# THE WEDGE SUM AND THE SMASH PRODUCT IN HOMOTOPY TYPE THEORY

**MASTER THESIS**
**MATHEMATICAL INSTITUTE**
**LMU MUNICH**

ANDREAS FRANZ
THESIS SUPERVISOR: DR. IOSIF PETRAKIS

Abstract. Martin-Löf's intensional type theory (ITT) can be extended with higher inductive types (HITs), which generalize inductive types, since, in addition to point constructors, constructors that output (higher) paths are allowed. We focus on a specific class of HITs, the so-called pushout types, which define important types from the geometric point of view. In order to show that pushouts are invariant under homotopy, and thus feasible in the framework of homotopy type theory, we develop a tool-set to induce the right morphisms between them. We elaborate two fundamental examples of pushout types, the wedge sum and the smash product of pointed types. Using our tool-set we show that these two pushouts correspond to the coproduct and tensor product, respectively, in the subcategory of pointed types and base-point preserving functions.

## Contents



*Date*: August 25, 2017.

## 1. Introduction

In this thesis we study an extension of intensional type theory (ITT) by higher inductive types (HITs). ITT was originally proposed by Per Martin Löf in the 1970s [9] as a formal system for doing constructive mathematics, e.g. in the sense of Bishop [2]. In section 2 we give an elementary introduction to this theory. In particular, we include several standard examples of inductive types. Roughly speaking, an inductive type is defined by giving its constructors (constructors can be seen as functions that generate terms of the type being defined) and specifying an induction principle, which expresses that the inductive type is "freely generated" by this constructors. The identity type of two terms $x, y$ of type $A$, written $x =_A y$, is an example of an inductive type that will be of particular interest to us. It has just one constructor $\mathsf{refl}_a : a =_A a$ expressing that every term is equal to itself. Similar to sets, types can have *inhabitants* and one might think of identity types $x =_A y$ as sets with at most one element, depending on weather $x$ equals $y$ or not. This idea that any two inhabitants of some identity type are equal is also called the principle of *uniqueness of identity proofs* (UIP) and ITT becomes *extensional*,

when we add such a principle. This approach has some disadvantages, in particular type checking is not algorithmically decidable, whereas in ITT it is. The first hint that it is reasonable to refute UIP was provided by Hofmann and Streicher [7], who gave a model of ITT in which the interpretation of identity types has indeed multiple elements. The novel point of view taken in homotopy type theory is to think of types $A$ not as sets, but as *spaces* and in section 3 we present basic results that support this nomenclature. Under the *types as spaces* interpretation the terms $a : A$ become *points* and the inhabitants of the identity types $x =_A y$ become *paths* between $x$ and $y$ in the space $A$. What is more, this interpretation can be extended in a natural way when forming identity types recursively, i.e. from $x, y : A$ we can form the type $x =_A y$ of identifications or *paths* between $x$ and $y$, from $p, q : x =_A y$ we can form the type $p =_{x=_A y} q$ of identifications between those identifications or *homotopies* and so on. Under the types as spaces interpretation it is not desirable anymore to assume that identity types are inhabited by a single term, since there may be several paths between two points that are not homotopic to each other. In section 4 we present pushout-types, a specific class of higher inductive types (HITs) that generalize inductive types, since constructors that generate identifications (that may now be unequal to reflexivity) are allowed. Pushout is a type former, where from three types $A, B$ and $C$ together with two functions $f : C \to A$ and $g : C \to B$ we may form the pushout-type $A \sqcup^C B$. The reason we restrict ourselves to pushout-types is that, at the present moment, a general theory of higher inductive types is still missing. Still, pushout-types provide a general enough framework to formally represent and thus reason about a large class of spaces in type theory that have non trivial homotopy, such as the circle or the interval, without having to consider point set topology first. Additionally they are interesting also from a purely type theoretic point of view. As an example we include the proof of function extensionality (see lemma 4.6). In section 4.3 we show, that the pushout of types is invariant under homotopy in the sense that if we have equivalences $\alpha : A \to A'$, $\beta : B \to B'$ and $\gamma^{-1} : C' \to C$, the pushouts $A \sqcup^C B$ of $(A, B, C, f, g)$ and $A' \sqcup^{C'} B'$ of $(A', B', C', \alpha \circ f \circ \gamma^{-1}, \beta \circ g \circ \gamma^{-1})$ are equivalent. To this end we develop a tool-set to induce the right morphisms between pushout types. In the last two sections of this work we restrict our

attention to pointed types and base-point preserving maps, i.e. every type $A$ comes together with a specified base-point $*_A : A$ and every map $f : A \to B$ comes together with a proof $p_f : *_B =_B f(*_A)$. We present several examples of important pushouts in this setting, in particular the appropriate notions for a (binary) coproduct called "wedge sum" and a tensor product, called the "smash product" of pointed types. We note that related work has been done by Guillaume Brunerie [3], Evan Cavallo [5] and Robert Graham [6] and parts of their work have been formalized in Agda (see [4]).

**Acknowledgements.** I want to thank my thesis supervisor Dr. Iosif Petrakis, as well as Dr. Chuangjie Xu, for their guidance and support throughout this project. They suggested countless improvements to this work and greatly helped me to develop my understanding of homotopy type theory. Without the many hours they took to read and discuss my ideas, I would not have come as far. I also want to express my appreciation to Prof. Schuster, who gave me the possibility to work on homotopy type theory during a research internship at the university of Verona and invited me to a conference, where I presented some of my results. None of this would have been possible without my supervisor Dr. Petrakis, who introduced us and could not have been more encouraging to me.

## 2. Intensional type theory

Before we present the basic types of ITT, we briefly discuss some of the underlying ideas of the theory. We refer to chapter one of the HoTT book [10] for a more detailed description. In ITT there is a primitive notion for *collection*, called a *type* and we call the *elements* of types *terms*, *inhabitants* or *points*. Membership of a term $a$ to a type $A$, written $a : A$, and judgmental equality between two terms $a : A$ and $a' : A$, written $a \equiv a'$, are the two basic judgments of type theory. We note that judgments bind more loosely than anything else. The basic rules of ITT stipulate certain constructions on types and terms that we are allowed to perform. We will make use of a universe type, denoted by $\mathcal{U}$, whose inhabitants are itself types. For consistency reasons $\mathcal{U}$ can not inhabit itself, thus formally we have to work inside an infinite hierarchy of universes $\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : ...$, but since we will never leave the scope of a particular universe, we will just omit the subscript. It is important to

understand that the rules of a specific type theory are not formulated inside some meta-theory, but instead have to be specified separately, whenever we introduce a new type, thus ITT is really a collection of concrete types, each coming with its own set of rules. The general pattern to define a new type is as follows. We need to give:

(i) A formation rule, explaining how to form new types, that is, how to infer judgments of the form $A : \mathcal{U}$ that $A$ is a type. We can for example introduce the type of functions $A \to B : \mathcal{U}$ if we are given two types $A : \mathcal{U}$ and $B : \mathcal{U}$.

(ii) Introduction rules, explaining how to introduce new terms of a type. The introduction rules for the type of natural numbers $\mathbb{N}$ say that $0 : \mathbb{N}$ , i.e. zero is a natural number and, given some number $n : \mathbb{N}$, also its successor is a natural number, that is, $\mathsf{succ}(n) : \mathbb{N}$.

(iii) Elimination rules, also called induction principles, explaining how to use inhabitants of a type. For example, the elimination rule for function types expresses that we can apply a function $f : A \to B$ to an inhabitant $a$ of $A$ in the domain of the function and this yields an inhabitant $f(a)$ of $B$ in the codomain.

(iv) Computation rules, explaining the procedural behavior of eliminators acting on points. For instance, if we are given some $b : B$, we can introduce the constant function $b_A : A \to B$ with the obvious computation rule $b_A(a) \equiv b$.

(v) Optionally a uniqueness principle for maps into or out of the type.

We can interpret first order logic directly in ITT by assigning the quantifiers and logical connectives appropriate formation rules in such a way that the introduction and elimination rules correspond precisely to the rules of logical inference. In other words, propositions are special types and proving a proposition means to construct an inhabitant of the corresponding type (we may even speak of a proof-term in this situation). For instance, logical implication is naturally interpreted in function types and proving an implication is identified with the act of constructing a function of appropriate type. This is also called the *propositions as types interpretation* or *Curry-Howard isomorphism* (see [8] for Howard's original paper). To give a complete presentation of this

interpretation, but also for convenience reasons, we introduce types for *each* of the logical connectives, even though some could be introduced as special cases of others.

Lastly we note that in the setting of type theory, terms can not be introduced in isolation of their type, but the type of a term is a part of its intrinsic nature. Consequently it does not make sense to ask if a term $a$ has type $A$. This sort of question would be sensible in the setting of set theory, where $a \in A$ is a proposition which has to be proven inside the superstructure of first order logic. Under the *propositions as types* interpretation $a : A$ is not a proposition but is expressing the judgment that $a$ is a proof of the proposition $A$.

2.1. **Function types.** Given two types $A$ and $B$, we may form the type of functions $A \to B$. When forming function types, we associate to the right, i.e. $A \to B \to C$ is to be read as $A \to (B \to C)$. The most basic way to construct elements of a function type is by $\lambda$-abstraction. Given some term $\Phi : B$, possibly involving a free occurrence of the variable $x : A$, we can set:

$$\lambda x.\Phi : A \to B$$

The computation rule, also called $\beta$-reduction, for this function says that in order to evaluate it at some given point $a : A$, we need to substitute $a$ for $x$ in the expression $\Phi$. We write this as:

$$(\lambda x.\Phi)(a) :\equiv \Phi[x/a]$$

If not specified otherwise by using brackets, the scope of a $\lambda$-abstraction is the entire expression after "$\lambda x.$". Alternatively, we can name a function by giving a defining equation. For instance, in the above context, we can introduce a function $f : A \to B$ by

$$f(x) :\equiv \Phi,$$

which evaluates for a given $a : A$ as:

$$f(a) :\equiv \Phi[x/a]$$

Of course it should not matter, if we define a function by $\lambda-$abstraction or by giving it a name and therefore we add a rule:

$$f \equiv \lambda x.f(x)$$

This rule is also called $\eta-$conversion and it expresses that functions are determined by their values, i.e. it is a uniqueness principle for functions. Under the propositions as types interpretation functions correspond to proofs of implications. We note that indeed we can read the introduction and elimination rules as "given that $B$ is true, then $A \to B$ is true" and "if $A \to B$ is true and $A$ is true, then $B$ is true".

*Remark* 2.1. Sometimes we will also use the notation $x \mapsto \Phi$ instead of $\lambda x.\Phi$, to denote a function of type $A \to B$, where $x : A$ and the expression $\Phi : B$ possibly involves a free occurrence of the variable $x$.

2.2. **Dependent function types.** Dependent functions are a generalization of ordinary functions for which the type of the codomain may vary depending on points in the domain. We model this by using type families $P$ over $A$, which are inhabitants of the type $A \to \mathcal{U}$, that is, $P$ represents a function that gives a type $P(a) : \mathcal{U}$ for every given $a : A$. Given a type $A : \mathcal{U}$ and a type family $P : A \to \mathcal{U}$ we can form the type of dependent functions $\prod_{(x:A)} P(x)$. If not specified otherwise by using brackets, the scope of such an expression is everything after "$\prod_{(x:A)}$". Assume we are given some term $\Phi : P(x)$, possibly involving a free occurrence of the variable $x$. As in the non-dependent case, to introduce a dependent function, we can use $\lambda$-abstraction

$$\lambda x.\Phi : \prod_{x:A} P(x)$$

or specify a name $\mathsf{F}$ by giving a defining equation

$$\mathsf{F}(x) :\equiv \Phi.$$

As above we formulate a uniqueness principle

$$\mathsf{F} \equiv \lambda x.\mathsf{F}(x)$$

and an obvious computation rule

$$\mathsf{F}(a) :\equiv \Phi[x/a].$$

Under the proposition as types interpretation dependent functions correspond to universal quantification and applying a dependent function to an inhabitant $a$ of $A$ gives a proof $\mathsf{F}(a) : P(a)$ that $a$ has property $P$. We note that the type of functions $A \to B$ is just a special case, when the type family $P$ is constant $B$. An important class of dependent functions are polymorphic functions, which take as one of their inputs a type. This can be seen as a generic way to define functions. A basic example is given by the class of identity functions

$$\mathsf{id} : \prod_{A:\mathcal{U}} A \to A,$$

which is defined by

$$\lambda A.\lambda x.x.$$

We write $\mathsf{id}_A : A \to A$ for $\mathsf{id}(A)$, the identity function on $A$. Another example are constant functions, where we assume we are given types $A, B : \mathcal{U}$ and some fixed $b : B$. We can then define:

$$\lambda A.\lambda B.\lambda x.b : \prod_{A,B:\mathcal{U}} A \to B$$

We also write $b_A$ for the function $(\lambda A.\lambda B.\lambda x.b)(A)(B)$.

2.3. **Inductive types.** The idea of inductive types is to specify a set of *constructors*, that is, functions whose codomain is allowed to be the type under definition, together with an induction principle stating that in order to show a property for all elements of the inductive type, it suffices to establish the property for all those elements that can be reached by applying the constructors repeatedly. We say that the type is *freely generated* by its constructors. Recall that in ITT, predicates on inhabitants $x$ of $A$ are modeled by type families $P : A \to \mathcal{U}$ and to proof that $P$ holds for all $x : A$ means to construct an inhabitant of $\prod_{(x:A)} P(x)$. We mentioned that if the type family is constant, i.e. $P(x) \equiv B$ for some type $B$, the type $\prod_{(x:A)} P(x)$ is the type $A \to B$ of ordinary functions and then we speak of recursion instead of induction.

**Example 2.2.** A standard example of an inductive type is the type $\mathbb{N}$ of natural numbers. It is generated by two constructors

$$0 : \mathbb{N}$$

and

$$\mathsf{succ} : \mathbb{N}.$$

In order to establish some property $P : \mathbb{N} \to \mathcal{U}$ for all $n : \mathbb{N}$, we need to give a proof

$$p_0 : P(0)$$

that 0 has property $P$ and additionally provide an inhabitant

$$p_s : \prod_{n:\mathbb{N}} P(n) \to P\big(\mathsf{succ}(n)\big)$$

and then, from a proof

$$q : P(n)$$

that $n$ has property $P$, we get a proof

$$p_s(n)(q) : P\big(\mathsf{succ}(n)\big)$$

that $\mathsf{succ}(n)$ has property $P$.

2.3.1. *Identity types.* One example of an inductive type that is particularly important to us is the type of identifications.

**Definition 2.3** (Identity types)**.** For every type $A : \mathcal{U}$ and every two terms $x, y : A$ we can form the type of identifications $x =_A y$. For every $x : A$ we have a canonical proof

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

that $x$ is equal to itself. The induction principle says that if we are given a type family

$$C : \prod_{x,y:A} (x =_A y) \to \mathcal{U}$$

and a function

$$c : \prod_{x:A} C(x, x, \mathsf{refl}_x),$$

then there is a dependent function

$$\mathsf{F} : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p),$$

such that

$$\mathsf{F}(x, x, \mathsf{refl}_x) :\equiv c(x)$$

for every $x : A$.

*Remark* 2.4. Often the type $A : \mathcal{U}$ in $x =_A y$ will be clear from the context and in this case we may also write $x = y$ for the type of identifications between $x$ and $y$.

Let us take a closer look at the induction principle. Put in simpler terms it says that, in order to establish some property $C(x, y, p)$ for $x, y : A$ and $p : x =_A y$, it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$. Therefore, in order to establish a property $C$ for all triplets $(x, y, p)$, it suffices to show it on the triplets of the form $(x, x, \mathsf{refl}_x)$. Using the induction principle, we now present several properties of identity types. We will do the proofs in this section in more detail and later on switch to terminology as noted above.

**Lemma 2.5.** *Identity is a symmetric relation, i.e. for every type $A$ and every $x, y : A$ there is a function*

$$(x =_A y) \to (y =_A x)$$

*denoted by $\lambda p.p^{-1}$, such that for every $x : A$*

$$\mathsf{refl}_x{}^{-1} \equiv \mathsf{refl}_x.$$

*We call $p^{-1}$ the **inverse** of $p$.*

*Proof.* We consider the type family

$$C : \prod_{x,y:A} (x =_A y) \to \mathcal{U},$$

defined by the equation

$$C(x, y, p) :\equiv (y =_A x).$$

Then, for every $x : A$, we have that

$$C(x, x, \mathsf{refl}_x) \equiv (x =_A x)$$

and therefore

$$\mathsf{refl}_x : C(x, x, \mathsf{refl}_x).$$

We conclude that

$$\lambda x.\mathsf{refl}_x : \prod_{x:A} C(x, x, \mathsf{refl}_x).$$

By the induction principle for identity types, there is a dependent function

$$\lambda x.\lambda y.\lambda p.p^{-1} : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p),$$

such that

$$\mathsf{refl}_x{}^{-1} \equiv \mathsf{refl}_x.$$

□

**Lemma 2.6.** *Identity is a transitive relation, i.e. for every type $A$ and every $x, y, z : A$ there is a function*

$$(x =_A y) \to (y =_A z) \to (x =_A z)$$

*denoted by $\lambda p.\lambda q.p \centerdot q$, such that, for every $x : A$*

$$\mathsf{refl}_x \centerdot \mathsf{refl}_x \equiv \mathsf{refl}_x.$$

*We call $p \centerdot q$ the **concatenation** of $p$ and $q$.*

*Proof.* We consider the type family

$$C : \prod_{x,y:A} (x =_A y) \to \mathcal{U},$$

defined by the equation

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

We aim to inhabit the type

$$\prod_{x:A} C(x, x, \mathsf{refl}_x). \tag{1}$$

Since for every $x : A$, we have that

$$C(x, x, \mathsf{refl}_x) \equiv \prod_{z:a} (x =_A z) \to (x =_A z),$$

this means to inhabit

$$\prod_{x,z:A} (x =_A z) \to (x =_A z).$$

Again, we can use the induction principle for identity types, where we consider the type family

$$D : \prod_{x,z:A} (x =_A z) \to \mathcal{U},$$

defined by the equation

$$D(x, z, q) :\equiv (x =_A z).$$

Then, for every $x : A$, we have that

$$D(x, x, \mathsf{refl}_x) \equiv (x =_A x)$$

and therefore

$$\lambda x.\mathsf{refl}_x : \prod_{x:A} D(x, x, \mathsf{refl}_x).$$

By the induction principle for identity types, there is a dependent function

$$\mathsf{F} : \prod_{x,z:A} \prod_{q:x=_A z} D(x, z, q),$$

such that

$$\mathsf{F}(x, x, \mathsf{refl}_x) \equiv \mathsf{refl}_x.$$

As noted above, we have

$$\prod_{x:A} C(x, x, \mathsf{refl}_x) \equiv \prod_{x:A} \prod_{z:A} (x =_A z) \to (x =_A z) \equiv$$
$$\equiv \prod_{x,z:A} \prod_{q:x=_A z} D(x, z, q),$$

therefore $\mathsf{F}$ inhabits (1). By the induction principle for identity types, there is a function

$$\mathsf{G} : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p),$$

such that

$$\mathsf{G}(x, x, \mathsf{refl}_x) \equiv \mathsf{F}(x).$$

Lastly, we note that

$$\mathsf{H} :\equiv \lambda x.\lambda y.\lambda z.\lambda p.\lambda q.\mathsf{G}(x, y, p)(z, q)$$

has type

$$\prod_{x,y,z:A} \prod_{p:x=_A y} \prod_{q:y=_A z} x =_A z$$

and therefore, given $x, y, z : A$, $p : x =_A y$ and $q : y =_A z$, we suppress $x, y$ and $z$ and define

$$(p \centerdot q) :\equiv \mathsf{H}(x, y, z, p, q)$$

and then

$$(\mathsf{refl}_x \centerdot \mathsf{refl}_x) \equiv \mathsf{H}(x, x, x, \mathsf{refl}_x, \mathsf{refl}_x) \equiv$$
$$\equiv \mathsf{G}(x, x, \mathsf{refl}_x)(x, \mathsf{refl}_x) \equiv \mathsf{F}(x, x, \mathsf{refl}_x) \equiv \mathsf{refl}_x.$$

□

Lastly we prove that equal terms can be substituted for one another: if $x = y$, then any property $P : A \to \mathcal{U}$ that we can establish for $x$ must hold also for $y$.

