) \to P(y)\right) \to P(x)\right) \to \left(\prod_{(x:X)} P(x)\right)$$

that comes equipped with an identification

$$i(h,x) = h(x, \lambda y.\, \lambda e.\, i(h,y))$$

for every $h : \prod_{(x:\mathsf{W}(A,B))} \left(\prod_{(y:\mathsf{W}(A,B))} (y \in x) \to P(y)\right) \to P(x)$, and every $x : \mathsf{W}(A,B)$.

Proof For any type family P over $\mathsf{W}(A,B)$, we first define a new type family $\Box P$ over $\mathsf{W}(A,B)$ given by

$$\Box P(x) :\equiv \prod_{(y:\mathsf{W}(A,B))} (y \in x) \to P(y).$$

The family $\Box P(x)$ comes equipped with a map

$$\eta : \left(\prod_{(x:\mathsf{W}(A,B))} P(x)\right) \to \left(\prod_{(x:\mathsf{W}(A,B))} \Box P(x)\right)$$

given by $\eta(f,x,y,e) :\equiv f(y)$. Conversely, there is a map

$$\varepsilon(h) : \left(\prod_{(y:\mathsf{W}(A,B))} \Box P(y)\right) \to \left(\prod_{(x:\mathsf{W}(A,B))} P(x)\right)$$

for every $h : \prod_{(y:\mathsf{W}(A,B))} \Box P(y) \to P(y)$, given by $\varepsilon(h, g, x) :\equiv h(x, g(x))$. Note that the induction principle can now be stated as

$$i : \left(\prod_{(y:\mathsf{W}(A,B))} \Box P(y) \to P(y)\right) \to \left(\prod_{(x:\mathsf{W}(A,B))} P(x)\right),$$

and the computation rule states that

$$i(h, x) = h(x, \eta(i(h), x)).$$

Before we prove the induction principle, we prove the intermediate claim that there is a function

$$i' : \left(\prod_{(y:\mathsf{W}(A,B))} \Box P(y) \to P(y)\right) \to \left(\prod_{(x:\mathsf{W}(A,B))} \Box P(x)\right)$$

equipped with an identification

$$j'(h, x, y, e) : i'(h, x, y, e) = h(y, i'(h, y))$$

for every $h : \prod_{(y:\mathsf{W}(A,B))} \Box P(y) \to P(y)$ and every $x, y : \mathsf{W}(A, B)$ equipped with $e : y \in x$. Both i' and j' are defined by pattern matching:

$$i'(h, \mathsf{tree}(a, f), f(b), (b, \mathsf{refl})) :\equiv h(f(b), i'(h, f(b)))$$
$$j'(h, \mathsf{tree}(a, f), f(b), (b, \mathsf{refl})) :\equiv \mathsf{refl}.$$

Now we define $i(h) :\equiv \varepsilon(h, i'(h))$. Note that we have the judgmental equalities

$$\begin{aligned} i(h, x) &\doteq \varepsilon(h, i'(h), x) \\ &\doteq h(x, i'(h, x)), \end{aligned}$$

and

$$\begin{aligned} h(x, \lambda y.\, \lambda e.\, i(h, y)) &\doteq h(x, \lambda y.\, \lambda e.\, \varepsilon(h, i'(h), y)) \\ &\doteq h(x, \lambda y.\, \lambda e.\, h(y, i'(h, y))). \end{aligned}$$

The computation rule is therefore satisfied by the identification

$$h(x, i'(h, x)) \xrightarrow{\mathsf{ap}_{h(x)}(\mathsf{eq\text{-}htpy}(\lambda y.\, \mathsf{eq\text{-}htpy}(j'(h,x,y))))} h(x, \lambda y.\, \lambda e.\, h(y, i'(h, y))).$$

□

It is tempting to think that an element $w : \mathsf{W}(A, B)$ is completely determined by the elements $z : \mathsf{W}(A, B)$ equipped with a proof $z \in w$. However, this may not be the case. For instance, a W-type $\mathsf{W}(A, B)$ might have *two* unary constructors, e.g., when $A :\equiv \mathbf{1} + \mathsf{bool}$ and the family B over A is given by

$$B(\mathsf{inl}(x)) :\equiv \emptyset$$

$$B(\mathsf{inr}(y)) :\equiv \mathbf{1}.$$

If we write f and g for the two unary constructors of $\mathsf{W}(A, B)$, then we see that for any element $w : \mathsf{W}(A, B)$, the elements

$$u :\equiv \mathsf{tree}(\mathsf{inr}(\mathsf{false}), \mathsf{const}_w) \qquad \text{and} \qquad v :\equiv \mathsf{tree}(\mathsf{inr}(\mathsf{true}), \mathsf{const}_w)$$

both only contain the element w. However, the elements u and v are distinct in $\mathsf{W}(A, B)$.

Something similar happens in the type of planar binary trees. Given two binary trees S and T, there are two ways to combine S and T into a new binary tree: we have $[S, T]$ and $[T, S]$. Both contain precisely the elements S and T, but they are distinct. Nevertheless, there are many important W-types in which the elements w are uniquely determined by the elements $z \in w$. Such W-types are called extensional.

Definition B.4.3 We say that a W-type $\mathsf{W}(A, B)$ is **extensional** if the canonical map

$$(x = y) \to \textstyle\prod_{(z:\mathsf{W}(A,B))} (z \in x) \simeq (z \in y)$$

is an equivalence.

In the following theorem we give a precise characterization of the inhabited extensional W-types.

Theorem B.4.4 *Consider an inhabited W-type $\mathsf{W}(A, B)$. Then the following are equivalent:*

(i) *The W-type $\mathsf{W}(A, B)$ is extensional.*
(ii) *The family B is **univalent** in the sense that the map*

$$\mathsf{tr}_B : (x = y) \to (B(x) \simeq B(y))$$

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

Remark B.4.5 Note that if the W-type $\mathsf{W}(A, B)$ is empty, then it is vacuously extensional. However, we saw in Proposition B.1.5 that any family B of inhabited types over A gives rise to an empty W-type $\mathsf{W}(A, B)$, so there is no hope of showing that B is a univalent family if $\mathsf{W}(A, B)$ is empty.

We also note that a type family B over A is univalent if and only if the map $B : A \to \mathcal{U}$ is an embedding. In other words, the claim in Theorem B.4.4 is that an inhabited W-type $\mathsf{W}(A, B)$ is extensional if and only if B is the canonical type family over a subuniverse A of $\mathcal{U}$.

Proof We will first show that (ii) is equivalent to the following property:

(ii') The map

$$\mathsf{tr}_B : (\mathsf{symbol}(x) = y) \to (B(\mathsf{symbol}(x)) \simeq B(y))$$

is an equivalence for every $x : \mathsf{W}(A, B)$ and every $y : A$.

Clearly, (ii) implies (ii'). For the converse we use the assumption that $\mathsf{W}(A, B)$ is inhabited. Since the property in (ii) is a proposition, we may assume an element $w : \mathsf{W}(A, B)$. Using w, we obtain for every $x : A$ the element

$$\mathsf{tree}(x, \mathsf{const}_w) : \mathsf{W}(A, B)$$

The pre-arity of $\mathsf{tree}(x, \mathsf{const}_w)$ is x, and therefore the hypothesis that (ii') holds implies that the map $(x = y) \to (B(x) \simeq B(y))$ is an equivalence. This concludes the proof that (ii) is equivalent to (ii'). It remains to show that (i) is equivalent to (ii').

Let $x : \mathsf{W}(A, B)$. By the fundamental theorem of identity types, the W-type $\mathsf{W}(A, B)$ is extensional if and only if the total space

$$\textstyle\sum_{(y:\mathsf{W}(A,B))} \prod_{(z:\mathsf{W}(A,B))} (z \in x) \simeq (z \in y)$$

is contractible, for any $x : \mathsf{W}(A, B)$. When x is of the form $\mathsf{tree}(a, \alpha)$, the type $z \in x$ is just the fiber $\mathsf{fib}_\alpha(z)$. Using this observation, we see that the above type is equivalent to the type

$$\textstyle\sum_{(b:A)} \sum_{(\beta:B(b)\to\mathsf{W}(A,B))} \prod_{(z:\mathsf{W}(A,B))} \mathsf{fib}_\alpha(z) \simeq \mathsf{fib}_\beta(z). \tag{$*$}$$

By Exercise 13.15 (c) it follows that this type is equivalent to the type

$$\textstyle\sum_{(y:A)} \sum_{(\beta:B(y)\to\mathsf{W}(A,B))} \sum_{(e:B(x)\simeq B(y))} \alpha \sim e \circ \beta.$$

Note that the type $\sum_{(\beta:B(y)\to\mathsf{W}(A,B))} \alpha \sim e \circ \beta$ is contractible for any equivalence $e : B(x) \simeq B(y)$. Therefore, it follows that the above type is contractible if and only if the type

$$\textstyle\sum_{(y:A)} B(x) \simeq B(y)$$

is contractible, which is the case if and only if the map $(x = y) \to (B(x) \simeq B(y))$ is an equivalence for all $y : A$. □

Example B.4.6 The type N of Example B.1.6, the type of binary trees Example B.1.8, and the type of finitely branching trees Example B.1.10 are all examples extensional W-types. On the other hand, the type of planar binary trees of Example B.1.7 and the type of planar finitely branching tres of Example B.1.9 are not extensional.

B.5 Russell's paradox in type theory

Russell's paradox tells us that there cannot be a set of all sets. If there were such a set S, then we could form the set

$$R :\equiv \{x \in S \mid x \notin x\},$$

for which we have $R \in R \leftrightarrow R \notin R$, a contradiction. To reproduce Russell's paradox in type theory, we first recall a crucial difference between the type theoritic judgment $a : A$ and the set theoretic proposition $x \in y$. Although the judgment $a : A$ plays a similar role in type theory as the elementhood relation, types and their elements are fundamentally different entities, whereas in Zermelo-Fraenkel set theory there are only sets, and the proposition $x \in y$ can be formed for any two sets x and y. In type theory, there is no relation on the universe that is similar to the elementhood relation.