**Lemma 2.7** (Indiscernability of identicals)**.** *Assume we are given a type family $P : A \to \mathcal{U}$, two terms $x, y : A$ and a path $p : x =_A y$. Then there is a function*

$$\mathsf{transport}^P(p) : P(x) \to P(y),$$

*such that for every $x : A$ we have*

$$\mathsf{transport}^P(\mathsf{refl}_x) \equiv \mathsf{id}_{P(x)}.$$

*Proof.* We consider the type family

$$C : \prod_{x,y:A} (x =_A y) \to \mathcal{U},$$

defined by the equation

$$C(x, y, p) :\equiv P(x) \to P(y).$$

Then we have that

$$C(x, x, \mathsf{refl}_x) \equiv P(x) \to P(x)$$

and we can inhabit this type by $\mathsf{id}_{P(x)}$. But then

$$\lambda x.\mathsf{id}_{P(x)} : \prod_{x:A} C(x, x, \mathsf{refl}_x)$$

and by the induction principle for identity types, there is a dependent function

$$\mathsf{F} : \prod_{x,y:A} \prod_{p:x=_A y} C(x,y,p),$$

such that

$$\mathsf{F}(x,x,\mathsf{refl}_x) \equiv \mathsf{id}_{P(x)}.$$

Therefore, for any given $x, y : A$ and $p : x =_A y$, we suppress $x$ and $y$ and define

$$\mathsf{transport}^P(p) :\equiv \mathsf{F}(x,y,p).$$

Then it holds that

$$\mathsf{transport}^P(\mathsf{refl}_x) \equiv \mathsf{F}(x,x,\mathsf{refl}_x) \equiv \mathsf{id}_{P(x)}.$$

□

The following lemma covers the case where the type family $P$ in lemma 2.7 is constant.

**Lemma 2.8.** *If $P : A \to \mathcal{U}$ is a constant type family, i.e. $P(x) :\equiv B$ for some fixed $B : \mathcal{U}$, then for every $x, y : A$, $p : x = y$ and fixed $b : B$ there is a path*

$$\mathsf{transportconst}^B_p(b) : \mathsf{transport}^P(p,b) = b$$

*such that*

$$\mathsf{transportconst}^B_{\mathsf{refl}_x}(b) \equiv \mathsf{refl}_b.$$

*Proof.* We consider the type family

$$C : \prod_{x,y} (x =_A y) \to \mathcal{U},$$

defined by the equation

$$C(x,y,p) :\equiv \mathsf{transport}^P(p,b) = b.$$

Since by lemma 2.7

$$\mathsf{transport}^P(\mathsf{refl}_x) \equiv \mathsf{id}_{P(x)} \equiv \mathsf{id}_B,$$

we have for every $x : A$ that

$$C(x,x,\mathsf{refl}_x) \equiv b =_B b$$

and therefore

$$\mathsf{refl}_b : C(x, x, \mathsf{refl}_x).$$

We conclude that

$$\lambda x.\mathsf{refl}_b : \prod_{x:A} C(x, x, \mathsf{refl}_x).$$

By the induction principle for identity types, there is a dependent function

$$\mathsf{F} : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p),$$

such that

$$\mathsf{F}(x, x, \mathsf{refl}_x) \equiv \mathsf{refl}_b.$$

For any given $x, y : A$ and $p : x =_A y$, we suppress $x$ and $y$ and define

$$\mathsf{transportconst}_p^B(b) :\equiv \mathsf{F}(x, y, p)$$

and then

$$\mathsf{transportconst}_{\mathsf{refl}_x}^B(b) \equiv \mathsf{F}(x, x, \mathsf{refl}_x) \equiv \mathsf{refl}_b.$$

□

2.3.2. *Empty type.* The type $\mathbf{0}$ with no inhabitants is defined by no constructors. Assume we are given a type family $P : \mathbf{0} \to \mathcal{U}$, then there is a dependent function

$$\mathsf{F} : \prod_{x:\mathbf{0}} P(x),$$

with no computation rules. Under the proposition as types interpretation the empty type corresponds to falsity.

2.3.3. *Unit type.* The unit type $\mathbf{1}$ is the inductive type freely generated by just one point constructor:

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

Assume we are given a type family $P : \mathbf{1} \to \mathcal{U}$ together with a term $t : P(\star)$, then there is a dependent function

$$\mathsf{F} : \prod_{x:\mathbf{1}} P(x),$$

such that

$$\mathsf{F}(\star) \equiv t.$$

The elimination rule for identity types allows us to formulate a propositional uniqueness principle for the unit type.

**Lemma 2.9.** *The unit type has only one inhabitant, that is:*

$$\prod_{x:\mathbf{1}} x =_{\mathbf{1}} \star$$

*Proof.* We consider the type family

$$P : \mathbf{1} \to \mathcal{U},$$

defined by the equation

$$P(x) :\equiv x =_{\mathbf{1}} \star$$

and aim to inhabit the type

$$\prod_{x:\mathbf{1}} P(x).$$

By the induction principle of the unit type, it suffices to give a proof for

$$P(\star) \equiv \star = \star$$

and in this case we can use $\mathsf{refl}_\star$. □

2.3.4. *Pair types.* We note that the following results will be generalized in the next section, where we will introduce dependent pairs. We primarily present the simpler versions here for the reader not yet familiar with the subject. So, assume we are given two types $A, B : \mathcal{U}$, then we can form the type of pairs or product type $A \times B$, which can be interpreted as logical conjunction. The pair type is freely generated by one constructor:

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

This means that for every inhabitant $x : A$ and every inhabitant $y : B$, we can introduce an inhabitant

$$(x, y) : A \times B.$$

If we are given a type family $P : A \times B \to \mathcal{U}$ together with a dependent function

$$\mathsf{G} : \prod_{x:A} \prod_{y:B} P\big((x, y)\big),$$

we can introduce a dependent function

$$\mathsf{F} : \prod_{z:A\times B} P(z),$$

with the obvious computation rule:

$$\mathsf{F}\big((x,y)\big) :\equiv \mathsf{G}(x)(y)$$

The induction principle for pair types gives the projection functions, since we can define

$$\lambda x.\lambda y.x : A \to B \to A,$$

which, by the induction principle of pair types, gives a function

$$\pi_1 : A \times B \to A,$$

such that for every $x :$ and $y : B$

$$\pi_1\big((x,y)\big) \equiv x$$

and

$$\lambda x.\lambda y.y : A \to B \to B,$$

which gives a function

$$\pi_2 : A \times B \to B$$

such that for every $x : A$ and $y : B$

$$\pi_2\big((x,y)\big) \equiv y.$$

Using the projection functions we can express a propositional uniqueness principle, as we did for the unit type.

**Lemma 2.10.** *Assume we are given two types $A, B : \mathcal{U}$. The inhabitants of the pair type $A \times B$ are the pairs of terms in $A$ and $B$, that is:*

$$\prod_{z:A\times B} z = \big(\pi_1(z), \pi_2(z)\big)$$

*Proof.* We consider the type family

$$P : A \times B \to \mathcal{U},$$

defined by the equation

$$P(z) :\equiv z = \big(\pi_1(z), \pi_2(z)\big)$$

and we aim to inhabit the type

$$\prod_{z:A\times B} P(z).$$

Since by the computation rules of the projection functions we have for every $x : A$ and $y : B$ that

$$\pi_1\big((x,y)\big) \equiv x$$

and

$$\pi_2\big((x,y)\big) \equiv y,$$

we get that

$$P\big((x,y)\big) \equiv \big((x,y) = (x,y)\big).$$

Therefore we can define a dependent function

$$\mathsf{G} : \prod_{x:A}\prod_{y:B} P\big((x,y)\big),$$

by the defining equation

$$\mathsf{G}(x)(y) :\equiv \mathsf{refl}_{(x,y)}.$$

Then, by the induction principle for pairs, there is a dependent function

$$\mathsf{F} : \prod_{z:A\times B} z = \big(\pi_1(z), \pi_2(z)\big),$$

such that

$$\mathsf{F}\big((x,y)\big) \equiv \mathsf{refl}_{(x,y)}.$$

□

*Remark* 2.11. Assume we are given two functions $f : A \to A'$ and $g : B \to B'$. We can define

$$\lambda x.\lambda y.\big(f(x), g(y)\big) : A \to B \to A' \times B'$$

and then, by the recursion principle of the product, there is a function

$$(f,g) : A \times B \to A' \times B',$$

such that

$$(f,g)\big((x,y)\big) \equiv \big(f(x), g(y)\big).$$

2.3.5. *Dependent pair types.* Similar to the case of dependent functions, we can generalize the notion of a product in such a way that the type of the

second coordinate may vary along the points in the first coordinate. Assume we are given a type $A : \mathcal{U}$ together with a type family $P : A \to \mathcal{U}$, then we can introduce the dependent pair type $\sum_{x:A} P(x)$. If not specified otherwise by using brackets, the scope of such an expression is everything after "$\sum_{x:A}$". The dependent pair types are freely generated by one constructor:

$$(\_,\_) : \prod_{x:A} \big(P(x) \to \sum_{x:A} P(x)\big),$$

This means that for every inhabitant $x : A$ and every proof $t : P(a)$, we can introduce an inhabitant

$$(a, t) : \sum_{x:A} P(x).$$

If we are given a type $A : \mathcal{U}$, a type family

$$P : A \to \mathcal{U},$$

a type family

$$C : \big(\sum_{x:A} P(x)\big) \to \mathcal{U}$$

and a dependent function

$$\mathsf{G} : \prod_{x:A} \prod_{t:P(x)} C\big((x, t)\big),$$

then there is a dependent function

$$\mathsf{F} : \prod_{s:\sum_{x:A} P(x)} C(s),$$

such that for every $x : A$ and $t : P(x)$

$$\mathsf{F}\big((x, t)\big) \equiv \mathsf{G}(x)(t).$$

We note that pair types are just a special case of dependent pair types, where the type family $P$ is constant, that is, if $P \equiv \lambda x.B$ for some fixed $B : \mathcal{U}$, we have that

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

We can generalize the projection functions to dependent pairs, but since the situation is more involved, we state this in a lemma.

**Lemma 2.12.** *Assume we are given a type $A : \mathcal{U}$ together with a type family $P : A \to \mathcal{U}$. Then there is a function*

$$\pi_1 : \big(\sum_{x:A} P(x)\big) \to A$$

*and there is a dependent function*

$$\pi_2 : \prod_{s:\sum_{x:A} P(x)} P\big(\pi_1(s)\big),$$

*such that for every $x : A$ and $t : P(x)$ we have that*

$$\pi_1\big((x,t)\big) \equiv x$$

*and*

$$\pi_2\big((x,t)\big) \equiv t.$$

*Proof.* To define the first projection, we consider the constant type family

$$C : \big(\sum_{x:A} P(x)\big) \to \mathcal{U},$$

defined by the equation

$$C(s) :\equiv A$$

and we aim to inhabit

$$\prod_{s:\sum_{x:A} P(x)} C(s) \equiv \big(\sum_{x:A} P(x)\big) \to A.$$

To this end we define an inhabitant

$$\mathsf{G} : \prod_{x:A} \prod_{t:P(x)} A$$

by the defining equation

$$\mathsf{G}(x,t) :\equiv x$$

and then, by the recursion principle for dependent pairs, there is a function

$$\pi_1 : \big(\sum_{x:A} P(x)\big) \to A,$$

such that

$$\pi_1\big((x,t)\big) :\equiv x.$$

For the second projection we note that the type of the codomain may now vary, i.e. the second projection has to be a dependent function of type:

$$\pi_2 : \prod_{s:\sum_{x:A} P(x)} P\big(\pi_1(s)\big).$$

We use the induction principle for dependent pairs, where we consider the type family

$$C : \big(\sum_{x:A} P(x)\big) \to \mathcal{U},$$

defined by the equation

$$C(s) :\equiv P\big(\pi_1(s)\big)$$

and aim to inhabit the type

$$\prod_{s:\sum_{x:A} P(x)} C(s).$$

We can define a function

$$\mathsf{G} : \prod_{x:A} \prod_{t:P(x)} C\big((x,t)\big)$$

by the defining equation

$$\mathsf{G}(x)(t) :\equiv p$$

and then, by the induction principle for dependent pairs, there is a dependent function

$$\pi_2 : \prod_{s:\sum_{x:A} P(x)} P\big(\pi_1(s)\big)$$

such that

$$\pi_2\big((x,t)\big) \equiv t.$$

□

*Remark* 2.13. If we are given a dependent function $\mathsf{F} : \prod_{(s:\sum_{x:A} P(x))} C(s)$, we may write $\mathsf{F}(x,t)$ instead of $\mathsf{F}\big((x,t)\big)$ for simplicity.

Under the propositions as types interpretation, dependent pairs correspond to existential quantification. We note that our notion of existence is indeed constructive, since for every $s : \sum_{x:A} P(x)$ the projection to the first coordinate $\pi_1(y) : A$ gives an inhabitant of $A$ and the fact that this inhabitant has property $P$ is proven by $\pi_2(y) : P\big(\pi_1(y)\big)$. As we did before for pair types,

we can prove that the inhabitants of the dependent pair types are indeed the pairs of terms $x$ of $A$ and $t$ of $P(x)$.

**Lemma 2.14.** *Assume we are given a type $A : \mathcal{U}$ together with a type family $P : A \to \mathcal{U}$, then we have:*

$$\prod_{s:\sum_{x:A} P(x)} y = \big(\pi_1(s), \pi_2(s)\big)$$

*Proof.* Since by definition of the projection functions

$$\pi_1(x, t) \equiv x$$

and

$$\pi_2(x, t) \equiv t,$$

We can define a dependent function

$$\mathsf{G} : \prod_{x:A} \prod_{t:P(x)} (x, t) = \big(\pi_1(x, t), \pi_2(x, t)\big),$$

by the defining equation

$$\mathsf{G}(x)(t) :\equiv \mathsf{refl}_{(x,t)}.$$

By the induction principle for dependent pairs, there is a dependent function

$$\mathsf{F} : \prod_{s:\sum_{x:A} P(x)} s = \big(\pi_1(s), \pi_2(s)\big),$$

such that

$$\mathsf{F}(x, t) \equiv \mathsf{refl}_{(x,t)}.$$

□

2.3.6. *Coproduct types.* Assume we are given two types $A, B : \mathcal{U}$, then we can form the coproduct or sum type $A + B$, which can be thought of as the disjoint union of the two types or as logical disjunction under the proposition as types interpretation. The coproduct of $A$ and $B$ is the inductive type, freely generated by the constructors

$$\mathsf{inl} : A \to A + B$$

and

$$\mathsf{inr} : B \to A + B.$$

If we are given a type family $P : A + B \to \mathcal{U}$ and two dependent functions

$$\mathsf{G}_l : \prod_{x:A} P\big(\mathsf{inl}(x)\big)$$

and

$$\mathsf{G}_r : \prod_{y:B} P\big(\mathsf{inr}(b)\big),$$

then there is a dependent function

$$\mathsf{F} : \prod_{w:A+B} P(w),$$

such that

$$\mathsf{F}\big(\mathsf{inl}(a)\big) :\equiv \mathsf{G}_l(a)$$

and

$$\mathsf{F}\big(\mathsf{inr}(b)\big) :\equiv \mathsf{G}_r(b).$$

2.4. **Propositions as types.** In the preceding sections we presented the *propositions as types interpretation*, by assigning the quantifiers and logical connectives appropriate formation rules. We will frequently use terminology in accordance to this and therefore summarize it in the following table:

| type theory | first order logic |
|---|---|
| $\mathbf{0}$ | $\bot$ |
| $\mathbf{1}$ | $\top$ |
| $A \to B$ | $A \to B$ |
| $A \times B$ | $A \wedge B$ |
| $A + B$ | $A \vee B$ |
| $\prod_{(x:A)} P(x)$ | $\forall_x P(x)$ |
| $\sum_{x:A} P(x)$ | $\exists_x P(x)$ |

For the purpose of this work, another point of view will be more important and is the content of the following section.

## 3. The homotopy interpretation

3.1. **Basic behavior of functions and identifications.** Based on work of Hofmann and Streicher (see [7]), who gave a model of ITT in which identity types are inhabited by more then one element, Awodey, Warren and Voevodsky (see [1], [11]) proposed the point of view that inhabitants of the

identity type $x =_A y$ better ought to be thought of as *paths* between $x$ and $y$ in a space $A$ and equalities of the form $p =_{x=y} q$ ought to be thought of as *homotopies* between the paths $p, q : x = y$ in $A$ and so on and this we call the *homotopy-* or *types as spaces* interpretation. Under this interpretation it is not desirable anymore that there is only one way to inhabit identity types. To get a better grasp on this, let us consider again the induction principle for identity types, which from now on, to further stress the point of view that types are spaces and equalities are paths, will be called *path induction*. In a homotopic reading it states that in order to establish a property for every path $p : x = y$, with two distinct endpoints, it suffices to show it for the constant path $\mathsf{refl}_x$. This is reasonable, when we think of properties in our theory as being those that are invariant under homotopy, since every path with one free endpoint can be continuously deformed to the constant path. The same can not be said for loops $l : x = x$, for instance, it is well known that not all loops of the circle are homotopic to each other. Therefore, adding a principle such as UIP to ITT is not reasonable under the homotopy interpretation anymore.

The analogy between identity types and path spaces in topology is very strong and we will present the most basic results that show this in the rest of this section. In fact we have already defined operations on identity types that correspond to inversion and concatenation of paths in lemma 2.5 and lemma 2.6 respectively. The next lemma now justifies this nomenclature, by showing that these operations behave appropriately.

**Lemma 3.1.** *Assume we are given a type $A : \mathcal{U}$, points $x, y, z, w : A$ and paths $p : x = y$, $q : y = z$ and $r : z = w$. Then we have the following:*

*(i)* $p = p \centerdot \mathsf{refl}_y$ *and* $p = \mathsf{refl}_x \centerdot p$.

*(ii)* $p^{-1} \centerdot p = \mathsf{refl}_y$ *and* $p \centerdot p^{-1} = \mathsf{refl}_x$.

*(iii)* $(p^{-1})^{-1} = p$.

*(iv)* $p \centerdot (q \centerdot r) = (p \centerdot q) \centerdot r$.

*Proof.* We can proof all four points by path induction. For *(i)* it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$ and since by lemma 2.6 $\mathsf{refl}_x \centerdot \mathsf{refl}_x \equiv \mathsf{refl}_x$, in this case we can proof both claims by $\mathsf{refl}_{\mathsf{refl}_x}$.

For *(ii)* it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$ and since by lemma 2.5 ${\mathsf{refl}_x}^{-1} \equiv \mathsf{refl}_x$ and by lemma 2.6 $\mathsf{refl}_x \cdot \mathsf{refl}_x \equiv \mathsf{refl}_x$, in this case we can proof both claims by $\mathsf{refl}_{\mathsf{refl}_x}$.
For *(iii)* it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$ and since using lemma 2.5 twice gives ${\mathsf{refl}_x}^{-1^{-1}} \equiv {\mathsf{refl}_x}^{-1} \equiv \mathsf{refl}_x$, in this case we can proof the claim by $\mathsf{refl}_{\mathsf{refl}_x}$.
For *(iv)* it suffices to consider the case where $x \equiv y \equiv z \equiv w$ and $p \equiv q \equiv r \equiv \mathsf{refl}_x$. By using lemma 2.6 twice, we have that

$$\mathsf{refl}_x \cdot (\mathsf{refl}_x \cdot \mathsf{refl}_x) \equiv \mathsf{refl}_x \cdot \mathsf{refl}_x \equiv \mathsf{refl}_x$$

and similarly

$$(\mathsf{refl}_x \cdot \mathsf{refl}_x) \cdot \mathsf{refl}_x \equiv \mathsf{refl}_x \cdot \mathsf{refl}_x \equiv \mathsf{refl}_x.$$

Hence, in this case, we can proof the claim by $\mathsf{refl}_{\mathsf{refl}_x}$. □

Next we want to establish that functions respect equalities. Under the homotopy interpretation this says that all functions preserve paths.