However, we have seen in Appendix B.4 that it is possible to define an elementhood relation on arbitrary W-types. We will use this elementhood relation on the W-type $\mathsf{W}(\mathcal{U}, \mathcal{T})$ to derive a paradox analogous to Russell's paradox, and we will see that $\mathcal{U}$ cannot be equivalent to a type in $\mathcal{U}$.

The type $\mathsf{W}(\mathcal{U}, \mathcal{T})$ possesses a lot of further structure. In fact, it can be used to encode constructive set theory in type theory. There is, however, one significant difference with ordinary set theory: the elementhood relation it type-valued. In other words, there may be many ways in which $x \in y$ holds. The type $\mathsf{W}(\mathcal{U}, \mathcal{T})$ is therefore also called the type of **multisets**. It was fist studied by Aczel in [**AczelCZF**], with refinements in [**AczelGambinoCZF**], and in the setting of homotopy type theory it has been studied extensively by Gylterud in [**GylterudMultisets**].

Definition B.5.1 Consider a $\mathcal{U}$ with universal type family $\mathcal{T}$. We define the type

$$\mathbb{M}_{\mathcal{U}} :\equiv \mathsf{W}(\mathcal{U}, \mathcal{T}),$$

and the elements of $\mathbb{M}_{\mathcal{U}}$ are called **multisets in** $\mathcal{U}$. We will write

$$\{f(x) \mid x : A\}$$

for the multiset in $\mathcal{U}$ of the form $\mathsf{tree}(A, f)$. More generally, we will write

$$\{t(x_0, \ldots, x_n) \mid x_0 : A_0, \ldots, x_n : A_n(x_0, \ldots, x_{n-1})\}$$

for the multiset in $\mathcal{U}$ of the form

$$\mathsf{tree}\left(\textstyle\sum_{(x_0:A_0)} \cdots A_n(x_0, \ldots, x_{n-1}), \lambda(x_0, \ldots, x_n).\, t(x_0, \ldots, x_n)\right),$$

given an element $t(x_0, \ldots, x_n) : \mathbb{M}_{\mathcal{U}}$ in context $x_0 : A_0, \ldots, x_n : A_n(x_0, \ldots, x_{n-1})$, where each A_i is in $\mathcal{U}$.

Given a multiset $X \doteq \{f(x) \mid x : A\}$ in $\mathcal{U}$, the **cardinality** of X is the type A, and the **elements** of X are the multisets $f(x)$ in $\mathcal{U}$, for each $x : A$.

In the notation of multisets, the elementhood relation $\in : \mathbb{M}_{\mathcal{U}} \to \mathbb{M}_{\mathcal{U}} \to \mathcal{U}^+$ is defined by

$$(X \in \{g(y) \mid y : B\}) \doteq \sum_{(y:B)} g(y) = X.$$

In other words, a multiset X is in a multiset of the form $\{g(y) \mid y : B\}$ if and only if X comes equipped with an element $y : B$ and an identification $g(y) = X$. The W-type of multisets is extensional by Theorem B.4.4 and the univalence axiom.

Recall that for a universe $\mathcal{V}$, we say that a type A is essentially $\mathcal{V}$-small if A comes equipped with an element of type

$$\mathsf{is\text{-}small}_{\mathcal{V}}(A) :\equiv \sum_{(X:\mathcal{V})} A \simeq X.$$

Our goal in this section is to show, via Russell's paradox, that the universe $\mathcal{U}$ is not essentially $\mathcal{U}$-small, i.e., that there cannot be a type $U : \mathcal{U}$ equipped with an equivalence $\mathcal{U} \simeq U$. We will use a similar condition of smallness for multisets.

Definition B.5.2 Let $\mathcal{U}$ and $\mathcal{V}$ be universes. We say that a multiset $\{f(x) \mid x : A\}$ in $\mathcal{U}$ **is $\mathcal{V}$-small** if the type A is $\mathcal{V}$-small and if each mulitset $f(x)$ in $\mathcal{U}$ is $\mathcal{V}$-small. In other words, the type family

$$\mathsf{is\text{-}small\text{-}M}_{\mathcal{V}} : \mathbb{M}_{\mathcal{U}} \to \mathcal{U} \sqcup \mathcal{V}^+$$

is defined recursively by

$$\mathsf{is\text{-}small\text{-}M}_{\mathcal{V}}(\{f(x) \mid x : A\}) :\equiv \mathsf{is\text{-}small}_{\mathcal{V}}(A) \times \prod_{(x:A)} \mathsf{is\text{-}small\text{-}M}_{\mathcal{V}}(f(x)).$$

We will need quite a few properties of smallness before we can reproduce Russell's paradox. We begin with a simple lemma.

Lemma B.5.3 *Consider a $\mathcal{V}$-small multiset $\{f(x) \mid x : A\}$, and let B be a family of essentially $\mathcal{V}$-small types over A. Then the multiset*

$$\{f(x) \mid x : A, y : B(x)\}$$

is again $\mathcal{V}$-small.