**Lemma 3.2.** *Assume we are given two types $A, B : \mathcal{U}$ together with a function $f : A \to B$. Then, for every $x, y : A$, there is a function*

$$\mathsf{ap}_f : (x =_A y) \to (f(x) =_B f(y)),$$

*such that for every $x : A$ we have*

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

*Proof.* It suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$ and then we need to inhabit the type $f(x) =_B f(x)$. We can do so by $\mathsf{refl}_{f(x)}$. □

Later on we will also need the following.

**Lemma 3.3.** *Assume we are given two types $A, B : \mathcal{U}$ together with a function $f : A \to B$. Then we have that for every $x, y : A$ there is a function*

$$\mathsf{ap}_f^2 : \prod_{p,q:x=y} (p = q) \to \mathsf{ap}_f(p) = \mathsf{ap}_f(q),$$

*such that*

$$\mathsf{ap}_f^2(\mathsf{refl}_p) \equiv \mathsf{refl}_{\mathsf{ap}_f(p)}.$$

*Proof.* By path induction it suffices to consider the case where $p \equiv q$ and $r \equiv \mathsf{refl}_p$ and then we need to inhabit the type $\mathsf{ap}_f(p) = \mathsf{ap}_f(p)$. We can do so by $\mathsf{refl}_{\mathsf{ap}_f(p)}$. □

**Lemma 3.4.** *Assume we are given three types $A, B, C : \mathcal{U}$ together with functions $f : A \to B$ and $g : B \to C$ and paths $p : x =_A y$ and $q : y =_A z$. Then we have:*

*(i)* $\mathsf{ap}_f(p \cdot q) = \mathsf{ap}_f(p) \cdot \mathsf{ap}_f(q)$.

*(ii)* $\mathsf{ap}_f(p^{-1}) = \mathsf{ap}_f(p)^{-1}$.

*(iii)* $\mathsf{ap}_g\big(\mathsf{ap}_f(p)\big) = \mathsf{ap}_{g \circ f}(p)$.

*(iv)* $\mathsf{ap}_{\mathsf{id}_A}(p) = p$.

*Proof.* All claims can be proven by path induction. For *(i)* it suffices to consider the case where $x \equiv y \equiv z$ and $p \equiv q \equiv \mathsf{refl}_x$ and since by lemma 2.6

$$\mathsf{refl}_x \cdot \mathsf{refl}_x = \mathsf{refl}_x$$

and by lemma 3.2

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

this means we need to show

$$\mathsf{refl}_{f(x)} = \mathsf{refl}_{f(x)},$$

which we can do by $\mathsf{refl}_{\mathsf{refl}_{f(x)}}$.
For *(ii)* it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$ and since by lemma 2.5

$$\mathsf{refl}_x{}^{-1} \equiv \mathsf{refl}_x,$$

in this case we need to show

$$\mathsf{ap}_f(\mathsf{refl}_x) = \mathsf{ap}_f(\mathsf{refl}_x)^{-1}.$$

Since by lemma 3.2

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

and using again lemma 2.5 gives

$$\mathsf{refl}_{f(x)}{}^{-1} \equiv \mathsf{refl}_{f(x)},$$

we can further simplify this to

$$\mathsf{refl}_{f(x)} = \mathsf{refl}_{f(x)},$$

which we can inhabit by $\mathsf{refl}_{\mathsf{refl}_{f(x)}}$.
For *(iii)* it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$ and since by lemma 3.2 for any $h : A \to C$ it holds that

$$\mathsf{ap}_h(\mathsf{refl}_x) \equiv \mathsf{refl}_{h(x)},$$

proofing the claim means to show that

$$\mathsf{refl}_{g(f(x))} = \mathsf{refl}_{g(f(x))}$$

and we can do so by $\mathsf{refl}_{g(f(x))}$.
For *(iv)* it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$ and since by lemma 3.2 we have that

$$\mathsf{ap}_{\mathsf{id}_A}(\mathsf{refl}_x) \equiv \mathsf{refl}_{\mathsf{id}_A(x)},$$

and

$$\mathsf{id}_A(x) \equiv x,$$

in this case we need to show

$$\mathsf{refl}_x = \mathsf{refl}_x$$

and we can do so by $\mathsf{refl}_{\mathsf{refl}_x}$. □

*Remark* 3.5 (Application of a constant function). Assume we are given two types $A, B : \mathcal{U}$, some fixed $b : B$ and let us consider the constant function:

$$b_A :\equiv \lambda x.b : A \to B$$

Then we have that:

$$\prod_{x,y:A} \prod_{p:x=y} \mathsf{ap}_{b_A}(p) = \mathsf{refl}_b$$

*Proof.* By path induction it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$, but since $b_A(x) \equiv b$ and applying the computation rule from lemma 3.2, we can use $\mathsf{refl}_{\mathsf{refl}_b}$ in this case. □

We note that we can not say the same thing for functions that are only propositionally constant, i.e. functions $f : A \to B$, such that $f = b_A$, because

we have for every $x, y : A$ and every $p : x = y$ that

$$\mathsf{ap}_f(p) : f(x) = f(y)$$

and

$$\mathsf{refl}_b : b = b.$$

But this means that

$$\mathsf{ap}_f(p) = \mathsf{refl}_b$$

is generally not well typed.

*Remark* 3.6 (Application of identity is identity). Assume we are given a type $A : \mathcal{U}$ and let us consider the identity function

$$\mathsf{id}_A :\equiv \lambda x.x.$$

Then we have that:

$$\prod_{x,y:A} \prod_{p:x=y} \mathsf{ap}_{\mathsf{id}_A}(p) = p$$

*Proof.* By path induction it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$, but since $\mathsf{id}_A(x) \equiv x$ and applying the computation rule from lemma 3.2, we can use $\mathsf{refl}_{\mathsf{refl}_x}$ in this case. □

There is also a dependent version of lemma 3.2 that we want to consider next. However we need to be careful, because for every dependent map

$$\mathsf{F} : \prod_{x:A} P(x)$$

and every path

$$p : x =_A y,$$

the expression $\mathsf{F}(x) = \mathsf{F}(y)$ is generally not well typed, since $\mathsf{F}(x) : P(x)$ and $\mathsf{F}(y) : P(y)$. The solution is to use transport from lemma 2.7.

**Lemma 3.7** (Dependent application)**.** *Assume we are given a type $A : \mathcal{U}$, a type family $P : A \to \mathcal{U}$ and a dependent function $\mathsf{F} : \prod_{(x:A)} P(x)$. Then, for every $x, y : A$, there is a dependent function*

$$\mathsf{apd}_{\mathsf{F}} : \prod_{p:x=y} \big(\mathsf{transport}^P(p, \mathsf{F}(x)) = \mathsf{F}(y)\big),$$

*such that for every* $x : A$ *we have*

$$\mathsf{apd}_{\mathsf{F}}(\mathsf{refl}_x) \equiv \mathsf{refl}_{\mathsf{F}(x)}.$$

*Proof.* By path induction it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$ and in this case we need to show that

$$\mathsf{transport}^P(\mathsf{refl}_x, \mathsf{F}(x)) = \mathsf{F}(x).$$

By lemma 2.7 we have that

$$\mathsf{transport}^P(\mathsf{refl}_x) \equiv \mathsf{id}_{P(x)}$$

and hence our goal simplifies to

$$\mathsf{F}(x) = \mathsf{F}(x),$$

which can be proven by $\mathsf{refl}_{\mathsf{F}(x)}$. □

We recall that function types are just a special case of dependent function types, where the type family is constant, i.e

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

Therefore it makes sense to consider for every function $f : A \to B$ and every path $p : x =_A y$ application

$$\mathsf{ap}_f(p) : f(x) = f(y),$$

as well as dependent application

$$\mathsf{apd}_f(p) : \mathsf{transport}^B(p, f(x)) = f(y)$$

of $f$ to $p$ and we want to relate the two in the next lemma.

**Lemma 3.8.** *Assume we are given two types* $A, B : \mathcal{U}$*, a function* $f : A \to B$*, two points* $x, y : A$ *and a path* $p : x =_A y$*, then we have:*

$$\mathsf{apd}_f(p) = \mathsf{transportconst}_p^B(f(x)) \cdot \mathsf{ap}_f(p)$$

*Proof.* By path induction it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$. By lemma 3.7 the left hand side of the equality simplifies

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

By lemma 2.8 we have

$$\mathsf{transportconst}^{B}_{\mathsf{refl}_x}(f(x)) \equiv \mathsf{refl}_{f(x)}$$

and by lemma 3.2 we have

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

Together with the computation rule from lemma 2.6, we get that also the right hand side of the equality is judgmentally equal to $\mathsf{refl}_{f(x)}$. Therefore, in this case, we can proof the claim by $\mathsf{refl}_{\mathsf{refl}_{f(x)}}$. □

3.2. **Homotopies and equivalences.** In this section we transfer more ideas from homotopy theory to ITT and present appropriate notions for *homotopy between two functions* and *equivalence of spaces*.

**Definition 3.9** (Homotopies)**.** Assume we are given two types $A, B : \mathcal{U}$ together with two maps $f : A \to B$ and $g : A \to B$. A **homotopy** from $f$ to $g$ is a dependent function of type

$$(f \sim g) :\equiv \prod_{x:A} \big(f(x) = g(x)\big).$$

Thus, a homotopy between functions $f$ and $g$ is a proof that these functions are pointwise equal. Next we present a proof, that equality $f = g$ between functions implies pointwise equality, i.e. equivalence.

**Lemma 3.10.** *Assume we are given two types $A, B : \mathcal{U}$ together with two functions $f, g : A \to B$. Then there is a function*

$$\mathsf{happly} : (f = g) \to (f \sim g),$$

*such that, if $f \equiv g$, we have*

$$\mathsf{happly}(\mathsf{refl}_f) \equiv \lambda x.\mathsf{refl}_{f(x)}.$$

*Proof.* We consider the type family

$$C : \prod_{f,g:A\to B} (f = g) \to \mathcal{U}$$

defined by the equation

$$C(f, g, p) :\equiv (f \sim g).$$

Then, for every $f : A \to B$, we have that

$$C(f, f, \mathsf{refl}_f) \equiv (f \sim f)$$

and therefore

$$\lambda x.\mathsf{refl}_{f(x)} : C(f, f, \mathsf{refl}_f).$$

We conclude that

$$\lambda f.\lambda x.\mathsf{refl}_{f(x)} : \prod_{f:A\to B} C(f, f, \mathsf{refl}_f)$$

and then, by path induction, there is a dependent function

$$\mathsf{F} : \prod_{f,g:A\to B} \prod_{p:f=_{A\to B}g} C(f, g, p),$$

such that

$$\mathsf{F}(f, f, \mathsf{refl}_f) \equiv \lambda x.\mathsf{refl}_{f(x)}.$$

Lastly, given two functions $f, g : A \to B$, we suppress $f$ and $g$ and define

$$\mathsf{happly}(p) :\equiv \mathsf{F}(f, g, p).$$

□

**Lemma 3.11.** *We let* $\mathsf{eval} : A \to (A \to B) \to B$ *be function evaluation at a, i.e. for every function* $f : A \to B$ *we define* $\mathsf{eval}(a, f) :\equiv f(a)$*. Then we have for every two functions* $f, g : A \to B$*, such that* $p : f = g$ *and for every* $x : A$*:*

$$\mathsf{ap}_{\mathsf{eval}(x)}(p) = \mathsf{happly}(p)(x)$$

*Proof.* By path induction, it suffices to consider the case where $f \equiv g$ and $p \equiv \mathsf{refl}_f$. Since by the computation rule of $\mathsf{happly}$, given in lemma 3.10, we have

$$\mathsf{happly}(\mathsf{refl}_f)(x) \equiv \mathsf{refl}_{f(x)}$$

and

$$\mathsf{ap}_{\mathsf{eval}(x)}(\mathsf{refl}_f) \equiv \mathsf{refl}_{\mathsf{eval}(x,f)} \equiv \mathsf{refl}_{f(x)},$$

we can use $\mathsf{refl}_{\mathsf{refl}_{f(x)}}$ in this case. □

Next we present the notion of an equivalence. For our purposes, the following definition, which is due to Voevodsky, will suffice. There are

multiple other ways to define equivalences, we refer to chapter four of the HoTT book [10] for a detailed discussion.

**Definition 3.12** (Equivalences)**.** We say that a function $f : A \to B$ is an **equivalence**, if it has a left and a right inverse, that is:

$$\mathsf{isequiv}(f) :\equiv \Big(\sum_{g:B\to A} (f \circ g \sim \mathsf{id}_B)\Big) \times \Big(\sum_{h:B\to A} (h \circ f \sim \mathsf{id}_A)\Big).$$

We say the type $A : \mathcal{U}$ is equivalent to the type $B : \mathcal{U}$, if there is an equivalence between the two types:

$$(A \simeq B) :\equiv \sum_{f:A\to B} \mathsf{isequiv}(f)$$

Next we present a proof that $\mathsf{transport}$ from lemma 2.7 is always an equivalence.

**Lemma 3.13.** *Assume we are given a type $A : \mathcal{U}$ together with a type family $P : A \to \mathcal{U}$. Then we have that:*

$$\prod_{x,y:A} \prod_{p:x=y} \mathsf{isequiv}\big(\mathsf{transport}^P(p)\big)$$

*Proof.* By path induction it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$. By the computation rule from lemma 2.7 we have that

$$\mathsf{transport}^P(\mathsf{refl}_x) \equiv \mathsf{id}_{P(x)},$$

which is obviously an equivalence, since it is its own quasi inverse. □

*Remark* 3.14. Assume we are given two types $A, B : \mathcal{U}$ and functions $f : A \to B$ and $g : B \to A$. If we have

$$q_l : f \circ g \sim \mathsf{id}_B,$$

and

$$q_r : g \circ f \sim \mathsf{id}_A$$

we can inhabit $\mathsf{isequiv}(f)$ by

$$\big((g, q_l), (g, q_r)\big).$$

In this situation we may also say that $f$ and $g$ are **quasi-inverse** or **mutually inverse**.

**Lemma 3.15.** *Assume we are given two types $A, B : \mathcal{U}$ and a function $f : A \to B$, such that* $\mathsf{isequiv}(f)$*. Then there is a function $f' : B \to A$ that is quasi inverse to $f$.*

*Proof.* Assume we are given $\big((g, \mathsf{G}), (h, \mathsf{H})\big) : \mathsf{isequiv}(f)$, that is:

- $g : B \to A$
- $\mathsf{G} : \prod_{(y:B)} \big(f \circ g(y) =_B y\big)$
- $h : B \to A$
- $\mathsf{H} : \prod_{(x:A)} \big(h \circ f(x) =_A x\big)$

Then we can define terms

$$\mathsf{H} \circ g : \prod_{y:B} \big(h \circ f \circ g(y) = g(y)\big)$$

and

$$\lambda y.\mathsf{ap}_h\big(\mathsf{G}(y)\big) : \prod_{y:B} \big(h \circ f \circ g(y) = h(y)\big)$$

and, using these two terms, we can inhabit $g \sim h$ by

$$\mathsf{I}(y) :\equiv \big(\mathsf{H} \circ g(y)\big)^{-1} \cdot \mathsf{ap}_h\big(\mathsf{G}(y)\big).$$

Next, we note that

$$\mathsf{I} \circ f : \prod_{x:A} g \circ f(x) = h \circ f(x)$$

and hence we can inhabit

$$g \circ f \sim \mathsf{id}_A$$

by

$$\mathsf{J}(x) :\equiv \mathsf{I}\big(f(x)\big) \cdot \mathsf{H}(x).$$

Since we already have $\mathsf{G}$, which proves that $f \circ g \sim \mathsf{id}_B$, at hand, this shows that $f$ and $g$ are mutually inverse. □

With the notion of an equivalence at hand, we can present an alternative description of paths in dependent pair types.

**Theorem 3.16.** *Assume we are given a type $A : \mathcal{U}$ together with a type family $P : A \to \mathcal{U}$ and let $w, w' : \sum_{(x:A)} P(x)$. Then there is an equivalence*

$$(w = w') \simeq \sum_{p:\pi_1(w)=\pi_1(w')} \mathsf{transport}^P\big(p, \pi_2(w)\big) = \pi_2(w').$$

*Proof.* Step 1: Firstly, we want to define for every $w, w' : \sum_{(x:A)} P(x)$ a function of type

$$(w = w') \to \sum_{p:\pi_1(w)=\pi_1(w')} \mathsf{transport}^P\big(p, \pi_2(w)\big) = \pi_2(w').$$

To this end, we consider the type family

$$C : \prod_{w,w':\sum_{x:A} P(x)} (w =_{\sum_{x:A} P(x)} w') \to \mathcal{U},$$

defined by the equation

$$C(w, w', q) :\equiv \sum_{p:\pi_1(w)=\pi_1(w')} \mathsf{transport}^P(p, \pi_2(w)) = \pi_2(w').$$

By path induction, it suffices to consider the case where $w \equiv w'$ and $q \equiv \mathsf{refl}_w$ and then we have that

$$C(w, w, \mathsf{refl}_w) \equiv \sum_{p:\pi_1(w)=\pi_1(w)} \mathsf{transport}^P\big(p, \pi_2(w)\big) = \pi_2(w).$$

Now, since firstly

$$\mathsf{refl}_{\pi_1(w)} : \pi_1(w) = \pi_1(w)$$

and secondly, by lemma 2.7

$$\mathsf{transport}^P(\mathsf{refl}_{\pi_1(w)}) \equiv \mathsf{id}_{P(\pi_1(w))},$$

we can inhabit $C(w, w, \mathsf{refl}_w)$ by the pair

$$(\mathsf{refl}_{\pi_1(w)}, \mathsf{refl}_{\pi_2(w)}).$$

Therefore

$$\lambda w.(\mathsf{refl}_{\pi_1(w)}, \mathsf{refl}_{\pi_2(w)}) : \prod_{w:\sum_{x:A} P(x)} C(w, w, \mathsf{refl}_w)$$

and then, by the path induction, there is a dependent function

$$\Phi : \prod_{w,w':\sum_{x:A} P(x)} \prod_{q:w=w'} C(w, w', q),$$

such that

$$\Phi(w, w, \mathsf{refl}_w) \equiv (\mathsf{refl}_{\pi_1(w)}, \mathsf{refl}_{\pi_2(w)}).$$

Step 2: Next, we want to define a function in the other direction, i.e. we want to inhabit the type:

$$\prod_{w,w':\sum_{x:A} P(x)} \Big( \sum_{p:\pi_1(w)=\pi_1(w')} \mathsf{transport}^P(p, \pi_2(w)) = \pi_2(w') \Big) \to (w = w')$$

We start by using induction on the terms

$$w, w' : \sum_{x:A} P(x)$$

and

$$r : \sum_{p:\pi_1(w)=\pi_1(w')} \mathsf{transport}^P(p, \pi_2(w)) = \pi_2(w'),$$

which splits them into pairs $(w_1, w_2)$, $(w_1', w_2')$ and $(r_1, r_2)$ respectively, where

- $w_1, w_1' : A$,
- $w_2 : P(w_1)$,
- $w_2' : P(w_1')$,
- $r_1 : w_1 = w_1'$ and
- $r_2 : \mathsf{transport}^P(r_1, w_2) = w_2'$.

So, we consider the type family

$$E : \prod_{w_1,w_1':A} (w_1 = w_1') \to \mathcal{U}$$

defined by the equation

$$E(w_1, w_1', r_1) :\equiv \prod_{w_2:P(w_1)} \prod_{w_2':P(w_1')} \prod_{r_2:\mathsf{transport}^P(r_1,w_2)=w_2'} (w_1, w_2) = (w_1', w_2')$$

and aim to inhabit the type

$$\prod_{w_1,w_1':A} \prod_{r_1:w_1=w_1'} E(w_1, w_1', r_1).$$

By path induction it suffices to consider the case where $w_1 \equiv w_1'$ and $r_1 \equiv \mathsf{refl}_{w_1}$ and then we have that $w_2$ and $w_2'$ are both of type $P(w_1)$. Since by the computation rule from lemma 2.7 it holds that

$$\mathsf{transport}^P(\mathsf{refl}_{w_1}) \equiv \mathsf{id}_{P(w_1)},$$

we get that

$$\mathsf{transport}^P(\mathsf{refl}_{w_1}, w_2) \equiv w_2.$$

But this means that we need to inhabit

$$E(w_1, w_1, \mathsf{refl}_{w_1}) \equiv \prod_{w_2, w_2' : P(w_1)} \prod_{r_2 : w_2 = w_2'} \big((w_1, w_2) = (w_1, w_2')\big).$$

We use path induction again, where we consider the type family

$$Q : \prod_{w_2, w_2' : P(w_1)} (w_2 = w_2') \to \mathcal{U},$$

defined by the equation

$$Q(w_2, w_2', r_2) :\equiv \big((w_1, w_2) = (w_1, w_2')\big)$$

and we aim to inhabit the type

$$\prod_{w_2, w_2' : P(w_1)} \prod_{r_2 : w_2 = w_2'} Q(w_2, w_2', r_2).$$

Then it suffices to consider the case where $w_2 \equiv w_2'$ and $r_2 \equiv \mathsf{refl}_{w_2}$, but in this case we can proof the claim by $\mathsf{refl}_{(w_1, w_2)}$. Therefore, for every $w_1 : A$, there is a dependent function

$$\mathsf{G}_{w_1} : \prod_{w_2, w_2' : P(w_1)} \prod_{r_2 : w_2 = w_2'} Q(w_2, w_2', r_2),$$

such that

$$\mathsf{G}_{w_1}(w_2, w_2, \mathsf{refl}_{w_2}) \equiv \mathsf{refl}_{(w_1, w_2)}.$$

But as noted above, we have that

$$E(w_1, w_1, \mathsf{refl}_{w_1}) \equiv \prod_{r_2 : w_2 = w_2'} Q(w_2, w_2', r_2)$$

and hence $\mathsf{G}_{w_1}$ inhabits $E(w_1, w_1, \mathsf{refl}_{w_1})$. This, in turn, gives a dependent function

$$\mathsf{J} : \prod_{w_1, w_1' : A} \prod_{r_1 : w_1 = w_1'} E(w_1, w_1', r_1),$$

such that

$$\mathsf{J}(w_1, w_1, \mathsf{refl}_{w_1}) \equiv \mathsf{G}_{w_1}.$$

This finishes the induction on $w$, $w'$ and $r$ and therefore gives a dependent function

$$\Psi : \prod_{w,w' : \sum_{x:A} P(x)} \Big( \sum_{p : \pi_1(w) = \pi_1(w')} \mathsf{transport}^P(p, \pi_2(w)) = \pi_2(w') \Big) \to (w = w'),$$

such that

$$\Psi\big((w_1, w_2), (w_1', w_2'), (r_1, r_2)\big) \equiv \mathsf{J}(w_1, w_1', r_1)(w_2, w_2', r_2).$$

Step 3: Next we want to show that for every $w, w' : \sum_{x:A} P(x)$, the functions $\Phi(w, w')$ and $\Psi(w, w')$ are mutually inverse. For readability we omit the occurrences of $w$ and $w'$. We start by showing that for every

$$r : \sum_{p : \pi_1(w) = \pi_1(w')} \mathsf{transport}^P(p, \pi_2(w))$$

it holds that

$$\Phi\big(\Psi(r)\big) = r.$$

As before, we use induction on

$$w, w' : \sum_{x:A} P(x)$$

and

$$r : \sum_{p : \pi_1(w) = \pi_1(w')} \mathsf{transport}^P(p, \pi_2(w)) = \pi_2(w'),$$

which splits them into pairs $(w_1, w_2)$, $(w_1', w_2')$ and $(r_1, r_2)$ respectively, where

- $w_1, w_1' : A$,
- $w_2 : P(w_1)$,
- $w_2' : P(w_1')$,
- $r_1 : w_1 = w_1'$ and
- $r_2 : \mathsf{transport}^P(r_1, w_2) = w_2'$.

Then we consider the type family

$$R : \prod_{w_1, w_1' : A} (w_1 = w_1') \to \mathcal{U}$$

defined by the equation

$$R(w_1, w_1', r_1) :\equiv \prod_{w_2 : P(w_1)} \prod_{w_2' : P(w_1')} \prod_{r_2 : \mathsf{transport}^P(r_1, w_2) = w_2'} \Phi\big(\Psi(r)\big) = r$$

and we aim to inhabit the type

$$\prod_{w_1,w_1':A}\ \prod_{r_1:w_1=w_1'} R(w_1,w_1',r_1).$$

By path induction it suffices to consider the case where $w_1 \equiv w_1'$ and $r_1 \equiv \mathsf{refl}_{w_1}$ and since by the computation rule from lemma 2.7 it holds that

$$\mathsf{transport}^P(\mathsf{refl}_{w_1}) \equiv \mathsf{id}_{P(w_1)}$$

and both $w_2$ and $w_2'$ are of the same tpe $P(w_1)$, we get

$$R(w_1,w_1,\mathsf{refl}_{w_1}) \equiv \prod_{w_2,w_2':P(w_1)}\ \prod_{r_2:w_2=w_2'} \Phi\Big(\Psi\big((\mathsf{refl}_{w_1},r_2)\big)\Big) = (\mathsf{refl}_{w_1},r_2).$$

Using path induction again, it suffices to consider the case where $w_2 \equiv w_2'$ and $r_2 \equiv \mathsf{refl}_{w_2}$ and we are left to proof

$$\Phi\Big(\Psi\big((\mathsf{refl}_{w_1},\mathsf{refl}_{w_2})\big)\Big) = (\mathsf{refl}_{w_1},\mathsf{refl}_{w_2}). \tag{2}$$

In this case we compute

$$\begin{aligned}\Psi\big((w_1,w_2),(w_1,w_2),(\mathsf{refl}_{w_1},\mathsf{refl}_{w_2})\big) &\equiv\\ \equiv \mathsf{J}(w_1,w_1,\mathsf{refl}_{w_1})(w_2,w_2,\mathsf{refl}_{w_2}) &\equiv\\ \equiv \mathsf{G}_{w_1}(w_2,w_2,\mathsf{refl}_{w_2}) \equiv \mathsf{refl}_{(w_1,w_2)}&\end{aligned}$$

and then

$$\Phi((w_1,w_2),(w_1,w_2),\mathsf{refl}_{(w_1,w_2)}) \equiv (\mathsf{refl}_{w_1},\mathsf{refl}_{w_2})$$

and therefore, in this case, we can proof (2) by $\mathsf{refl}_{(\mathsf{refl}_{w_1},\mathsf{refl}_{w_2})}$.
Step 4: Similarly we want to show that for every $w,w' : \sum_{x:A} P(x)$ and every $q : w = w'$, we have that:

$$\Psi\big(\Phi(q)\big) = q \tag{3}$$

As in step 1, we firstly use path induction on $q$, by which it suffices to consider the case where $w \equiv w'$ and $q \equiv \mathsf{refl}_w$ and then we have

$$\Phi(w,w,\mathsf{refl}_w) \equiv (\mathsf{refl}_{\pi_1(w)},\mathsf{refl}_{\pi_2(w)}).$$

Secondly we use induction on $w$, which splits the term into $(w_1,w_2)$ and then we can proceed as in step 3 and compute

$$\Psi\big((w_1,w_2),(w_1,w_2),(\mathsf{refl}_{w_1},\mathsf{refl}_{w_2})\big) \equiv \mathsf{refl}_{(w_1,w_2)}.$$

Therefore, in this case we can proof (3) by $\mathsf{refl}_{\mathsf{refl}_{(w_1,w_2)}}$. □

Theorem 3.16 gives an alternative way to infer uniqueness for dependent pairs from lemma 2.14.

**Corollary 3.17.** *Assume we are given a type $A : \mathcal{U}$ and a type family $P : A \to \mathcal{U}$, then we have:*

$$\prod_{y:\sum_{x:A} P(x)} y = \big(\pi_1(y), \pi_2(y)\big)$$

*Proof.* Assume we are given $y : \sum_{x:A} P(x)$. By theorem 3.16 it suffices to show that

$$\mathsf{transport}^P(\mathsf{refl}_{\pi_1(y)}, \pi_2(y)) = \pi_2(y)$$

and since by lemma 2.7 we have

$$\mathsf{transport}^P\big(\mathsf{refl}_{\pi_1(y)}\big) \equiv \mathsf{id}_{P(\pi_1(y))},$$

we can proof this claim by $\mathsf{refl}_{\pi_2(y)}$. □

For the special case of a non dependent product, theorem 3.16 tells us that the paths in a product space are pairs of paths.

**Corollary 3.18.** *Assume we are given two types $A, B : \mathcal{U}$ and two points $r, s : A \times B$, then we have:*

$$(r = s) \simeq \Big(\big(\pi_1(r) = \pi_1(s)\big) \times \big(\pi_2(r) = \pi_2(s)\big)\Big)$$

*Proof.* We let $P : A \to \mathcal{U}$ be the constant type family, defined by the equation $P(x) :\equiv B$. By theorem 3.16 we have that

$$(r = s) \simeq \sum_{p:(\pi_1(r)=\pi_1(s))} \mathsf{transport}^P\big(p, \pi_2(r)\big) = \pi_2(s).$$

Next we note that

$$\Big(\mathsf{transport}^P\big(p, \pi_2(r)\big) = \pi_2(s)\Big) \simeq \big(\pi_2(r) = \pi_2(s)\big),$$

since by lemma 2.8 we have a path

$$\mathsf{transportconst}^B_p(b) : \mathsf{transport}^P(p, \pi_2(r)) = \pi_2(r)$$

and hence, if we are given some

$$p : \mathsf{transport}^P\big(p, \pi_2(r)\big) = \pi_2(s),$$

we can define

$$\mathsf{transportconst}^B_p(b) \cdot p : \mathsf{transport}^P(p, \pi_2(r)) = \pi_2(r)$$

and if we are given some

$$q : \mathsf{transport}^P(p, \pi_2(r)) = \pi_2(r),$$

we can define

$$\big(\mathsf{transportconst}^B_p(b)\big)^{-1} \cdot q : \pi_2(r) = \pi_2(r).$$

This defines an equivalence by lemma 2.6 and therefore

$$(r = s) \simeq \big(\pi_1(r) = \pi_1(s)\big) \times \big(\pi_2(r) = \pi_2(s)\big).$$

□

**Definition 3.19.** We denote by

$$\mathsf{pair}^= : \Big(\big(\pi_1(r) = \pi_1(s)\big) \times \big(\pi_2(r) = \pi_2(s)\big)\Big) \to (r = s)$$

the equivalence from corollary 3.18. In particular we have that

$$\mathsf{pair}^=(\mathsf{refl}_x, \mathsf{refl}_y) \equiv \mathsf{refl}_{(x,y)}.$$

The next lemma characterizes inversion and concatenation of paths in a product space component-wise.

**Lemma 3.20.** *Assume we are given two types $A, B : \mathcal{U}$ together with points $x, y, z : A$ and $x', y', z' : B$, paths $p : x = y$ and $q : y = z$ and paths $r : x' = y'$ and $s : y' = z'$. Then we have:*

*(i)* $\mathsf{pair}^=(p^{-1}, r^{-1}) = \mathsf{pair}^=(p, r)^{-1}$

*(ii)* $\mathsf{pair}^=(p, r) \cdot \mathsf{pair}^=(q, s) = \mathsf{pair}^=(p \cdot q, r \cdot s)$

*Proof.* We can proof both claims by path induction. For *(i)* it suffices to consider the case where $x \equiv y$, $x' \equiv y'$, $p \equiv \mathsf{refl}_x$ and $r \equiv \mathsf{refl}_{x'}$. By definition 3.19, we have that

$$\mathsf{pair}^=(\mathsf{refl}_x, \mathsf{refl}_{x'}) \equiv \mathsf{refl}_{(x,x')}$$

and since by lemma 2.5 we have that

$$\mathsf{refl}^{-1}_{(x,x')} \equiv \mathsf{refl}_{(x,x')},$$

in this case we can proof the claim by reflexivity. Similarly for *(ii)* it suffices to consider the case where $x \equiv y \equiv z$, $x' \equiv y' \equiv z'$, $p \equiv q \equiv \mathsf{refl}_x$ and $r \equiv s \equiv \mathsf{refl}_{x'}$. Using again definition 3.19 and lemma 2.6, by which we have that

$$\mathsf{refl}_{(x,x')} \cdot \mathsf{refl}_{(x,x')} \equiv \mathsf{refl}_{(x,x')},$$

we can proof the claim by reflexivity in this case. □

**Lemma 3.21.** *Assume we are given four types $A, A', B, B' : \mathcal{U}$ together with two functions $f : A \to A'$ and $g : B \to B'$, then it holds that*

$$\prod_{z,z':A\times B} \prod_{p:z=z'} \mathsf{ap}_{(f,g)}(p) = \mathsf{pair}^{=}\big(\mathsf{ap}_{f\circ\pi_1}(p), \mathsf{ap}_{g\circ\pi_2}(p)\big)$$

*Proof.* We use induction on $z$ and $z'$, which splits them into pairs. By path induction it suffices to consider the case where $(x, y) \equiv (x', y')$ and $p \equiv \mathsf{refl}_{(x,y)}$ but since $\mathsf{pair}^{=}(\mathsf{refl}_{f(x)}, \mathsf{refl}_{g(x)}) \equiv \mathsf{refl}_{(f(x),g(x))}$, in this case we can use $\mathsf{refl}_{\mathsf{refl}_{(f(x),g(x))}}$. □

**Lemma 3.22.** *Assume we are given three types $A, B, C : \mathcal{U}$ together with a function $f : A \times B \to C$. For every $x : A$, we define*

$$f_x(y) :\equiv f(x, y)$$

*and for every $y : B$, we define*

$$f_y(x) :\equiv f(x, y).$$

*Then it holds*

$$\prod_{y:B} \prod_{x,x':A} \prod_{p:x=x'} \mathsf{ap}_f\big(\mathsf{pair}^{=}(p, \mathsf{refl}_y)\big) = \mathsf{ap}_{f_y}(p)$$

*and*

$$\prod_{x:A} \prod_{y,y':B} \prod_{q:y=y'} \mathsf{ap}_f\big(\mathsf{pair}^{=}(\mathsf{refl}_x, q)\big) = \mathsf{ap}_{f_x}(q).$$

*Proof.* We check that the above expressions are well typed. For every $y : B$, $x, x' : A$ and $p : x = x'$ we have that

$$\mathsf{pair}^{=}(p, \mathsf{refl}_y) : (x, y) = (x', y)$$

and therefore

$$\mathsf{ap}_f\big(\mathsf{pair}^{=}(p, \mathsf{refl}_y)\big) : f(x, y) = f(x', y).$$

But we also have that

$$\mathsf{ap}_{f_y}(p) : f_y(x) = f_y(x')$$

and by definition

$$f_y(x) \equiv f(x, y)$$

and

$$f_y(x') \equiv f(x', y).$$

The second case is similar. Now, for the first claim, we assume that we are given some $y : B$ and then, by path induction it suffices to consider the case where $x \equiv x'$ and $p \equiv \mathsf{refl}_x$. Since by definition 3.19 we have that

$$\mathsf{pair}^{=}(\mathsf{refl}_x, \mathsf{refl}_y) \equiv \mathsf{refl}_{(x,y)}$$

and by lemma 3.2 we have that

$$\mathsf{ap}_{f_y}(\mathsf{refl}_x) \equiv \mathsf{refl}_{f_y(x)} \equiv \mathsf{refl}_{f(x,y)},$$

in this case we are left to proof that

$$\mathsf{refl}_{f(x,y)} = \mathsf{refl}_{f(x,y)}$$

and we can use $\mathsf{refl}_{\mathsf{refl}_{f(x,y)}}$. The second claim follows in a completely analogue way. □

**Definition 3.23** (Contractability)**.** We say that a type $A : \mathcal{U}$ is contractible, if there exists an inhabitant

$$(a, \mathsf{F}) : \sum_{x:A} \prod_{y:A} x = y :\equiv \mathsf{isContr}(A)$$

and we call $a$ the **center of contraction**.

Another useful lemma is the following. It states that paths for which one of the end-points is allowed to vary and the other one is fixed are homotopic to the constant path in the total space.

**Lemma 3.24.** *Assume we are given some type $A : \mathcal{U}$ together with some fixed $a : A$, then the following type is inhabited:*

$$\mathsf{isContr}(\sum_{x:A} x = a)$$

*The center of contraction is given by $(a, \mathsf{refl}_a)$.*

*Proof.* We aim to inhabit the type

$$\prod_{s:\sum_{x:A} x=a} s = (a, \mathsf{refl}_a)$$

We use induction on $s$, which splits it into $s_1 : A$ and $s_2 : s_1 = a$. Next we note that by theorem 3.26 we have that

$$\mathsf{transport}^{x\mapsto x=a}(s_2, s_2) = s_2^{-1} \cdot s_2$$

and by lemma 3.1 we have that

$$s_2^{-1} \cdot s_2 = \mathsf{refl}_a.$$

The claim therefore follows from the characterization of paths in a $\sum$-space, theorem 3.16. □

**Lemma 3.25.** *Assume we are given two types $A, B : \mathcal{U}$ together with terms*

$$\mathsf{F} : \mathsf{isContr}(A)$$

*and*

$$\mathsf{G} : \mathsf{isContr}(B),$$

*then we have that*

$$A \simeq B.$$

*Proof.* From $\mathsf{F}$ and $\mathsf{G}$ we get points $\pi_1(\mathsf{F}) : A$ and $\pi_1(\mathsf{G}) : B$, hence we can define

$$\big(\pi_1(\mathsf{G})\big)_A : A \to B$$

and

$$\big(\pi_1(\mathsf{F})\big)_B : B \to A.$$

We compute that

$$\big(\pi_1(\mathsf{F})\big)_B \circ \big(\pi_1(\mathsf{G})\big)_A(x) \equiv \pi_1(\mathsf{F})$$

and

$$\big(\pi_1(\mathsf{G})\big)_A \circ \big(\pi_1(\mathsf{F})\big)_B(y) \equiv \pi_1(\mathsf{G})$$

and therefore we have that

$$\pi_2(\mathsf{F}) : \big(\pi_1(\mathsf{F})\big)_B \circ \big(\pi_1(\mathsf{G})\big)_A \sim \mathsf{id}_A$$

and

$$\pi_2(\mathsf{G}) : \big(\pi_1(\mathsf{G})\big)_A \circ \big(\pi_1(\mathsf{F})\big)_B \sim \mathsf{id}_B.$$

□

The following theorem characterizes transport, when the type family states a pointwise equality between functions. We will frequently use it, when constructing equivalences.

**Theorem 3.26.** *Assume we are given two functions $f, g : A \to B$, two points $x, y : A$, a path $p : x =_A y$ and a path $q : f(x) =_B g(x)$. We let $P : A \to \mathcal{U}$ be the type family defined by the equation*

$$P(x) :\equiv f(x) =_B g(x).$$

*Then we have:*

$$\mathsf{transport}^P(p, q) =_{f(a')=g(a')} \big(\mathsf{ap}_f(p)\big)^{-1} \cdot q \cdot \mathsf{ap}_g(p).$$

*Proof.* By path induction it suffices to consider the case where $x \equiv y$ and $p \equiv \mathsf{refl}_x$. Since by lemma 2.7 we have that

$$\mathsf{transport}^P(\mathsf{refl}_x) \equiv \mathsf{id}_{P(x)}$$

and by the computation rules from lemmas 3.2 and 2.5 that

$$\big(\mathsf{ap}_f(p)\big)^{-1} \equiv \mathsf{refl}_{f(x)}$$

and

$$\mathsf{ap}_g(\mathsf{refl}_x) \equiv \mathsf{refl}_{g(x)},$$

in this case we can use lemma 3.1 and proof the claim by $\mathsf{refl}_q$. □

## 4. Pushouts

4.1. **Basics.** In section 2.3 we encountered inductive types. The idea was to specify a set of constructors that generate points in the type under definition. This idea is generalized by allowing constructors to additionally generate paths. We call these types *higher inductive types* (HITs). There is no general theory for HITs yet, so we will focus instead on a specific class, so called *pushout types*, which provide a general enough framework to define many important examples of types with non-trivial homotopy, such as the interval, $n$-spheres and many more.

**Definition 4.1.** Assume we are given three types $A, B, C : \mathcal{U}$ together with two functions $f : C \to A$ and $g : C \to B$. The **pushout** of $(A, B, C, f, g)$ is the higher inductive type $A \sqcup^C B$ generated by

- a function $\mathsf{inl} : A \to A \sqcup^C B$,
- a function $\mathsf{inr} : B \to A \sqcup^C B$, and
- for each $z : C$ a path $\mathsf{Glue}(z) : \mathsf{inl}\big(f(z)\big) = \mathsf{inr}\big(g(z)\big)$.