Proof If the multiset $\{f(x) \mid x : A\}$ is $\mathcal{V}$-small, then the type A is essentially $\mathcal{V}$-small. By the assumption that B is a family of $\mathcal{V}$-small types together with

the fact that essentially $\mathcal{V}$-small types are closed under formation of Σ-types, we obtain that the type

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

is essentially $\mathcal{V}$-small. Furthermore, since each $f(x)$ is $\mathcal{V}$-small, it follows that the multiset $\{f(x) \mid x : A, y : B(x)\}$ is $\mathcal{V}$-small. □

The main purpose of the following lemma is to know that the elementhood relation takes values in the essentially $\mathcal{V}$-small types, when it is applied to $\mathcal{V}$-small multisets. We will use the univalence axiom to prove this fact.

Proposition B.5.4 *Consider two univalent universes $\mathcal{U}$ and $\mathcal{V}$, and let X and Y be $\mathcal{V}$-small multisets in $\mathcal{U}$. We make two claims:*

(i) *The type $X = Y$ is essentially $\mathcal{V}$-small.*
(ii) *The type $X \in Y$ is essentially $\mathcal{V}$-small.*

Proof For the first claim, let $X \doteq \{f(x) \mid x : A\}$ and let $Y \doteq \{g(y) \mid y : B\}$. The proof is by induction. Via Theorem B.2.3 it follows that the type $X = Y$ is equivalent to the type

$$\sum_{(p:A=B)} \prod_{(x:A)} f(x) = g(\mathsf{equiv\text{-}eq}(p)).$$

The type $A = B$ is $\mathcal{V}$-small because it is equivalent to the type $A \simeq B$, which is essentially $\mathcal{V}$-small. Therefore it suffices to show that the type

$$\prod_{(x:A)} f(x) = g(\mathsf{equiv\text{-}eq}(p))$$

is essentially $\mathcal{V}$-small, for every $p : A = B$. Here we proceed by path induction, and the type $\prod x : A f(x) = g(x)$ is a product of essentially $\mathcal{V}$-small types by the induction hypothesis. This concludes the proof of the first claim.

For the second claim, let $Y \doteq \{g(y) \mid y : B\}$. Then the type

$$\sum_{(y:B)} g(y) = X$$

is a dependent sum of essentially $\mathcal{V}$-small types, indexed by an essentially $\mathcal{V}$-small type, which is again essentially $\mathcal{V}$-small. □

The condition that a multiset $\{f(x) \mid x : A\}$ in $\mathcal{U}$ is $\mathcal{V}$-small suggests that there is an 'equivalent' multiset in $\mathcal{V}$.

Definition B.5.5 Given two universes $\mathcal{U}$ and $\mathcal{V}$, we define an inclusion function

$$i : \left(\sum_{(X:\mathbb{M}_{\mathcal{U}})} \mathsf{is\text{-}small\text{-}M}_{\mathcal{V}}(X)\right) \to \mathbb{M}_{\mathcal{V}},$$

of the $\mathcal{V}$-small multisets in $\mathcal{U}$ into the multisets in $\mathcal{V}$, inductively by

$$i(\{f(x) \mid x : A\}) :\equiv \{i(f(e^{-1}(y))) \mid y : B\}.$$

for any multiset $\{f(x) \mid x : A\}$ of which the type A is equipped with an equivalence $e : A \simeq B$ for some B in $\mathcal{V}$, and such that the multiset $f(x)$ in $\mathcal{U}$ is $\mathcal{V}$-small for each $x : A$.

Proposition B.5.6 *The inclusion function i of $\mathcal{V}$-small multisets in $\mathcal{U}$ into the multisets in $\mathcal{V}$ satisfies the following properties*

(i) *For each $\mathcal{V}$-small multiset X in $\mathcal{U}$, the multiset $i(X)$ in $\mathcal{V}$ is $\mathcal{U}$-small.*
(ii) *The induced map*

$$\left(\textstyle\sum_{(X:\mathbb{M}_{\mathcal{U}})} \mathsf{is\text{-}small\text{-}M}_{\mathcal{V}}(X)\right) \to \left(\textstyle\sum_{(Y:\mathbb{M}_{\mathcal{V}})} \mathsf{is\text{-}small\text{-}M}_{\mathcal{U}}(Y)\right)$$

is an equivalence.

Consequently, the inclusion function i is an embedding.

Proof To see that $i(\{f(x) \mid x : A\})$ is $\mathcal{U}$-small for each $\mathcal{V}$-small multiset $\{f(x) \mid x : A\}$ in $\mathcal{U}$, note that the assumption that $\{f(x) \mid x : A\}$ is $\mathcal{V}$-small gives us an equivalence $e : A \simeq B$ and an element $H(x) : \mathsf{is\text{-}small\text{-}M}_{\mathcal{V}}(f(x))$ for each $x : A$. The type B is the indexing type of $i(\{f(x) \mid x : A\})$, and B is $\mathcal{U}$-small because it is equivalent to the type A in $\mathcal{U}$. Furthermore, each multiset $i(f(e^{-1}(y)))$ is $\mathcal{U}$-small by the inductive hypothesis. This completes the proof of the first claim.

We therefore have inclusion functions

$$\left(\textstyle\sum_{(X:\mathbb{M}_{\mathcal{U}})} \mathsf{is\text{-}small\text{-}M}_{\mathcal{V}}(X)\right) \underset{i}{\overset{i}{\leftrightarrows}} \left(\textstyle\sum_{(Y:\mathbb{M}_{\mathcal{V}})} \mathsf{is\text{-}small\text{-}M}_{\mathcal{U}}(Y)\right)$$

To see that the maps i and i are mutual inverses, it suffices to show that $i(i(X)) = X$. This follows by induction from the following calculation, where we assume an equivalence $e : A \simeq B$ into a $\mathcal{V}$-small type B.

$$\begin{aligned} i(i(\{f(x) \mid x : A\})) &\doteq i(\{i(f(e^{-1}(y))) \mid y : B\}) \\ &\doteq \{i(i(f(e^{-1}(e(x))))) \mid x : A\} \\ &= \{i(i(f(x))) \mid x : A\} \\ &= \{f(x) \mid x : A\}. \end{aligned}$$

For the last claim, note that we have factored i as an equivalence followed by an embedding

$$\left(\textstyle\sum_{(X:\mathbb{M}_{\mathcal{U}})} \mathsf{is\text{-}small\text{-}M}_{\mathcal{V}}(X)\right) \longrightarrow \left(\textstyle\sum_{(Y:\mathbb{M}_{\mathcal{V}})} \mathsf{is\text{-}small\text{-}M}_{\mathcal{U}}(Y)\right) \longrightarrow \mathbb{M}_{\mathcal{V}},$$

and therefore i is an embedding. □

Furthermore, the embedding i induces equivalences on the elementhood relation on multisets.

Proposition B.5.7 *Consider a multiset X in $\mathcal{U}$ and a multiset Y in $\mathcal{V}$. Furthermore, suppose that X is $\mathcal{V}$-small and that Y is $\mathcal{U}$-small. Then we have*

$$(i(X) \in Y) \simeq (X \in i(Y)).$$

Proof Let $X \doteq \{f(x) \mid x : A\}$ and $Y \doteq \{g(y) \mid y : B\}$. By the assumption that Y is $\mathcal{U}$-small we have an equivalence $e : B \simeq B'$ to a type B' in $\mathcal{U}$. Then we have the equivalences

$$\begin{aligned} i(X) \in \{g(y) \mid y : B\} &\doteq \textstyle\sum_{(y:B)} g(y) = i(X) \\ &\simeq \textstyle\sum_{(y:B)} i(g(y)) = X \\ &\simeq \textstyle\sum_{(y':B')} i(g(e^{-1}(y'))) = X \\ &\doteq X \in i(Y). \end{aligned}$$

□

We are now almost in position to reproduce Russell's paradox. We will need one mre ingredient: the universal tree, i.e., the multiset of all multisets in $\mathcal{U}$.

Definition B.5.8 Let $\mathcal{U}$ be a universe. Then we define the **universal tree** $\mathbb{Y}_{\mathcal{U}}$ to be the multiset

$$\mathbb{Y}_{\mathcal{U}} :\equiv \{i(X) \mid X : \mathbb{M}_{\mathcal{U}}\}$$

in $\mathcal{U}^+$, where $i : \mathbb{M}_{\mathcal{U}} \to \mathbb{M}_{\mathcal{U}^+}$ is the inclusion of the multisets in $\mathcal{U}$ to the multisets in $\mathcal{U}^+$ given by the fact that each multiset in $\mathcal{U}$ is $\mathcal{U}^+$-small.

Proposition B.5.9 *Consider two universes $\mathcal{U}$ and $\mathcal{V}$, and suppose that $\mathcal{U}$ as well as each $X : \mathcal{U}$ are essentially $\mathcal{V}$-small. Then the universal tree $\mathbb{Y}_{\mathcal{U}}$ is also $\mathcal{V}$-small.*

Proof To show that the universal tree $\{i(X) \mid X : \mathbb{M}_{\mathcal{U}}\}$ is $\mathcal{V}$-small, we first have to show that the type $\mathbb{M}_{\mathcal{U}}$ is $\mathcal{V}$-small. This follows from the more general fact that the subuniverse of $\mathcal{V}$-small types is closed under the formation of W-types. Indeed, if a type A is $\mathcal{V}$-small, and if $B(x)$ is $\mathcal{V}$-small for each $x : A$, then we have an equivalence $\alpha : A \simeq A'$ to a type A' in $\mathcal{V}$, and for each $x' : A'$ we have an equivalence $B(\alpha^{-1}(x')) \simeq B'(x')$ in $\mathcal{V}$. These equivalences induce an equivalence

$$\mathsf{W}(A, B) \simeq \mathsf{W}(A', B')$$

into the type $W(A', B')$, which is in $\mathcal{V}$. This concludes the proof that $\mathbb{M}_{\mathcal{U}}$ is essentially $\mathcal{V}$-small.