The recursion principle for pushout types states that, if we are given some type $D : \mathcal{U}$ together with two functions

$$s_l : A \to D$$

and

$$s_r : B \to D$$

and a dependent function

$$\mathsf{Com} : \prod_{z:C} s_l \circ f(z) = s_r \circ g(z),$$

then there is a function

$$s : A \sqcup^C B \to D,$$

such that:

- $s \circ \mathsf{inl} \equiv s_l$
- $s \circ \mathsf{inr} \equiv s_r$
- $\mathsf{ap}_s\big(\mathsf{Glue}(z)\big) = \mathsf{Com}(z)$

The induction principle states that, if we are given a type family

$$P : A \sqcup^C B \to \mathcal{U}$$

together with two dependent functions

$$\mathsf{F}_l : \prod_{x:A} P\big(\mathsf{inl}(x)\big)$$

and

$$\mathsf{F}_r : \prod_{y:B} P\big(\mathsf{inr}(y)\big)$$

and a dependent function

$$\mathsf{G} : \prod_{z:C} \mathsf{transport}^P\Big(\mathsf{Glue}(z), \mathsf{F}\Big(\mathsf{inl}\big(f(z)\big)\Big)\Big) = \mathsf{F}\Big(\mathsf{inr}\big(g(z)\big)\Big),$$

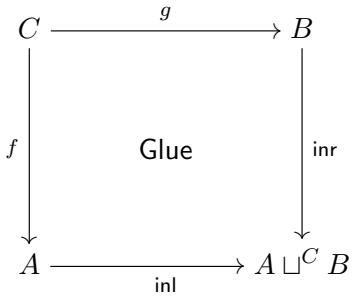
then there is a dependent function

$$\mathsf{F} : \prod_{w:A\sqcup^C B} P(w),$$

such that:

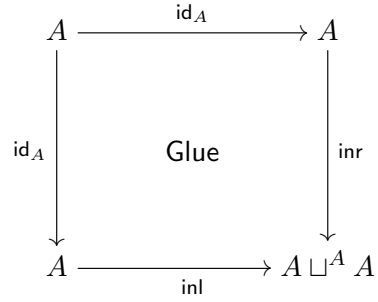
- $\mathsf{F} \circ \mathsf{inl} \equiv \mathsf{F}_l$
- $\mathsf{F} \circ \mathsf{inr} \equiv \mathsf{F}_r$
- $\mathsf{apd}_{\mathsf{F}}\big(\mathsf{Glue}(z)\big) = \mathsf{G}(z)$

In other words, $A \sqcup^C B$ is the disjoint union of $A$ and $B$, where for every $z : C$ we have a path from $\mathsf{inl}\big(f(z)\big)$ to $\mathsf{inr}\big(g(z)\big)$. We depict this information in a square:




Geometrically, pushout types can be thought of as "gluing" the types $A$ and $B$ together along the type $C$ and this is done by making the points of $C$ into paths.

*Remark* 4.2. Firstly we establish that gluing a type $A$ with itself, along itself, does not change its homotopy class. That is, we consider the following pushout square:




Then we have an equivalence:

$$A \sqcup^A A \simeq A$$

*Proof.* We define $\Phi : A \sqcup^A A \to A$ by means of the recursion principle of $A \sqcup^A A$ as follows:

- $\Phi \circ \mathsf{inl} :\equiv \mathsf{id}_A$
- $\Phi \circ \mathsf{inr} :\equiv \mathsf{id}_A$
- $\mathsf{ap}_\Phi\big(\mathsf{Glue}(x)\big) := \mathsf{refl}_x$

We want to proof, that $\Phi$ and $\mathsf{inl}$ are mutually inverse, that is we aim to show that it holds for every $w : A \sqcup^A A$ that:

$$P(w) :\equiv \mathsf{inl} \circ \Phi(w) = w$$

We apply the induction principle of $A \sqcup^A A$ and set:

- $\mathsf{refl}_{\mathsf{inl}(x)} : P\big(\mathsf{inl}(x)\big)$
- $\mathsf{Glue}(x) : P\big(\mathsf{inr}(x)\big)$

Then it is left to show:

$$\mathsf{transport}^P(\mathsf{Glue}(x), \mathsf{refl}_{\mathsf{inl}(x)}) = \mathsf{Glue}(x)$$

Since firstly, by theorem 3.26 we have

$$\mathsf{transport}^P(\mathsf{Glue}(x), \mathsf{refl}_{\mathsf{inl}(x)}) = \mathsf{ap}_{\mathsf{inl}\circ\Phi}\big(\mathsf{Glue}(x)\big)^{-1} \cdot \mathsf{Glue}(x)$$

and secondly, by definition we have $\mathsf{ap}_\Phi\big(\mathsf{Glue}(x)\big) = \mathsf{refl}_a$, we can proof the claim by $\mathsf{refl}_{\mathsf{Glue}(x)}$. Conversely, we let

$$Q(x) :\equiv \Phi \circ \mathsf{inl}(x) = x$$

and then, since

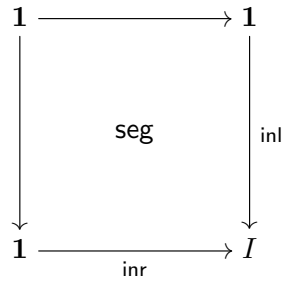
$$\Phi \circ \mathsf{inl}(x) \equiv \mathsf{id}_A(x) \equiv x,$$

we can define $\lambda a.\mathsf{refl}_a : \prod_{(x:A)} Q(x)$. □

*Remark* 4.3 (Sum types). The sum $A + B$ can be defined as the pushout of the square:

$$\begin{array}{ccc} \mathbf{0} & \longrightarrow & B \\ \downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ A & \xrightarrow[\mathsf{inl}]{} & A + B \end{array}$$

**Example 4.4** (The interval)**.** The interval is the pushout of the square:




We introduce a special notation and set:

- $\mathsf{inl}(\star) :\equiv 0_I$
- $\mathsf{inr}(\star) :\equiv 1_I$

Up to equivalence, the interval is just a point, as we present in the following lemma and may therefore not seem very interesting at first sight.

**Lemma 4.5.** *The interval is contractible*

*Proof.* We claim that $0_I$ is the center of contraction. To this end we need to inhabit the type

$$\prod_{i:I} 0_I = i.$$

We apply the induction principle of the interval, where we consider the type family

$$P : I \to \mathcal{U},$$

defined by the equation

$$P(i) :\equiv 0_I = i.$$

Then we can use

$$\mathsf{refl}_{0_I} : P(0_I)$$

and

$$\mathsf{seg} : P(1_I),$$

to proof the claim over the points. Then it is left to show that

$$\mathsf{transport}^P(\mathsf{seg}, \mathsf{refl}_{0_I}) = \mathsf{seg}$$

But by theorem 3.26, this means to show that

$$\mathsf{seg} = \mathsf{seg},$$

which we can inhabit by $\mathsf{refl}_{\mathsf{seg}}$. □

We have seen in lemma 3.10 that from an equality $f = g$ between two functions, we can infer pointwise equality. Once we introduce pushouts and therefore the interval into our theory also the converse is true. We present the proof in the next lemma. This also demonstrates that HITs are interesting from a purely type theoretic point of view.

**Lemma 4.6** (Function extensionality)**.** *Assume we are given two types $A, B : \mathcal{U}$ and two functions $f, g : A \to B$. Then there is a function:*

$$\mathsf{funct Ext} : (f \sim g) \to (f = g)$$

*Proof.* Assume we are given $\mathsf{H} : f \sim g$, then we have for every $x : A$ that

$$\mathsf{H}(x) : f(x) = g(x)$$

and can therefore define a function $\Phi_x : I \to B$ by the recursion principle of the interval, such that:

- $\Phi_x(0_I) :\equiv f(x)$
- $\Phi_x(1_I) :\equiv g(x)$
- $\mathsf{ap}_{\Phi_x}(\mathsf{seg}) :\equiv \mathsf{H}(x)$

But then we can define

$$\Psi :\equiv \lambda i.\lambda x.\Phi_x(i) : I \to A \to B$$

and this gives

$$\mathsf{ap}_{\Psi}(\mathsf{seg}) : (\lambda x.\Phi_x(0_I)) = (\lambda x.\Phi_x(1_I)).$$

Lastly by definition of $\Phi_x$ we have that

$$\lambda x.\Phi_x(0_I) \equiv f$$

and

$$\lambda x.\Phi_x(1_I) \equiv g.$$

□

Next, we prove a dependent version of the previous lemma.

**Lemma 4.7** (Dependent function extensionality)**.** *Assume we are given a type $A : \mathcal{U}$ together with a type family $P : A \to \mathcal{U}$ and two dependent functions*

$\mathsf{F}, \mathsf{G} : \prod_{(x:A)} P(x)$. *Then there is a dependent function:*

$$\mathsf{FunctExt} : \Big(\prod_{x:A} \mathsf{F}(x) =_{P(x)} \mathsf{G}(x)\Big) \to (\mathsf{F} = \mathsf{G})$$

*Proof.* Assume we are given $\mathsf{H} : \prod_{(x:A)} \mathsf{F}(x) =_{P(x)} \mathsf{G}(x)$, then we have for every $x : A$ that

$$\mathsf{H}(x) : \mathsf{F}(x) =_{P(x)} \mathsf{G}(x)$$

and can therefore define a function $\Phi_x : I \to P(x)$ by the recursion principle of the interval, such that:

- $\Phi_x(0_I) :\equiv \mathsf{F}(x)$
- $\Phi_x(1_I) :\equiv \mathsf{G}(x)$
- $\mathsf{ap}_{\Phi_x}(\mathsf{seg}) \coloneqq \mathsf{H}(x)$

But then we can define

$$\Psi :\equiv \lambda i.\lambda x.\Phi_x(i) : I \to \prod_{x:A} P(x)$$

and this gives

$$\mathsf{ap}_{\Psi}(\mathsf{seg}) : \lambda x.\Phi(0_I) =_{\prod_{(x:A)} P(x)} \lambda x.\Phi(1_I).$$

Lastly by definition of $\Phi_x$ we have that

$$\lambda x.\Phi_x(0_I) \equiv \mathsf{F}$$

and

$$\lambda x.\Phi_x(1_I) \equiv \mathsf{G}.$$

□

**Corollary 4.8** (Weak function extensionality)**.** *Assume that we are given a type $A : \mathcal{U}$ together with a type family $P : A \to \mathcal{U}$ and an inhabitant*

$$\mathsf{H} : \prod_{x:A} \mathsf{isContr}\big(P(x)\big),$$

*then it also holds that*

$$\mathsf{isContr}\Big(\prod_{x:A} P(x)\Big).$$

*This is also called* ***weak function extensionality****.*

*Proof.* Since by assumption we are given some

$$\mathsf{H} : \prod_{x:A} \mathsf{isContr}\big(P(x)\big),$$

we have in particular for every $x : A$ inhabitants

$$\pi_1\big(\mathsf{H}(x)\big) : P(x)$$

and

$$\pi_2\big(\mathsf{H}(x)\big) : \prod_{y:P(x)} \pi_1\big(\mathsf{H}(x)\big) = y.$$

We then define for every $\mathsf{F} : \prod_{(x:A)} P(x)$ an dependent function $\mathsf{G}$, by the defining equation

$$\mathsf{G}(x) :\equiv \pi_2\big(\mathsf{H}(x)\big)\big(\mathsf{F}(x)\big),$$

which then has type

$$\prod_{x:A} \pi_1\big(\mathsf{H}(x)\big) =_{P(x)} \mathsf{F}(x).$$

Lastly we define

$$\lambda x.\pi_1\big(\mathsf{H}(x)\big) : \prod_{x:A} P(x)$$

and then, by lemma 4.7, we have that

$$\mathsf{FunctExt}\big(\mathsf{G}(x)\big) : \lambda x.\pi_1\big(\mathsf{H}(x)\big) = \mathsf{F}$$

Since $\mathsf{F}$ was chosen arbitrary, this shows the claim. □

We note that from lemma 4.8, one can proof that happly is an equivalence as has been done, e.g. in chapter 4.9 of the HoTT book [10]. For our purposes, the following lemma will suffice, which provides a direct proof that happly is left inverse to functExt.

**Lemma 4.9.** *The function* happly *from lemma 3.10 is left inverse to* functExt *from lemma 4.6.*

*Proof.* We aim to show that

$$\prod_{f,g:A\to B} \prod_{H:f\sim g} \mathsf{happly}\big(\mathsf{functExt}(\mathsf{H})\big) = \mathsf{H}.$$

So assume we start of with two functions $f, g : A \to B$ together with some $\mathsf{H} : f \sim g$. Following the procedure from lemma 4.6, we get a function

$$\Psi :\equiv \lambda i.\lambda x.\Phi_x(i) : I \to A \to B,$$

where $\Phi$ is defined by recursion on the interval, such that:

- $\Phi_x(0_I) :\equiv f(x)$
- $\Phi_x(1_I) :\equiv g(x)$
- $\mathsf{ap}_{\Phi_x}(\mathsf{seg}) \coloneqq \mathsf{H}(x)$

In particular, we then have a proof that $f$ and $g$ are equal:

$$\mathsf{ap}_{\Psi}(\mathsf{seg}) : f = g$$

Next, by lemma 3.11 we have for every $p : f = g$ that

$$\prod_{x:A} \mathsf{happly}(p)(x) = \mathsf{ap}_{\mathsf{eval}(x)}(p)$$

and hence, by dependent function extensionality (lemma 4.7) that

$$\mathsf{happly}\big(\mathsf{ap}_{\Psi}(\mathsf{seg})\big) = \lambda x.\mathsf{ap}_{\mathsf{eval}(x)}\big(\mathsf{ap}_{\Psi}(\mathsf{seg})\big) = \lambda x.\mathsf{ap}_{\mathsf{eval}(x)\circ\Psi}(\mathsf{seg})$$

and since we can compute

$$\lambda i.\big(\mathsf{eval}(x) \circ \Psi(i)\big) \equiv \lambda i.\big(\mathsf{eval}(x, \Psi(i))\big) \equiv \lambda i.\Psi(i, x) \equiv \lambda i.\Phi_x(i) \equiv \Phi_x$$

we get that

$$\lambda x.\mathsf{ap}_{\mathsf{eval}(x)\circ\Psi}(\mathsf{seg}) = \lambda x.\mathsf{ap}_{\Phi_x}(\mathsf{seg}).$$

Lastly by definition of $\Phi$ we have

$$\lambda x.\mathsf{ap}_{\Phi_x}(\mathsf{seg}) \equiv \mathsf{H}.$$

□

The interval further enforces the types as spaces interpretation. In topology a path is defined as a continuous function from the interval, in homotopy type theory we have the following lemma.

**Lemma 4.10.** *Assume we are given a type $A : \mathcal{U}$, then we have:*

$$I \to A \simeq \sum_{(x,y):A\times A} x = y$$

*Proof.* Since for some given $f : I \to A$ we have

$$\mathsf{ap}_f(\mathsf{seg}) : f(0_I) = f(1_I),$$

we can set

$$\big((f(0_I), f(1_I)), \mathsf{ap}_f(\mathsf{seg})\big) : \sum_{(x,y):A\times A} x = y.$$

If we are given $s : \sum_{(x,y):A\times A} x = y$, we can define a function

$$f : I \to A$$

by recursion on the interval. To this end we set:

- $f(0_I) :\equiv \pi_1\big(\pi_1(s)\big)$
- $f(1_I) :\equiv \pi_2\big(\pi_1(s)\big)$
- $\mathsf{ap}_f(\mathsf{seg}) \coloneqq \pi_2(s)$

Next we want to show that the above functions define an equivalence. So assume we start of with $f : I \to A$. The above method gives a term

$$\Big(\big(f(0_I), f(1_I)\big), \mathsf{ap}_f(\mathsf{seg})\Big) : \sum_{(x,y):A\times A} x = y$$

and from this we define a function $g : I \to A$ by recursion, such that:

(i) $g(0_I) :\equiv f(0_I)$

(ii) $g(1_I) :\equiv f(1_I)$

(iii) $\mathsf{ap}_g(\mathsf{seg}) \coloneqq \mathsf{ap}_f(\mathsf{seg})$

Since we have

$$\mathsf{refl}_{f(0_I)} : g(0_I) = f(0_I)$$

and

$$\mathsf{refl}_{f(1_I)} : g(1_I) = f(1_I)$$

and lastly, since by theorem 3.26 we have that

$$\mathsf{transport}^{i\mapsto f(i)=g(i)}(\mathsf{seg}, \mathsf{refl}_{f(0_I)}) = \mathsf{refl}_{f(1_I)}$$

equals

$$\mathsf{ap}_f(\mathsf{seg})^{-1} \cdot \mathsf{ap}_g(\mathsf{seg}) = \mathsf{refl}_{f(1_I)},$$

which is proven by ((iii)), we get $f \sim g$ by the induction principle of the interval. Function extensionality gives $f = g$ and therefore shows that we end up with what we started with. Conversely assume we start of with

$s : \sum_{(x,y):A\times A} x = y$. Then, firstly we get a function

$$f : I \to A,$$

such that:

- $f(0_I) :\equiv \pi_1\big(\pi_1(s)\big)$
- $f(1_I) :\equiv \pi_2\big(\pi_1(s)\big)$
- $\mathsf{ap}_f(\mathsf{seg}) :\equiv \pi_2(s)$

This function is then mapped to

$$\Big(\big(f(0_I), f(1_I)\big), \mathsf{ap}_f(\mathsf{seg})\Big) : \sum_{(x,y):A\times A} x = y$$

and we aim to show

$$s =_{\sum_{(x,y):A\times A} x=y} \Big(\big(f(0_I), f(1_I)\big), \mathsf{ap}_f(\mathsf{seg})\Big).$$

By theorem 3.16, it suffices to provide a path

$$p : \big(f(0_I), f(1_I)\big) = \pi_1(s),$$

such that

$$\mathsf{transport}^{(x,y)\mapsto x=y}\big(p, \mathsf{ap}_f(\mathsf{seg})\big) = \pi_2(s).$$

But we have by definition of $f$ that

$$\big(f(0_I), f(1_I)\big) \equiv \big(\pi_1\big(\pi_1(s)\big), \pi_2\big(\pi_1(s)\big)\big)$$

and since by lemma 2.10 we have

$$\big(\pi_1\big(\pi_1(s)\big), \pi_2\big(\pi_1(s)\big)\big) = \pi_1(s),$$

we can set $p :\equiv \mathsf{refl}_{\pi_1(s)}$. Then, since

$$\mathsf{transport}^{(x,y)\mapsto x=y}(\mathsf{refl}_{\pi_1(s)}) \equiv \mathsf{id}_{x=y},$$

we are left to show that

$$\mathsf{ap}_f(\mathsf{seg}) = \pi_2(s),$$

which is inhabited by the definition of $f$. □

For the next examples we need the following definitions.

**Definition 4.11.** We define the type of **pointed types** as

$$\mathcal{U}^* :\equiv \sum_{A:\mathcal{U}} A.$$

If we are given an inhabitant $(A, *_A) : \mathcal{U}^*$, we say the type $A : \mathcal{U}$ is **pointed**.

*Remark* 4.12. Usually we will just speak of a pointed type $A : \mathcal{U}$ and implicitly assume that $*_A : A$ has been chosen.

Later on we will need also the following.

**Definition 4.13.** Assume we are given two pointed types $(A, *_A) : \mathcal{U}^*$ and $(B, *_B) : \mathcal{U}^*$. We define the type of **pointed maps**:

$$\mathsf{Map}^*(A, B) :\equiv \sum_{f:A\to B} *_B =_B f(*_A)$$

If we are given an inhabitant $(f, p_f) : \mathsf{Map}^*(A, B)$, we say that $f$ is **base-point preserving** or $f$ is a **pointed function** and call $p_f$ a proof that $f$ is **base-point preserving**. We make $\mathsf{Map}^*(A, B)$ a pointed type by setting $\big((*_B)_A, \mathsf{refl}_{*_B}\big)$ as its base-point.

*Remark* 4.14. Assume we want to show that two pointed maps

$$(f, p_f), (g, p_g) : \mathsf{Map}^*(A, B)$$

are equal. By theorem 3.16, this means that we need to give a path

$$q : f = g,$$

such that

$$\mathsf{transport}^{h\mapsto *_B = h(*_A)}(q, p_f) = p_g.$$

By theorem 3.26, this means to show that

$$p_f \centerdot \mathsf{ap}_{\mathsf{eval}(*_A)}(q) = p_g$$

and using lemma 3.11, this is equivalent to

$$p_f \centerdot \mathsf{happly}(q)(*_A) = p_g.$$

In particular, if we aim to show that some given pointed function $(f, p_f) : \mathsf{Map}^*(A, B)$ is equal to the base-point $((*_B)_A, \mathsf{refl}_{*_B})$ in $\mathsf{Map}^*(A, B)$, we can concatenate the above equation with $\mathsf{happly}(q)(*_A)^{-1}$ from the right and this

gives

$$p_f = \mathsf{happly}(q)(*_A)^{-1}.$$

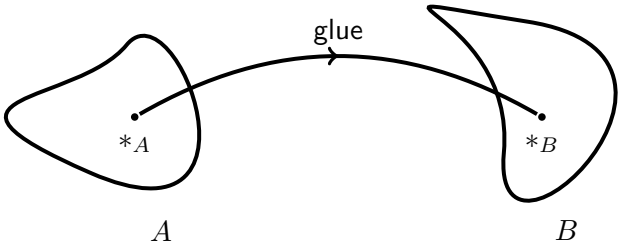
In words this reads that the proof that $f$ is base-point preserving is equal to the proof that the constant function $(*_B)_A$ is equal to $f$, evaluated at the base-point. The same argumentation gives that in order to show that $(f, p_f) : \mathsf{Map}^*(A, A)$ is equal to $(\mathsf{id}_A, \mathsf{refl}_{*_A})$, we need a

$$q : f = \mathsf{id}_A,$$

such that

$$p_f = \mathsf{happly}(q)(*_A)^{-1}.$$

**Example 4.15** (Wedge sum)**.** The wedge sum $A \vee B$ is a pushout of two pointed types $A$ and $B$. It can be thought of as the sum $A + B$ of $A$ and $B$, where the base-points $*_A$ and $*_B$ are identified by a path $\mathsf{glue}$.



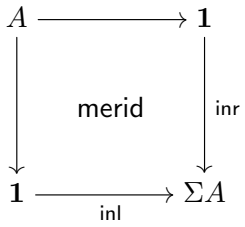

*Remark* 4.16. We write $*_A$ over the arrows to denote the constant function $\lambda \star .*_A$. For functions out of the unit $\mathbf{1} \to A$ to some pointed type $A$, if we do not write anything over the arrows, we always mean the constant function $(*_A)_{\mathbf{1}}$.

The wedge sum $A \vee B$ of two pointed types $A$ and $B$ is the pushout of the square:

$$\begin{array}{ccc} \mathbf{1} & \xrightarrow{*_B} & B \\ {\scriptstyle *_A}\downarrow & \mathsf{Glue} & \downarrow{\scriptstyle \mathsf{inr}} \\ A & \xrightarrow[\mathsf{inl}]{} & A \vee B \end{array}$$

We make $A \vee B$ a pointed type, by setting $*_{A \vee B} :\equiv \mathsf{inl}(*_A)$.

**Example 4.17** (Suspension)**.** Te suspension is the pushout of the square:




We make the suspension a pointed type by choosing $\mathsf{inl}(\star)$ as its base-point. We also introduce a special notation and set:

- $\mathsf{N} :\equiv \mathsf{inl}(\star)$
- $\mathsf{S} :\equiv \mathsf{inr}(\star)$

**Definition 4.18.** The type with just two points is the suspension of the empty type:

$$\mathbf{2} :\equiv \Sigma\mathbf{0}$$

We introduce a special notation and set

- $0_{\mathbf{2}} :\equiv \mathsf{inl}(\star)$ and
- $1_{\mathbf{2}} :\equiv \mathsf{inr}(\star)$.

**Definition 4.19** (The circle)**.** The circle is the suspension of the space with two points:

$$\mathbb{S}^1 :\equiv \Sigma\mathbf{2}$$

The following lemma shows that the circle has the universal property that we would expect.

**Lemma 4.20.** *Assume we a given a pointed type $A : \mathcal{U}$, then it holds that*

$$\mathsf{Map}^*(\mathbb{S}^1, A) \simeq (*_A =_A *_A)$$

*Proof.* Assume that we are given a base-point preserving function

$$(f, p_f) : \mathsf{Map}^*(\mathbb{S}^1, A).$$

Then we have in particular that

$$p_f : *_A = f(\mathsf{N}),$$

$$\mathsf{ap}_f\big(\mathsf{merid}(1_{\mathbf{2}})\big) : f(\mathsf{N}) =_A f(\mathsf{S})$$

and

$$\mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big)^{-1} : f(\mathsf{S}) =_A f(\mathsf{N}).$$

Hence we can define

$$p_f \mathbin{\centerdot} \mathsf{ap}_f\big(\mathsf{merid}(1_{\mathbf{2}})\big) \mathbin{\centerdot} \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big)^{-1} \mathbin{\centerdot} p_f^{-1} : (*_A = *_A).$$

Conversely, assume we are given a loop $l : (*_A = *_A)$, then we can define a function $g : \mathbb{S}^1 \to A$ by the recursion principle of the suspension. To this end we define:

- $g(\mathsf{N}) :\equiv g(\mathsf{S}) :\equiv *_A$
- $\mathsf{ap}_g\big(\mathsf{merid}(0_{\mathbf{2}})\big) \coloneqq \mathsf{refl}_{*_A}$
- $\mathsf{ap}_g\big(\mathsf{merid}(1_{\mathbf{2}})\big) \coloneqq l$

Then we have that $g$ is base-point preserving by $\mathsf{refl}_{*_A}$. We want to show that these functions are mutually inverse. So assume we start of with $f : \mathsf{Map}^*(\mathbb{S}^1, A)$. We have to show that the function $g$, defined by

- $g(\mathsf{N}) :\equiv g(\mathsf{S}) :\equiv *_A$ ,
- $\mathsf{ap}_g\big(\mathsf{merid}(0_{\mathbf{2}})\big) \coloneqq \mathsf{refl}_{*_A}$ and
- $\mathsf{ap}_g\big(\mathsf{merid}(1_{\mathbf{2}})\big) \coloneqq p_f \mathbin{\centerdot} \mathsf{ap}_f\big(\mathsf{merid}(1_{\mathbf{2}})\big) \mathbin{\centerdot} \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big)^{-1} \mathbin{\centerdot} p_f^{-1}$

is equal to $(f, p_f)$. So we consider the type family

$$P : \mathbb{S}^1 \to \mathcal{U},$$

defined by the equation

$$P(s) :\equiv g(s) = f(s).$$

If $s \equiv \mathsf{N}$, we compute $P(\mathsf{N}) \equiv (*_A = f(\mathsf{N}))$ and hence we can define

$$p_f : P(\mathsf{N}).$$

If $s \equiv \mathsf{S}$, we compute $P(\mathsf{S}) \equiv (*_A = f(\mathsf{S}))$ and hence we can define

$$p_f \mathbin{\centerdot} \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big) : P(\mathsf{N}).$$

It is left to show for every $b : \mathbf{2}$ that

$$\mathsf{transport}^P\big(\mathsf{merid}(b), p_f\big) = p_f \mathbin{\centerdot} \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big).$$

By theorem 3.26, this means to show for every $b : \mathbf{2}$ that

$$\mathsf{ap}_g\big(\mathsf{merid}(b)\big)^{-1} \cdot p_f \cdot \mathsf{ap}_f\big(\mathsf{merid}(b)\big) = p_f \cdot \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big).$$

If $b \equiv 0_{\mathbf{2}}$, we use the definition of $g$ and then we have to show that

$$p_f \cdot \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big) = p_f \cdot \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big),$$

which is inhabited by $\mathsf{refl}_{\mathsf{refl}_{p_f \cdot \mathsf{ap}_f(\mathsf{merid}(0_{\mathbf{2}}))}}$. Similarly, if $b \equiv 1_{\mathbf{2}}$, we have to show

$$p_f \cdot \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big) \cdot \mathsf{ap}_f\big(\mathsf{merid}(1_{\mathbf{2}})\big)^{-1} \cdot p_f^{-1} \cdot p_f \cdot \mathsf{ap}_f\big(\mathsf{merid}(1_{\mathbf{2}})\big) = \\ = p_f \cdot \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big),$$

which, after canceling inverses, is again inhabited by $\mathsf{refl}_{\mathsf{refl}_{p_f \cdot \mathsf{ap}_f(\mathsf{merid}(0_{\mathbf{2}}))}}$. By the induction principle of the circle, this gives an inhabitant

$$\mathsf{H} : g \sim f,$$

such that, over the points we have:

- $\mathsf{H}(\mathsf{N}) \equiv p_f$
- $\mathsf{H}(\mathsf{S}) \equiv p_f \cdot \mathsf{ap}_f\big(\mathsf{merid}(0_{\mathbf{2}})\big)$

Function extensionality gives:

$$\mathsf{funct Ext}(\mathsf{H}) : g = f$$

Now we use theorem 3.16 to show that

$$(f, p_f) =_{\mathsf{Map}^*(\mathbb{S}^1, A)} (g, \mathsf{refl}_{*_A}).$$

This requires us to show that

$$\mathsf{transport}^{h \mapsto *_A = h(\mathsf{N})}(\mathsf{funct Ext}(\mathsf{H}), p_g) = p_f.$$

By theorem 3.26, lemma 3.11 and lemma 4.9, this means to show that

$$\mathsf{H}(\mathsf{N}) = p_f,$$

but since $\mathsf{H}(\mathsf{N}) \equiv p_f$, we can inhabit this type by $\mathsf{refl}_{p_f}$.
Conversely, assume we start of with a loop $l : *_A = *_A$ and define:

- $g(\mathsf{N}) :\equiv g(\mathsf{S}) :\equiv *_A$
- $\mathsf{ap}_g\big(\mathsf{merid}(0_{\mathbf{2}})\big) :\equiv \mathsf{refl}_{*_A}$
- $\mathsf{ap}_g\big(\mathsf{merid}(1_{\mathbf{2}})\big) :\equiv l$

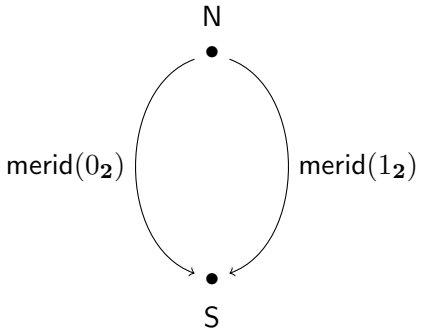
We want to show that

$$l = p_g \cdot \mathsf{ap}_g\big(\mathsf{merid}(1_{\mathbf{2}})\big) \cdot \mathsf{ap}_g\big(\mathsf{merid}(0_{\mathbf{2}})\big)^{-1} \cdot p_g^{-1}.$$

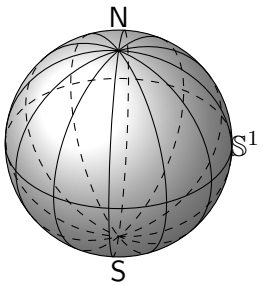
But $p_g \equiv \mathsf{refl}_{*_A}$ and by checking the above definitions, we see that we can use $\mathsf{refl}_l$ to inhabit this type. □

We note that one can use suspensions to define general $n$-spheres $\mathbb{S}^n$ for some natural number $n$, where, as indicated above, the type with two inhabitants plays the role of the "0-sphere", in accordance to what one would define in topology. We can depict the situation for the first few steps. The suspension of the empty type $\mathbf{2}$ simply consists of two points:


By adding paths between those two points, we get the circle:



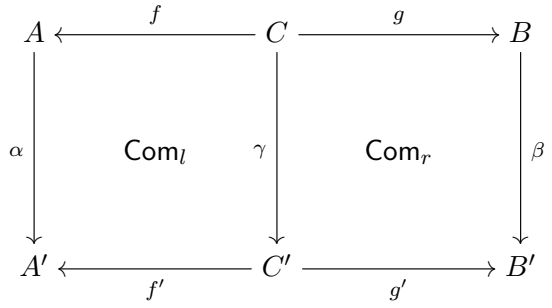

Making the points of the circle into paths, we get a sphere:




4.2. **Pushout functions.** Next, we reduce the task to define a function between pushout types $A \sqcup^C B \to A' \sqcup^{C'} B'$ to giving functions $\alpha : A \to A'$,

$\beta : B \to B'$ and $\gamma : C \to C'$. This will be done in such a way that if $\alpha$, $\beta$ and $\gamma$ are equivalences, then so is the resulting function.

**Lemma 4.21.** *Assume, we are given types $A, B, C, A', B', C' : \mathcal{U}$, functions $f : C \to A$, $g : C \to B$, $f' : C' \to A'$, $g' : C' \to B'$ and additionally two **commutative squares**, i.e. functions $\alpha : A \to A'$, $\beta : B \to B'$ and $\gamma : C \to C'$ together with two dependent functions $\mathsf{Com}_l : (f' \circ \gamma) \sim (\alpha \circ f)$ and $\mathsf{Com}_r : (g' \circ \gamma) \sim (\beta \circ g)$, as depicted below:*




*Then, there is a **pushout function** $\alpha \sqcup^{\gamma} \beta : A \sqcup^{C} B \to A' \sqcup^{C'} B'$, such that:*

- $(\alpha \sqcup^{\gamma} \beta) \circ \mathsf{inl} \equiv \mathsf{inl}' \circ \alpha$
- $(\alpha \sqcup^{\gamma} \beta) \circ \mathsf{inr} \equiv \mathsf{inr}' \circ \beta$
- $\mathsf{ap}_{\alpha \sqcup^{\gamma} \beta}\big(\mathsf{Glue}(z)\big) = \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)$

*Proof.* We apply the recursion principle of $A \sqcup^{C} B$. To this end we check:

- $\mathsf{Com}_l(z) : f' \circ \gamma(z) = \alpha \circ f(z)$
- $\mathsf{Com}_r(z) : g' \circ \gamma(z) = \beta \circ g(z)$
- $\mathsf{Glue}'\big(\gamma(z)\big) : \mathsf{inl}' \circ f' \circ \gamma(z) = \mathsf{inr}' \circ g' \circ \gamma(z)$

This shows that

$$\mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)$$

has the appropriate type

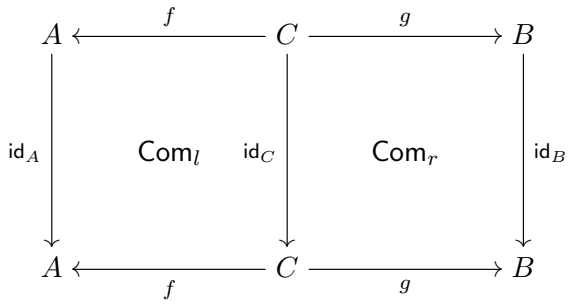
$$\mathsf{inl}' \circ \alpha \circ f(z) = \mathsf{inr}' \circ \beta \circ g(z)$$

to define a function by the recursion principle for pushout types with computation rules stated in the lemma. □

*Remark* 4.22. When we say that two squares as in lemma 4.21 commute, we always mean that we are given types $A, B, C, A', B', C' : \mathcal{U}$, functions

$f : C \to A$, $g : C \to B$, $f' : C' \to A'$, $g' : C' \to B'$, $\alpha : A \to A'$, $\beta : B \to B'$, $\gamma : C \to C'$ $\mathsf{Com}_l : (f' \circ \gamma) \sim (\alpha \circ f)$ and dependent functions $\mathsf{Com}_l : (f' \circ \gamma) \sim (\alpha \circ f)$ and $\mathsf{Com}_r : (g' \circ \gamma) \sim (\beta \circ g)$ and we will generally not write this down explicitly.

*Remark* 4.23. Assume we are given types $A, B, C : \mathcal{U}$ and we let $\alpha$ be the pushout function of the commutative squares:




One might guess that $\alpha \sim \mathsf{id}_{A \sqcup^C B}$, so let us try to proof it. We define a type family

$$P : A \sqcup^C B \to \mathcal{U},$$

by the equation

$$P(w) :\equiv \alpha(w) = w.$$

We compute

$$\alpha\big(\mathsf{inl}(x)\big) \equiv \mathsf{inl}(x)$$

and therefore

$$\mathsf{refl}_{\mathsf{inl}(x)} : P\big(\mathsf{inl}(x)\big)$$

and similarly

$$\alpha\big(\mathsf{inr}(y)\big) \equiv \mathsf{inl}(y)$$

and therefore

$$\mathsf{refl}_{\mathsf{inr}(y)} : P\big(\mathsf{inr}(y)\big).$$

Then it is left for us to show that

$$\mathsf{transport}^{w \mapsto \alpha(w) = w}(\mathsf{Glue}(z), \mathsf{refl}_{\mathsf{inl}(x)}) = \mathsf{refl}_{(\mathsf{inr}(y)}$$

and by theorem 3.26 this means to show that

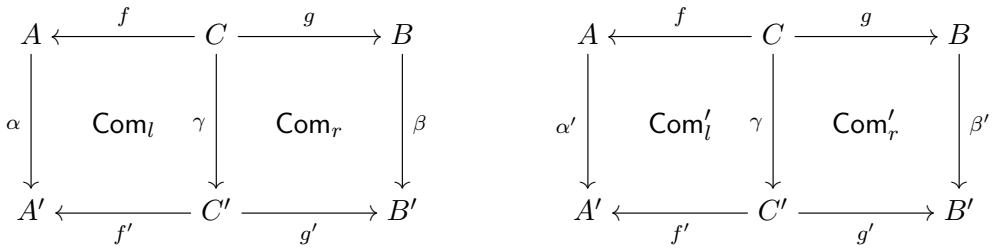
$$\mathsf{ap}_\alpha\big(\mathsf{Glue}(z)\big)^{-1} \cdot \mathsf{Glue}(z) = \mathsf{refl}_{(\mathsf{inr}(y)}.$$

Since $\alpha$ is a pushout function, we check the computation rules from lemma 4.21, by which we have:

$$\mathsf{ap}_{\alpha}\big(\mathsf{Glue}(z)\big) = \mathsf{ap}_{\mathsf{inl}}(\mathsf{Com}_l) \cdot \mathsf{Glue}(z) \cdot \mathsf{ap}_{\mathsf{inr}}\big(\mathsf{Glue}(z)\big)$$

Now we are stuck: to proceed, we need to know what the proofs of commutativity are. We see that it is generally not enough to assume that $\alpha \sim \alpha'$, $\beta \sim \beta'$ and $\gamma \sim \gamma'$, to conclude that $(\alpha \sqcup^{\gamma} \beta) \sim (\alpha' \sqcup^{\gamma'} \beta')$. We can however establish sufficient conditions under which it is possible to do this and this will be done next. We note that we can still say something about the $\alpha$ from above, namely that it is an equivalence. We will prove this in lemma 4.28.