It remains to show that the multiset $i(X)$ in $\mathcal{U}^+$ is $\mathcal{V}$-small, for each $X : \mathbb{M}_{\mathcal{U}}$. Equivalently, we have to show that each multiset X in $\mathcal{U}$ is $\mathcal{V}$-small. This follows by recursion: given a multiset $\{f(x) \mid x : A\}$, the type A is essentially $\mathcal{V}$-small by assumption, and the multiset $f(x)$ is $\mathcal{V}$-small by the induction hypothesis. □

Theorem B.5.10 *Consider a univalent universe $\mathcal{U}$. Then $\mathcal{U}$ cannot be essentially $\mathcal{U}$-small.*

Proof Suppose that $\mathcal{U}$ is $\mathcal{U}$-small, and consider the multiset

$$R :\equiv \{i(X) \mid X : \mathbb{M}_{\mathcal{U}}, H : X \notin X\}$$

in $\mathcal{U}^+$, where $i : \mathbb{M}_{\mathcal{U}} \to \mathbb{M}_{\mathcal{U}^+}$ is the inclusion of the multisets in $\mathcal{U}$ to the multisets in $\mathcal{U}^+$ given by the fact that each multiset in $\mathcal{U}$ is $\mathcal{U}^+$-small.

First, we note that R is $\mathcal{U}$-small. This follows from Lemma B.5.3, using the fact that the universal tree $\{i(X) \mid X : \mathbb{M}_{\mathcal{U}}\}$ is $\mathcal{U}$-small by Proposition B.5.9, and the fact that $X \in X$ is essentially $\mathcal{U}$-small by Proposition B.5.4.

Since R is $\mathcal{U}$-small, there is a multiset $R' : \mathbb{M}_{\mathcal{U}}$ such that $i(R') = R$. Now it follows that

$$\begin{aligned} R \in R &\simeq \textstyle\sum_{(X:\mathbb{M}_{\mathcal{U}})} \sum_{(H:X\notin X)} i(X) = R \\ &\simeq \textstyle\sum_{(X:\mathbb{M}_{\mathcal{U}})} \sum_{(H:X\notin X)} X = R' \\ &\simeq R' \notin R' \\ &\simeq R \notin R. \end{aligned}$$

In the second step we used Proposition B.5.6, where we showed that i is an embedding, and in the last step we used Proposition B.5.7. Now we obtain a contradiction, because it follows from Exercise 4.3 (a) that no type is (logically) equivalent to its own negation. □

Exercises

B.1 For a type family B over A, suppose that each $B(x)$ is empty. Show that the type $\mathsf{W}(A, B)$ is equivalent to the type A.

B.2 (a) Show that the elementhood relation $\in$ on $\mathsf{W}(A, B)$ is irreflexive, for any type family B over any type A.

(b) Use the previous fact along with Proposition B.5.7 to give a second proof of the fact that there can be no type $U : \mathcal{U}$ equipped with an equivalence $\mathcal{U} \simeq U$.

B.3 For each $x : \mathsf{W}(A, B)$, let $x < (-) : \mathsf{W}(A, B) \to \mathcal{U}$ be the type family generated inductively by the following constructors:

$$i : \prod_{(y:\mathsf{W}(A,B))} (x \in y) \to (x < y)$$
$$j : \prod_{(y,z:\mathsf{W}(A,B))} (y \in z) \to ((x < y) \to (x < z)).$$

(a) Show that the type-valued relation $<$ is transitive and irreflexive.
(b) Suppose that the type $\mathsf{W}(A, B)$ is inhabited. Show that the following are equivalent:
   (i) The type $x < y$ is a proposition for all $x, y : \mathsf{W}(A, B)$.
   (ii) The type A is a set and the type $B(a)$ is a proposition for all $a : A$.

   Thus, in general it is not the case that $<$ is a relation valued in propositions.
(c) Show that $\mathsf{W}(A, B)$ satisfies the following **strong induction principle**: For any type family P over $\mathsf{W}(A, B)$, if there is a function

$$h : \prod_{(x:\mathsf{W}(A,B))} \left(\prod_{(y:\mathsf{W}(A,B))} (y < x) \to P(y) \right) \to P(x),$$

   then there is a function $f : \prod_{(x:\mathsf{W}(A,B))} P(x)$ equipped with an identification

$$f(x) = h(x, \lambda y.\, \lambda p.\, f(y))$$

   for all $x : \mathsf{W}(A, B)$.
(d) Show that there can be no sequence of elements $x : \mathbb{N} \to \mathsf{W}(A, B)$ such that $x_{n+1} < x_n$ for all $n : \mathbb{N}$.

B.4 For a type family $B : A \to \mathcal{U}$, define a new dependent type $\hat{B}$ over $\hat{A}$, where

$$\hat{A} :\equiv \mathrm{im}(B)$$
$$\hat{B}(X) :\equiv X.$$

The type family $\hat{B}$ over $\hat{A}$ is called the **univalent completion** of the type family B over A, and the W-type $\mathsf{W}(\hat{A}, \hat{B})$ is called the **extensional completion** of the W-type $\mathsf{W}(A, B)$.

(a) Show that the W-type $\mathsf{W}(\hat{A}, \hat{B})$ is extensional.
(b) Show that there is a map $\eta : \mathsf{W}(A, B) \to \mathsf{W}(\hat{A}, \hat{B})$ equipped with a family of equivalences

$$(x \in y) \simeq (\eta(x) \in \eta(y)),$$

indexed by $x, y : \mathsf{W}(A,B)$. Conclude that

$$(\eta(x) = \eta(y)) \simeq \textstyle\prod_{(z:\mathsf{W}(A,B))} (z \in x) \simeq (z \in y).$$

for every $x, y : \mathsf{W}(A,B)$.

(c) Consider the map $\sigma : \mathsf{W}(A,B) \to (\mathsf{W}(A,B) \to \mathcal{U})$ given by

$$\sigma(x,y) :\equiv (y \in x).$$

Construct a commuting triangle

$$\begin{array}{ccc} & \mathsf{W}(A,B) & \\ {\scriptstyle\eta}\swarrow & & \searrow{\scriptstyle q} \\ \mathsf{W}(\hat{A},\hat{B}) & \xrightarrow[e]{} & \mathrm{im}(\sigma), \end{array}$$

in which the map e is an equivalence.

B.5 Consider the relation $\preceq : \mathsf{W}(A,B) \to (\mathsf{W}(A,B) \to \mathsf{Prop}_{\mathcal{U}})$ defined recursively by

$$(\mathsf{tree}(a,\alpha) \preceq \mathsf{tree}(b,\beta)) :\equiv \forall_{(x:B(a))} \exists_{(y:B(b))} \alpha(x) \preceq \beta(y).$$

If $x \preceq y$ holds, we say that x has **lower rank** than y. Furthermore, we define the relaction $\prec$ on $\mathsf{W}(A,B)$ by

$$(x \prec y) :\equiv \exists_{(z \in y)} x \preceq z.$$

If $x \prec y$ holds, we say that x has **strictly lower rank** than y.

(a) Show that $\preceq$ defines a preordering on $\mathsf{W}(A,B)$, i.e., show that $\preceq$ is reflexive and transitve. Furthermore, prove the following three properties, in which $<$ is the strict ordering on $\mathsf{W}(A,B)$ defined in Exercise B.3:

(i) $(x \preceq y) \leftrightarrow \forall_{(x'<x)} \exists_{(y'<y)} x' \preceq y'$
(ii) $(x < y) \to (x \preceq y)$
(iii) $(x < y) \to (y \not\preceq x)$
(iv) $\mathsf{is\text{-}constant}_{\mathsf{W}}(x) \leftrightarrow \forall_{(y:\mathsf{W}(A,B))} x \preceq y.$

(b) Show that the relation $\prec$ on $\mathsf{W}(A,B)$ is a strict ordering on $\mathsf{W}(A,B)$, i.e., show that it is irreflexive and transitive.

Since $\preceq$ defines a preordering on $\mathsf{W}(A,B)$, it follows that the preorder $(\mathsf{W}(A,B), \preceq)$ has a poset reflection. We will write

$$\eta : (\mathsf{W}(A,B), \preceq) \to (\mathcal{R}(A,B), \preceq)$$

for the poset reflection of $(\mathsf{W}(A,B), \preceq)$ and its quotient map. We will call the poset $(\mathcal{R}(A,B), \preceq)$ the **poset rank** of the W-type $\mathsf{W}(A,B)$.

(c) Show that if A and each $B(x)$ are finite, then the poset rank $(\mathcal{R}(A,B),\preceq)$ is either the empty poset, the poset with one element, or it is isomorphic to the poset $(\mathbb{N},\leq)$.

(d) Show that the strict ordering $\prec$ extends to a relation $\prec$ on $\mathcal{R}(A,B)$ with the following properties:

(i) We have $(x \prec y) \leftrightarrow (\eta(x) \prec \eta(y))$ for every $x, y : \mathsf{W}(A,B)$.

(ii) We have $(x \prec y) \to (x \preceq y)$ for every $x, y : \mathcal{R}(A,B)$.

(iii) The relation $\prec$ is transitive and irreflexive on $\mathcal{R}(A,B)$.

We will call the strictly ordered set $(\mathcal{R}(A,B),\prec)$ the **(strict) rank** of the W-type $\mathsf{W}(A,B)$.

(e) A strictly ordered set $(X,<)$, i.e., a set X equipped with a transitive, irreflexive relation $<$ valued in the propositions, is said to be **well-founded** if for any family P of propositions over X, the implication

$$\left(\forall_{(x:X)}\left(\forall_{(y<x)}P(y)\right)\to P(x)\right)\to\forall_{(x:X)}P(x).$$

holds. Show that the rank $(\mathcal{R}(A,B),\prec)$ of $\mathsf{W}(A,B)$ is well-founded.

(f) A strictly ordered set $(X,<)$ is said to be **extensional** if the logical equivalence

$$(x=y)\leftrightarrow\forall_{(z:X)}\,(z<x)\leftrightarrow(z<y)$$

holds for any $x, y : X$. Show that the rank $(\mathcal{R}(A,B),\prec)$ of $\mathsf{W}(A,B)$ is extensional.