**Lemma 4.24.** *Assume we are given* $\mathsf{H} : \alpha \sim \alpha'$, $\mathsf{I}\beta \sim \beta'$ *together with commutative squares:*




*If we have additionally for every* $z : C$ *proofs that*

- $\mathsf{Com}'_l(z) = \mathsf{Com}_l(z) \cdot \mathsf{H}\big(f(z)\big)$ *and*
- $\mathsf{Com}'_r(z) = \mathsf{Com}_r(z) \cdot \mathsf{I}\big(g(z)\big)$ ,

*then it holds for the pushout functions that*

$$\alpha \sqcup^{\gamma} \beta \sim \alpha' \sqcup^{\gamma} \beta'.$$

*Proof.* We want to apply the induction principle of $A \sqcup^{C} B$ to proof the claim. So let us consider the type family

$$P : A \sqcup^{C} B \to \mathcal{U},$$

defined by the equation

$$P(w) :\equiv (\alpha \sqcup^{\gamma} \beta)(w) = (\alpha' \sqcup^{\gamma} \beta')(w).$$

Then we have that

$$\mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{H}(x)\big) : P\big(\mathsf{inl}(x)\big)$$

and

$$\mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{I}(y)\big) : P\big(\mathsf{inr}(y)\big)$$

It is left for us to show:

$$\mathsf{transport}^P\Big(\mathsf{Glue}(z), \mathsf{ap}_{\mathsf{inl}'}\Big(\mathsf{H}\big(f(z)\big)\Big)\Big) = \mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{I}\big(g(z)\big)\Big)$$

By theorem 3.26 this means to show:

$$\mathsf{ap}_{\alpha \sqcup^\gamma \beta}\big(\mathsf{Glue}(z)\big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}'}\Big(\mathsf{H}\big(f(z)\big)\Big) \cdot \mathsf{ap}_{\alpha' \sqcup^\gamma \beta'}\big(\mathsf{Glue}(z)\big) = \mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{I}\big(g(z)\big)\Big) \tag{4}$$

By definition of pushout functions, given in lemma 4.21, we have that

$$\mathsf{ap}_{\alpha \sqcup^\gamma \beta}\big(\mathsf{Glue}(z)\big)^{-1} = \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)$$

and

$$\mathsf{ap}_{\alpha' \sqcup^\gamma \beta'}\big(\mathsf{Glue}(z)\big) = \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}'_l(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}'_r(z)\big).$$

By assumption we have that

- $\mathsf{Com}'_l(z) = \mathsf{Com}_l(z) \cdot \mathsf{H}\big(f(z)\big)$ and
- $\mathsf{Com}'_r(z) = \mathsf{Com}_r(z) \cdot \mathsf{I}\big(g(z)\big)$

and therefore, from lemma 3.3, we get that

$$\mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}'_l(z)\big)^{-1}$$

is equal to

$$\mathsf{ap}_{\mathsf{inl}'}\Big(\mathsf{H}\big(f(z)\big)\Big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)^{-1}$$

and

$$\mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}'_r(z)\big)$$

is equal to

$$\mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{I}\big(g(z)\big)\Big).$$

Therefore, we conclude that

$$\mathsf{ap}_{\alpha' \sqcup^\gamma \beta'}\big(\mathsf{Glue}(z)\big) = \mathsf{ap}_{\mathsf{inl}'}\Big(\mathsf{H}\big(f(z)\big)\Big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot$$
$$\cdot \mathsf{Glue}'\big(\gamma(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{I}\big(g(z)\big)\Big).$$

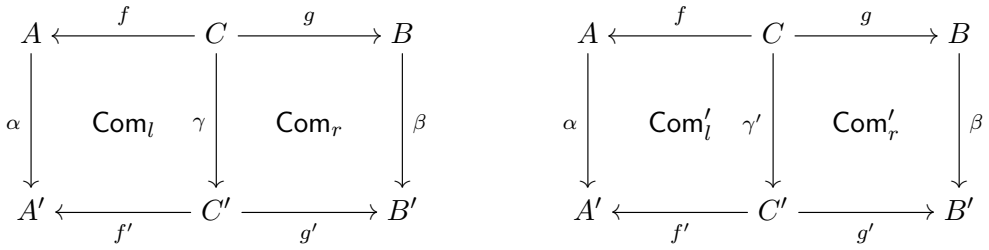
Plugging everything into the left side of (4) gives:

$$\mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big) \cdot$$

$$\cdot \mathsf{ap}_{\mathsf{inl}'}\Big(\mathsf{H}\big(f(z)\big)\Big) \cdot \mathsf{ap}_{\mathsf{inl}'}\Big(\mathsf{H}\big(f(z)\big)\Big)^{-1} \cdot$$
$$\cdot \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{I}\big(g(z)\big)\Big)$$

We can now cancel out inverses of $\mathsf{ap}_{\mathsf{inl}'}\Big(\mathsf{H}\big(f(z)\big)\Big)$, $\mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}'_l(z)\big)$, $\mathsf{Glue}'\big(\gamma(z)\big)$ and $\mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}'_r(z)\big)$ and this gives us that the left hand side of (4) equals $\mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{I}\big(g(z)\big)\Big)$. □

**Lemma 4.25.** *Assume we are given* $\mathsf{H} : \gamma \sim \gamma'$ *and commutative squares:*




*If we have additionally for every* $z : C$ *proofs that*

- $\mathsf{Com}'_l(z) = \mathsf{ap}_{f'}\big(\mathsf{H}(z)\big)^{-1} \cdot \mathsf{Com}_l(z)$ *and*
- $\mathsf{Com}'_r(z) = \mathsf{ap}_{g'}\big(\mathsf{H}(z)\big)^{-1} \cdot \mathsf{Com}_r(z)$ ,

*then it holds for the pushout functions that*

$$\alpha \sqcup^{\gamma} \beta \sim \alpha \sqcup^{\gamma'} \beta.$$

*Proof.* We want to apply the induction principle of $A \sqcup^C B$ to proof the claim. So let us consider the type family

$$P : A \sqcup^C B \to \mathcal{U},$$

defined by the equation

$$P(w) :\equiv (\alpha \sqcup^{\gamma} \beta)(w) = (\alpha \sqcup^{\gamma'} \beta)(w).$$

Then we have that

$$\mathsf{refl}_{\mathsf{inl}'(\alpha(x))} : P\big(\mathsf{inl}(x)\big)$$

and

$$\mathsf{refl}_{\mathsf{inr}'(\beta(y))} : P\big(\mathsf{inr}(y)\big).$$

It is left for us to show that:

$$\mathsf{transport}^P(\mathsf{Glue}(z), \mathsf{refl}_{\mathsf{inl}'(\alpha(x))}) = \mathsf{refl}_{\mathsf{inr}'(\beta(y))}$$

By theorem 3.26 this means to show:

$$\mathsf{ap}_{\alpha\sqcup^{\gamma}\beta}\big(\mathsf{Glue}(z)\big)^{-1} \cdot \mathsf{ap}_{\alpha\sqcup^{\gamma'}\beta}\big(\mathsf{Glue}(z)\big) = \mathsf{refl}_{\mathsf{inr}'(\beta(y))} \tag{5}$$

By definition of pushout functions, given in lemma 4.21 we have that

$$\mathsf{ap}_{\alpha\sqcup^{\gamma}\beta}\big(\mathsf{Glue}(z)\big)^{-1} = \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)$$

and

$$\mathsf{ap}_{\alpha\sqcup^{\gamma'}\beta}\big(\mathsf{Glue}(z)\big) = \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}'_l(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma'(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}'_r(z)\big)$$

By the first assumption we have that

$$\mathsf{Com}'_l(z) = \mathsf{ap}_{f'}\big(\mathsf{H}(z)\big)^{-1} \cdot \mathsf{Com}_l(z)$$

and therefore, from lemma 3.3, we get that the term $\mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}'_l(z)\big)^{-1}$ is equal to

$$\mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}'\circ f'}\big(\mathsf{H}(z)\big).$$

By the second assumption we have that

$$\mathsf{Com}'_r(z) = \mathsf{ap}_{g'}\big(\mathsf{H}(z)\big)^{-1} \cdot \mathsf{Com}_r(z)$$

and therefore, from lemma 3.3, we get that the term $\mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}'_r(z)\big)$ is equal to

$$\mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{ap}_{g'}\big(\mathsf{H}(z)\big)\Big)^{-1} \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big).$$

Plugging everything in the left side of (5) gives:

$$\begin{aligned}
&\mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big)^{-1} \cdot \\
&\cdot \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big) \cdot \mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \\
&\cdot \mathsf{ap}_{\mathsf{inl}'\circ f'}\big(\mathsf{H}(z)\big) \cdot \mathsf{Glue}'\big(\gamma'(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{ap}_{g'}\big(\mathsf{H}(z)\big)\Big)^{-1} \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)
\end{aligned}$$

We can cancel out inverses of $\mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}'_l(z)\big)$ and this leaves us with

$$\begin{aligned}
&\mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big)^{-1} \cdot \\
&\cdot \mathsf{ap}_{\mathsf{inl}'\circ f'}\big(\mathsf{H}(z)\big) \cdot \mathsf{Glue}'\big(\gamma'(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{ap}_{g'}\big(\mathsf{H}(z)\big)\Big)^{-1} \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big).
\end{aligned} \tag{6}$$

Lastly we note that

$$\mathsf{Glue}' : \prod_{c':C'} \mathsf{inl}'\big(f'(c')\big) = \mathsf{inr}'\big(g'(c')\big)$$

and since we have for every $z : C$ a path

$$\mathsf{H}(z) : \gamma(z) =_{C'} \gamma'(z),$$

we can define a path

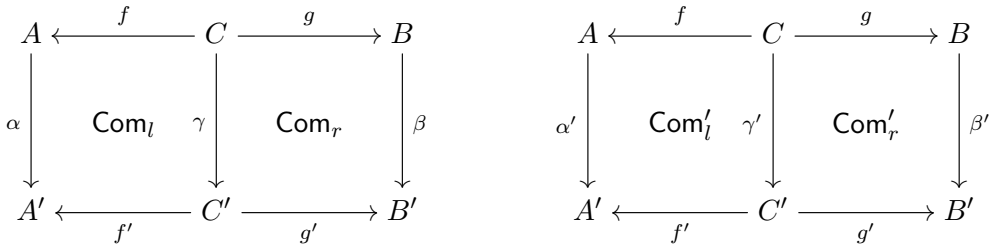
$$\mathsf{apd}_{\mathsf{Glue}'}\big(\mathsf{H}(z)\big) : \mathsf{ap}_{\mathsf{inl}' \circ f'}\big(\mathsf{H}(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}' \circ g'}\big(\mathsf{H}(z)\big) = \mathsf{Glue}'\big(\gamma'(z)\big),$$

where we used theorem 3.26. Plugging this in (6) leaves us with

$$\begin{aligned}&\mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}' \circ f'}\big(\mathsf{H}(z)\big) \cdot \mathsf{ap}_{\mathsf{inl}' \circ f'}\big(\mathsf{H}(z)\big)^{-1} \cdot \\ &\cdot \mathsf{Glue}'\big(\gamma(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}' \circ g'}\big(\mathsf{H}(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\Big(\mathsf{ap}_{g'}\big(\mathsf{H}(z)\big)\Big)^{-1} \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big).\end{aligned}$$

Canceling out all inverses shows that the left hand side of (5) equals $\mathsf{refl}_{\mathsf{inr}'(\beta(y))}$. □

**Corollary 4.26.** *Assume we are given* $\mathsf{H} : \alpha \sim \alpha'$, $\mathsf{I} : \beta \sim \beta'$, $\mathsf{J} : \gamma \sim \gamma'$ *together with commutative squares:*




*If we have additionally for every* $z : C$ *proofs that*

- $\mathsf{Com}'_l(z) = \mathsf{ap}_{f'}\big(\mathsf{J}(z)\big)^{-1} \cdot \mathsf{Com}_l(z) \cdot \mathsf{H}\big(f(z)\big)$ *and*
- $\mathsf{Com}'_r(z) = \mathsf{ap}_{g'}\big(\mathsf{J}(z)\big)^{-1} \cdot \mathsf{Com}_r(z) \cdot \mathsf{I}\big(g(z)\big)$ ,

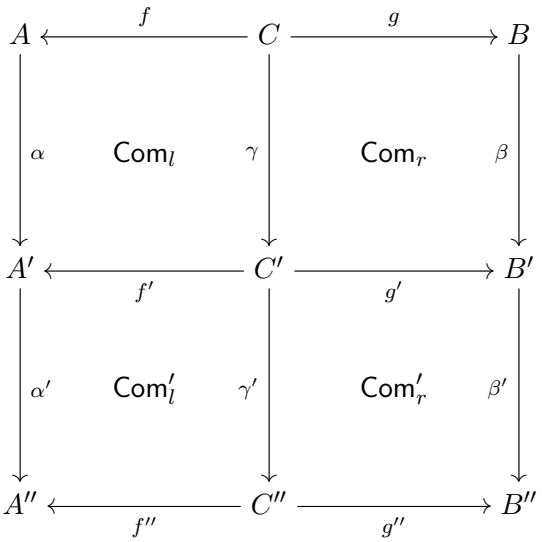
*then it holds for the induced functions*

$$\alpha \sqcup^{\gamma} \beta \sim \alpha' \sqcup^{\gamma'} \beta'$$

*Proof.* Immediate from lemma 4.24 and 4.25. □

Next we want to define a composition operation on pushout functions.

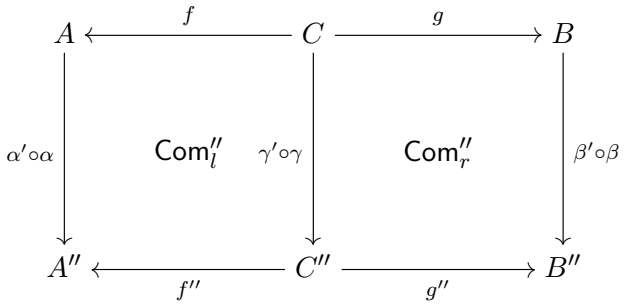
**Lemma 4.27.** *Assume, we are given four commutative squares:*




*We define proofs of commutativity by:*

- $\mathsf{Com}''_l(z) :\equiv \mathsf{Com}'_l\big(\gamma(z)\big) \centerdot \mathsf{ap}_{\alpha'}\big(\mathsf{Com}_l(z)\big)$
- $\mathsf{Com}''_r(z) :\equiv \mathsf{Com}'_r\big(\gamma(z)\big) \centerdot \mathsf{ap}_{\beta'}\big(\mathsf{Com}_r(z)\big)$

*We let* $(\alpha' \circ \alpha \sqcup^{\gamma' \circ \gamma} \beta' \circ \beta)$ *be the pushout function from the commutative squares:*




*Then we have:*

$$(\alpha' \sqcup^{\gamma'} \beta') \circ (\alpha \sqcup^{\gamma} \beta) \sim (\alpha' \circ \alpha \sqcup^{\gamma' \circ \gamma} \beta' \circ \beta)$$

*Proof.* We want to apply the induction principle of $A \sqcup^C B$ to proof the claim. So let us consider the type family

$$P : A \sqcup^C B \to \mathcal{U},$$

defined by the equation

$$P(w) :\equiv (\alpha' \sqcup^{\gamma'} \beta') \circ (\alpha \sqcup^{\gamma} \beta)(w) = (\alpha' \circ \alpha \sqcup^{\gamma' \circ \gamma} \beta' \circ \beta)(w).$$

If $w \equiv \mathsf{inl}(x)$ for some $x : A$, we compute

$$(\alpha' \sqcup^{\gamma'} \beta') \circ (\alpha \sqcup^{\gamma} \beta)\big(\mathsf{inl}(x)\big) \equiv (\alpha' \sqcup^{\gamma'} \beta')\Big(\mathsf{inl}'\big(\alpha(x)\big)\Big) \equiv \mathsf{inl}''\big(\alpha' \circ \alpha(x)\big)$$

and also

$$(\alpha' \circ \alpha \sqcup^{\gamma' \circ \gamma} \beta' \circ \beta)\big(\mathsf{inl}(x)\big) \equiv \mathsf{inl}''\big(\alpha' \circ \alpha(x)\big).$$

Similarly, if $w \equiv \mathsf{inr}(y)$ for some $y : B$, we compute

$$(\alpha' \sqcup^{\gamma'} \beta') \circ (\alpha \sqcup^{\gamma} \beta)\big(\mathsf{inr}(y)\big) \equiv (\alpha' \sqcup^{\gamma'} \beta')\Big(\mathsf{inr}'\big(\beta(y)\big)\Big) \equiv \mathsf{inr}''\big(\beta' \circ \beta(y)\big)$$

and also

$$(\alpha' \circ \alpha \sqcup^{\gamma' \circ \gamma} \beta' \circ \beta)\big(\mathsf{inr}(y)\big) \equiv \mathsf{inr}''\big(\beta' \circ \beta(y)\big).$$

Therefore we have that

$$\mathsf{refl}_{\mathsf{inl}''(\alpha' \circ \alpha(x))} : P\big(\mathsf{inl}(x)\big),$$

and in the same way we conclude

$$\mathsf{refl}_{\mathsf{inr}''\left(\beta' \circ \beta(y)\right)} : P\big(\mathsf{inr}(y)\big).$$

To finish the inductive step, we need to show for every $z : C$ that:

$$\mathsf{transport}^{P}(\mathsf{Glue}(z), \mathsf{refl}_{\mathsf{inl}''(\alpha' \circ \alpha(x))}) = \mathsf{refl}_{\mathsf{inr}''\left(\beta' \circ \beta(y)\right)} \tag{7}$$

By theorem 3.26 this means to show:

(8)

$$\mathsf{ap}_{(\alpha' \sqcup \gamma' \beta') \circ (\alpha \sqcup \gamma \beta)}\big(\mathsf{Glue}(z)\big)^{-1} \cdot \mathsf{ap}_{(\alpha' \circ \alpha \sqcup \gamma' \circ \gamma \beta' \circ \beta)}\big(\mathsf{Glue}(z)\big) = \mathsf{refl}_{\mathsf{inr}''(\beta' \circ \beta(y))}$$

To proceed, we want to compute

(i) $\mathsf{ap}_{(\alpha' \sqcup \gamma' \beta') \circ (\alpha \sqcup \gamma \beta)}\big(\mathsf{Glue}(z)\big)$ and

(ii) $\mathsf{ap}_{(\alpha' \circ \alpha \sqcup \gamma' \circ \gamma \beta' \circ \beta)}\big(\mathsf{Glue}(z)\big)$.

We start by checking the definition of pushout functions from lemma 4.21, which gives that $\mathsf{ap}_{(\alpha \sqcup \gamma \beta)}\big(\mathsf{Glue}(z)\big)$ equals

$$\mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big).$$

Since $(\alpha' \sqcup^{\gamma'} \beta') \circ \mathsf{inl}' \equiv \mathsf{inl}'' \circ \alpha'$ and similarly $(\alpha' \sqcup^{\gamma'} \beta') \circ \mathsf{inr}' \equiv \mathsf{inr}'' \circ \beta'$, we see that

$$\mathsf{ap}_{(\alpha' \sqcup^{\gamma'} \beta')}\Big(\mathsf{ap}_{\mathsf{inl}'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \mathsf{Glue}'\big(\gamma(z)\big) \cdot \mathsf{ap}_{\mathsf{inr}'}\big(\mathsf{Com}_r(z)\big)\Big) =$$
$$= \mathsf{ap}_{\mathsf{inl}'' \circ \alpha'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \mathsf{ap}_{(\alpha' \sqcup^{\gamma'} \beta')}\Big(\mathsf{Glue}'\big(\gamma(z)\big)\Big) \cdot \mathsf{ap}_{\mathsf{inr}'' \circ \beta'}\big(\mathsf{Com}_r(z)\big).$$

We use again the definition of pushout functions, by which we have that

$$\mathsf{ap}_{(\alpha' \sqcup^{\gamma'} \beta')}\Big(\mathsf{Glue}'\big(\gamma(z)\big)\Big)$$

equals:

$$\mathsf{ap}_{\mathsf{inl}''}\Big(\mathsf{inr}'\big(\beta(y)\big)\Big)^{-1} \cdot \mathsf{Glue}''\Big(\gamma'\big(\gamma(z)\big)\Big) \cdot \mathsf{ap}_{\mathsf{inr}''}\Big(\mathsf{Com}'_r\big(\gamma(z)\big)\Big).$$

Putting everything together, we see that (i) equals

$$\mathsf{ap}_{\mathsf{inl}'' \circ \alpha'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}''}\Big(\mathsf{inr}'\big(\beta(y)\big)\Big)^{-1} \cdot \mathsf{Glue}''\Big(\gamma'\big(\gamma(z)\big)\Big) \cdot$$
$$\cdot \mathsf{ap}_{\mathsf{inr}''}\Big(\mathsf{Com}'_r\big(\gamma(z)\big)\Big) \cdot \mathsf{ap}_{\mathsf{inr}'' \circ \beta'}\big(\mathsf{Com}_r(z)\big)$$

Next, we check again the definition of pushout functions from lemma 4.21, by which (ii) equals

$$\mathsf{ap}_{\mathsf{inl}''}\big(\mathsf{Com}''_l(z)\big)^{-1} \cdot \mathsf{Glue}''\Big(\gamma'\big(\gamma(z)\big)\Big) \cdot \mathsf{ap}_{\mathsf{inr}''}\big(\mathsf{Com}''_r(z)\big) \tag{9}$$

and then we use the definitions of $\mathsf{Com}''_l$ and $\mathsf{Com}''_r$, by which we have:

- $\mathsf{ap}_{\mathsf{inl}''}\big(\mathsf{Com}''_l(z)\big) \equiv \mathsf{ap}_{\mathsf{inl}''}\Big(\mathsf{inr}'\big(\beta(y)\big)\Big) \cdot \mathsf{ap}_{\mathsf{inl}''}(\mathsf{ap}_{\alpha'}\big(\mathsf{Com}_l(z)\big)$
- $\mathsf{ap}_{\mathsf{inr}''}\big(\mathsf{Com}''_r(z)\big) \equiv \mathsf{ap}_{\mathsf{inr}''}\Big(\mathsf{Com}'_r\big(\gamma(z)\big)\Big) \cdot \mathsf{ap}_{\mathsf{inr}''}(\mathsf{ap}_{\beta'}\big(\mathsf{Com}_r(z)\big)$

Plugging these definitions in (9) shows that (ii) equals:

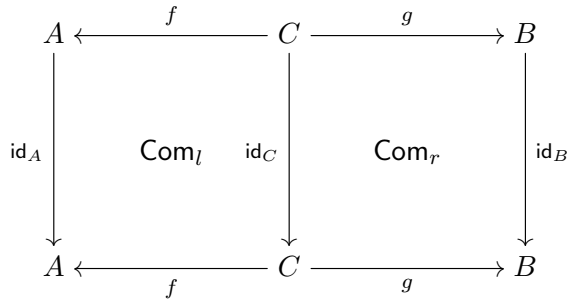
$$\mathsf{ap}_{\mathsf{inl}'' \circ \alpha'}\big(\mathsf{Com}_l(z)\big)^{-1} \cdot \mathsf{ap}_{\mathsf{inl}''}\Big(\mathsf{inr}'\big(\beta(y)\big)\Big)^{-1} \cdot \mathsf{Glue}''\Big(\gamma'\big(\gamma(z)\big)\Big) \cdot$$
$$\cdot \mathsf{ap}_{\mathsf{inr}''}\Big(\mathsf{Com}'_r\big(\gamma(z)\big)\Big) \cdot \mathsf{ap}_{\mathsf{inr}'' \circ \beta'}\big(\mathsf{Com}_r(z)\big)$$

We now see that (i) equals (ii) and this means that the left hand side of (8) equals a concatenation of inverse paths, which, by lemma 3.1, is equal to $\mathsf{refl}_{\mathsf{inr}''(\beta' \circ \beta(y))}$. □

We noted above that it is generally not enough to assume that all vertical maps of a pushout function are identity to conclude that this pushout function

is the identity. The following lemma tells us however that such a pushout function is always an equivalence.