(g) (To do, probably requires choice) A strictly ordered set $(X,<)$ is said to be **regular** if for any $x : X$, any map

$$f:\left(\textstyle\sum_{(y:X)}y<x\right)\to X$$

has an upper bound, in the sense that

$$\exists_{(b:X)}\forall_{(y<x)}f(y)<b.$$

Show that the rank $(\mathcal{R}(A,B),\prec)$ of $\mathsf{W}(A,B)$ is **regular**.

(h) (To do) Let $(X,<)$ be a well-founded, extensional, and regular strictly ordered set. Show that $(X,<)$ is isomorphic to $(\mathcal{R}(X,B),\prec)$, where the family B over X is given by

$$B(x)\coloneqq\textstyle\sum_{(y:X)}y<x.$$

Thus, we have completely characterized the strictly ordered sets in the image of the operation $\mathcal{R}$.

B.6 (Awodey, Gambino, Sojakova) For any type family B over A, the **polynomial endofunctor** $P_{A,B}$ acts on types by

$$P_{A,B}(X) :\equiv \textstyle\sum_{(x:A)} X^{B(x)},$$

and it takes a map $h : X \to Y$ to the map

$$P_{A,B}(h) : P_{A,B}(X) \to P_{A,B}(Y)$$

defined by $P_{A,B}(h, (x, \alpha)) :\equiv (x, h \circ \alpha)$. Furthermore, there is a canonical map

$$(h \sim h') \to (P_{A,B}(h) \sim P_{A,B}(h'))$$

taking a homotopy $H : h \sim h'$ to a homotopy $P_{A,B}(H) : P_{A,B}(h) \sim P_{A,B}(h')$.

A type X is said to be equipped with the **structure of an algebra** for the polynomial endofunctor $P_{A,B}$ if X comes equipped with a map

$$\mu : P_{A,B}(X) \to X.$$

Thus, **algebras** for the polynomial endofunctor $P_{A,B}$ are pairs (X, μ) where X is a type and $\mu : P_{A,B}(X) \to X$. Note that $\mathsf{W}(A, B)$ comes equipped with the structure of an algebra for $P_{A,B}$ by Proposition B.2.1.

Given two algebras X and Y for the polynomial endofunctor $P_{A,B}$, we say that a map $h : X \to Y$ is equipped with the **structure of a homomorphism** of algebras if it comes equipped with a homotopy witnessing that the square

$$\begin{array}{ccc} P_{A,B}(X) & \xrightarrow{P_{A,B}(h)} & P_{A,B}(Y) \\ {\scriptstyle \mu_X}\downarrow & & \downarrow{\scriptstyle \mu_Y} \\ X & \xrightarrow[h]{} & Y \end{array}$$

commutes. The type $\mathsf{hom}((X, \mu_X), (Y, \mu_Y))$ of homomorphisms of algebras for $P_{A,B}$ is therefore defined as

$$\mathsf{hom}((X, \mu_X), (Y, \mu_Y)) :\equiv \textstyle\sum_{(h:X\to Y)} h \circ \mu_X \sim \mu_Y \circ P_{A,B}(h).$$

(a) For any $(x, \alpha), (y, \beta) : P_{A,B}(X)$, construct an equivalence

$$((x, \alpha) = (y, \beta)) \simeq \textstyle\sum_{(p:x=y)} \alpha \sim \beta \circ \mathsf{tr}_B(p).$$

(b) For any two morphisms $(f, K), (g, L) : \mathsf{hom}((X, \mu_X), (Y, \mu_Y))$ of algebras for $P_{A,B}$, construct an equivalence

$$((f, K) = (g, L)) \simeq \textstyle\sum_{(H:f\sim g)} K \cdot (\mu_Y \cdot P_{A,B}(H)) \sim (H \cdot \mu_X) \cdot L.$$

(c) Show that the W-type $\mathsf{W}(A, B)$ equipped with the canonical structure ε of a $P_{A,B}$-algebra, constructed in Proposition B.2.1, is a **homotopy initial $P_{A,B}$-algebra** in the sense that the type

$$\mathsf{hom}((\mathsf{W}(A, B), \varepsilon), (X, \mu))$$

is contractible, for each $P_{A,B}$-algebra (X, μ).

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

Index

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021

Bibliography

[1] J. F. Adams. “On the nonexistence of elements of Hopf invariant one”. In: *Bull. Amer. Math. Soc.* 64.5 (Sept. 1958), pp. 279–282.

[2] Erret Bishop. *Foundations of constructive analysis*. New York: McGraw-Hill Book Co., 1967, pp. xiii+370.

[3] C.F. Gauss, W.C. Waterhouse, and A.A. Clarke. *Disquisitiones Arithmeticae*. Springer-Verlag, 1986. ISBN: 9783540962540.

[4] G. Peano. *Arithmetices principia: nova methodo exposita*. Fratres Bocca, 1889.

[5] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.

The best way to support this book while it is still being prepared for publication, is to let the author know if you find inaccuracies of any kind at egbert.rijke@fmf.uni-lj.si, or at the homotopy type theory chat at https://hott.zulipchat.com.

There are 306 exercises.

Ljubljana, November 10, 2021