**Lemma 4.28.** *Assume we are given types $A, B, C : \mathcal{U}$ and we let $\alpha$ be the pushout function of the commutative squares:*




*Then there exists a function $\beta$ that is a quasi-inverse of $\alpha$.*

*Proof.* Firstly we note that we can define for every $z : C$ proofs of commutativity

$$\mathsf{Com}'_l(z) :\equiv \mathsf{Com}_l(z)^{-1}$$

and

$$\mathsf{Com}'_r(z) :\equiv \mathsf{Com}_r(z)^{-1}$$

and these proofs of commutativity also make the given squares commute and therefore give a pushout function $\beta$. By lemma 4.27 we have that $\alpha \circ \beta$ is equal to the induced function of a square, where the vertical maps are identities and the proofs of commutativity are given by reflexivity. We use the induction principle of $A \sqcup^C B$ to proof, that $\alpha \circ \beta$ is pointwise equal to the identity function. To this end we consider the type family

$$P : A \sqcup^C B \to \mathcal{U},$$

defined by the equation

$$P(w) :\equiv \alpha \circ \beta(w) = \mathsf{id}_{A \sqcup^C B}(w).$$

It is obvious, that over the points, we can proof the claim by reflexivity. Then we need to show for every $z : C$ that:

$$\mathsf{transport}^P(\mathsf{Glue}(z), \mathsf{refl}_{\mathsf{inl}(f(z))}) = \mathsf{refl}_{\mathsf{inr}(g(z))}$$

By theorem 3.26 this means to show:

$$\mathsf{ap}_{\alpha\circ\beta}\big(\mathsf{Glue}(z)\big)^{-1} \cdot \mathsf{Glue}(z) = \mathsf{refl}_{\mathsf{inr}(g(z))} \tag{10}$$

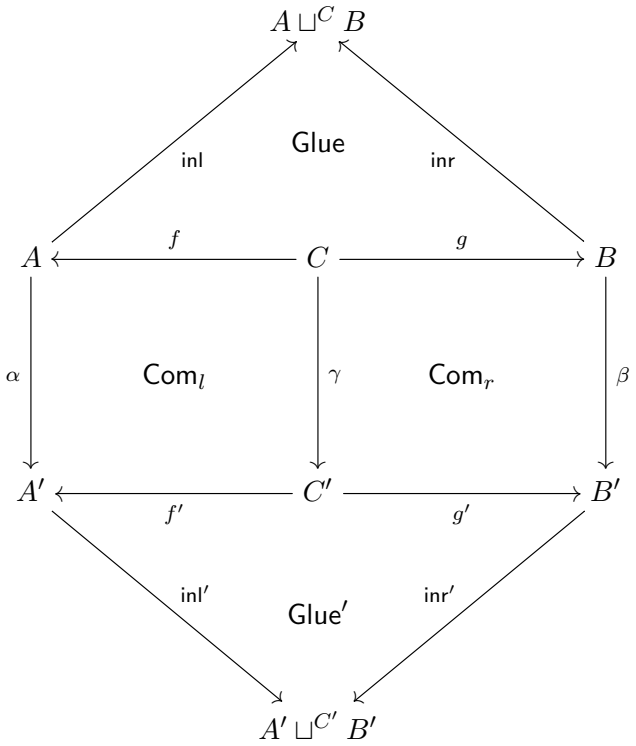
But, since now the proofs of commutativity are pointwise equal to reflexivity, by definition of pushout functions, we have

$$\mathsf{ap}_{\alpha\circ\beta}\big(\mathsf{Glue}(z)\big)^{-1} = \mathsf{Glue}(z)^{-1},$$

hence, by canceling inverse paths, we see, that the left hand side of (10) equals a concatenation of inverse paths, which, by lemma 3.1, is equal to $\mathsf{refl}_{\mathsf{inr}(g(z))}$. The same reasoning applies to $\beta \circ \alpha$. □

4.3. **Homotopy invariance of pushout types.** We are now able to show that pushout types are indeed invariant under homotopy.

**Theorem 4.29.** *Assume, we are given the following squares:*



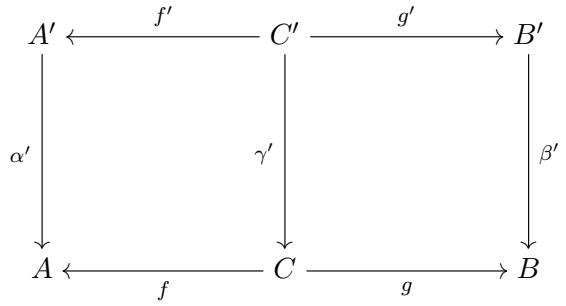

*Then, if $\alpha$, $\beta$ and $\gamma$ are equivalences, there is also an equivalence*

$$A \sqcup^C B \simeq A' \sqcup^{C'} B'$$

*Proof.* Since $\alpha$ and $\beta$ are equivalences, by lemma 3.15 there are maps $\alpha' : A' \to A$, $\beta' : B' \to B$ and $\gamma' : C' \to C$ together with proofs:

- $\mathsf{H} : \alpha' \circ \alpha \sim \mathsf{id}_A$
- $\mathsf{H}' : \alpha \circ \alpha' \sim \mathsf{id}_{A'}$
- $\mathsf{I} : \beta' \circ \beta \sim \mathsf{id}_B$
- $\mathsf{I}' : \beta \circ \beta' \sim \mathsf{id}_{B'}$
- $\mathsf{J} : \gamma' \circ \gamma \sim \mathsf{id}_C$
- $\mathsf{J}' : \gamma \circ \gamma' \sim \mathsf{id}_{C'}$

We claim that from $\mathsf{H}$, $\mathsf{I}$ and $\mathsf{J}'$ we can define proofs of commutativity that make the following squares commute:




To this end, we firstly note that for every $z : C$ we have that

$$\mathsf{ap}_{\alpha'}\big(\mathsf{Com}_l(z)\big) : \alpha'\Big(f'\big(\gamma(z)\big)\Big) = \alpha'\Big(\alpha\big(f(z)\big)\Big)$$

and

$$\mathsf{H}\big(f(z)\big) : \alpha' \circ \alpha\big(f(z)\big) = f(z).$$

Therefore we can define for every $z : C$ a path:

$$\mathsf{ap}_{\alpha'}\big(\mathsf{Com}_l(z)\big) \centerdot \mathsf{H}\big(f(z)\big) : \alpha'\Big(f'\big(\gamma(z)\big)\Big) = f(z)$$

Next we note that for every $z' : C'$ we have that

$$\mathsf{J}'(z') : \gamma \circ \gamma'(z') = c'$$

and therefore

$$\mathsf{ap}_{\alpha' \circ f'}\big(\mathsf{J}(z')\big) : \alpha'\Big(f'\big(\gamma(\gamma'(z'))\big)\Big) = \alpha'\big(f'(z')\big).$$

In particular, since for every $z' : C'$ we have that $\gamma'(z') : C$, we can define for every $z' : C'$ a path of type

$$f\big(\gamma'(z')\big) = \alpha'\big(f'(z')\big)$$

by

$$\mathsf{H}\Big(f\big(\gamma'(z')\big)\Big)^{-1} \cdot \mathsf{ap}_{\alpha'}\Big(\mathsf{Com}_l\big(\gamma'(z')\big)\Big)^{-1} \cdot \mathsf{ap}_{\alpha'\circ f'}\big(\mathsf{J}'(z')\big).$$

To summarize, we can make the left square of the above diagram commute by:

$$\mathsf{Com}'_l(z') :\equiv \mathsf{H}\Big(f\big(\gamma'(z')\big)\Big)^{-1} \cdot \mathsf{ap}_{\alpha'}\Big(\mathsf{Com}_l\big(\gamma'(z')\big)\Big)^{-1} \cdot \mathsf{ap}_{\alpha'\circ f'}\big(\mathsf{J}'(z')\big)$$

Similarly, we have for every $z : C$ that

$$\mathsf{ap}_{\beta'}\big(\mathsf{Com}_r(z)\big) : \beta'\Big(g'\big(\gamma(z)\big)\Big) = \beta'\Big(\beta\big(g(z)\big)\Big)$$

and

$$\mathsf{I}\big(g(z)\big) : \beta'\Big(\beta\big(g(z)\big)\Big) = g(z).$$

Therefore we can define for every $z : C$ a path:

$$\mathsf{I}\big(g(z)\big)^{-1} \cdot \mathsf{ap}_{\beta'}\big(\mathsf{Com}_r(z)\big)^{-1} : g(z) = \beta'\Big(g'\big(\gamma(z)\big)\Big)$$

Then we use again that for every $z' : C'$ we have that

$$\mathsf{J}'(z') : \gamma \circ \gamma'(z') = c'$$

and therefore

$$\mathsf{ap}_{\beta'}\Big(\mathsf{ap}_{g'}\big(\mathsf{J}'(z')\big)\Big) : \beta'\Big(g'\big(\gamma(\gamma'(z'))\big)\Big) = \beta'\big(g'(z')\big).$$

Then, since for every $z' : C'$ we have $\gamma'(z') : C$, we can define for every $z' : C'$ a path of type

$$g\big(\gamma'(z')\big) = \beta'\big(g'(z')\big)$$

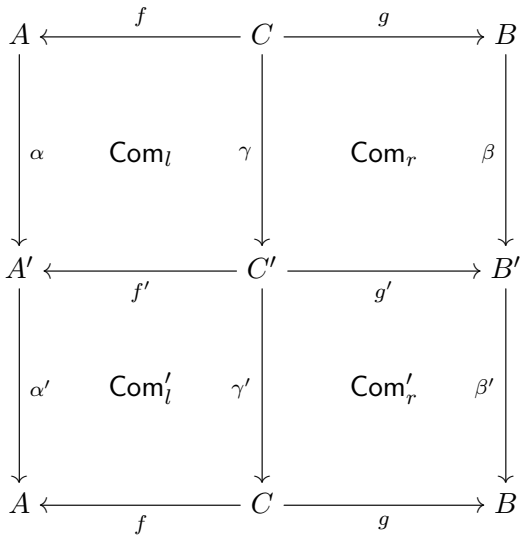
by

$$\mathsf{I}\Big(g\big(\gamma'(z')\big)\Big)^{-1} \cdot \mathsf{ap}_{\beta'}\Big(\mathsf{Com}_r\big(\gamma'(z')\big)\Big)^{-1} \cdot \mathsf{ap}_{\beta'}\Big(\mathsf{ap}_{g'}\big(\mathsf{J}'(z')\big)\Big).$$

To summarize, we can make the right square of the above diagram commute by:

$$\mathsf{Com}'_l(z') :\equiv \mathsf{I}\Big(g\big(\gamma'(z')\big)\Big)^{-1} \cdot \mathsf{ap}_{\beta'}\Big(\mathsf{Com}_r\big(\gamma'(z')\big)\Big)^{-1} \cdot \mathsf{ap}_{\beta'}\Big(\mathsf{ap}_{g'}\big(\mathsf{J}'(z')\big)\Big)$$

Putting everything together, we get the following commutative squares:




By lemma 4.27, there is a proof

$$(\alpha' \sqcup^{\gamma'} \beta') \circ (\alpha \sqcup^{\gamma} \beta) = (\alpha' \circ \alpha) \sqcup^{\gamma' \circ \gamma} (\beta' \circ \beta),$$

where the right hand side of this equation is the pushout function of two commutative squares, where proofs of commutativity are given by:

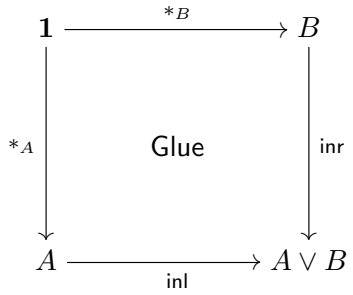
- $\mathsf{Com}''_l(z) :\equiv \mathsf{Com}'_l\big(\gamma(z)\big) \cdot \mathsf{ap}_{\alpha'}\big(\mathsf{Com}_l(z)\big)$
- $\mathsf{Com}''_r(z) :\equiv \mathsf{Com}'_r\big(\gamma(z)\big) \cdot \mathsf{ap}_{\beta'}\big(\mathsf{Com}_r(z)\big)$

By Corollary 4.26, from $\mathsf{H}$, $\mathsf{I}$ and $\mathsf{J}$ we get a path from $(\alpha' \circ \alpha) \sqcup^{\gamma' \circ \gamma} (\beta' \circ \beta)$ to the pushout function of two commutative squares, where all vertical maps are identity. By lemma 4.28, there is a function $\delta_r$, such that $(\alpha' \sqcup^{\gamma'} \beta') \circ (\alpha \sqcup^{\gamma} \beta) \circ \delta_r$ is the identity function.

By the same reasoning, where we now have to use $\mathsf{H}'$, $\mathsf{I}'$ and $\mathsf{J}'$, we see that also $(\alpha \sqcup^{\gamma} \beta) \circ (\alpha' \sqcup^{\gamma'} \beta')$ is equal to the pushout function of commutative squares, where all vertical maps are identity, hence by 4.28 there is a function $\delta_l$, such that $\delta_l \circ (\alpha \sqcup^{\gamma} \beta) \circ (\alpha' \sqcup^{\gamma'} \beta')$ is the identity function. But this means $(\alpha \sqcup^{\gamma} \beta) \circ \delta_r$ is right inverse to $(\alpha' \sqcup^{\gamma'} \beta')$ and $\delta_l \circ (\alpha \sqcup^{\gamma} \beta)$ is left inverse to $(\alpha' \sqcup^{\gamma'} \beta')$. This shows, that $(\alpha' \sqcup^{\gamma'} \beta')$ is an equivalence. □

## 5. The wedge sum

In example 4.15 we presented the wedge sum $A \vee B$ of two pointed types $A$ and $B$. It was defined as the pushout of the square:




We made $A \vee B$ a pointed type, by setting $*_{A\vee B} :\equiv \mathsf{inl}(*_A)$. From now on we will shift our attention to the category of pointed maps and base-point preserving functions (see 4.11 and 4.13), i.e. every type $A : \mathcal{U}$ will come along with a specified base-point $*_A : A$ and every map $f : A \to B$ will come along with a proof $p_f : *_B = f(*_A)$ that $f$ is base-point preserving.

*Remark* 5.1. In the definition of 4.15 we noted that for any two pointed types $A, B : \mathcal{U}$ also $A \vee B$ is pointed, where we chose $*_{A\vee B} :\equiv \mathsf{inl}(*_A)$ as its base-point. Similarly we have proofs that $\mathsf{inl}_{A\vee B}$ and $\mathsf{inr}_{A\vee B}$ are base-point preserving by $\mathsf{refl}_{\mathsf{inl}(*_A)} \equiv \mathsf{refl}_{*_{A\vee B}}$ and $\mathsf{glue}_{A\vee B}$ respectively.

*Remark* 5.2. Assume we are given three pointed types $A, B, C : \mathcal{U}$ and base-point preserving functions $(f, p_f) : \mathsf{Map}^*(A, B)$ and $(g, p_g) : \mathsf{Map}^*(B, C)$, then we can define a composition operation

$$\_ \circ \_ : \mathsf{Map}^*(B, C) \times \mathsf{Map}^*(A, B) \to \mathsf{Map}^*(A, C)$$

by

$$(g, p_g) \circ (f, p_f) :\equiv \big(g \circ f, p_g \cdot \mathsf{ap}_g(p_f)\big).$$

We check that

$$\mathsf{ap}_g(p_f) : g(*_B) = g\big(f(*_A)\big)$$

and

$$p_g : *_C = g(*_B)$$

and therefore

$$p_g \cdot \mathsf{ap}_g(p_f) : *_C = g\big(f(*_A)\big)$$

has appropriate type. We note that this is exactly how we defined the proofs of commutativity in lemma 4.27.

**Lemma 5.3.** *Assume we are given two pointed types $A, B : \mathcal{U}$. Then there is an equivalence:*

$$A \vee B \simeq B \vee A$$

*Proof.* Firstly, by applying the recursion principle of $A \vee B$, we define a function $\Phi : A \vee B \to B \vee A$:

- $\Phi\big(\mathsf{inl}_{A\vee B}(x)\big) :\equiv \mathsf{inr}_{B\vee A}(x)$
- $\Phi\big(\mathsf{inr}_{A\vee B}(y)\big) :\equiv \mathsf{inl}_{B\vee A}(y)$
- $\mathsf{ap}_{\Phi}(\mathsf{glue}_{A\vee B}) \coloneqq \mathsf{glue}_{B\vee A}$

Similarly we define a function $\Psi : B \vee A \to A \vee B$:

- $\Phi\big(\mathsf{inl}_{B\vee A}(y)\big) :\equiv \mathsf{inr}_{A\vee B}(y)$
- $\Phi\big(\mathsf{inr}_{B\vee A}(x)\big) :\equiv \mathsf{inl}_{A\vee B}(x)$
- $\mathsf{ap}_{\Phi}(\mathsf{glue}_{B\vee A}) \coloneqq \mathsf{glue}_{A\vee B}$

We want to show that $\Phi$ and $\Psi$ are mutually inverse. It is obvious from the computation rules that

$$\mathsf{refl}_{\mathsf{inl}_{A\vee B}(x)} : \Psi \circ \Phi\big(\mathsf{inl}_{A\vee B}(x)\big) = \mathsf{inl}_{A\vee B}(x)$$

and that

$$\mathsf{refl}_{\mathsf{inr}_{A\vee B}(y)} : \Psi \circ \Phi\big(\mathsf{inr}_{A\vee B}(y)\big) = \mathsf{inr}_{A\vee B}(y).$$

Then it is left to show that

$$\mathsf{transport}^{w\mapsto \Psi\circ\Phi(w)=w}(\mathsf{glue}_{A\vee B}, \mathsf{refl}_{\mathsf{inl}_{A\vee B}(x)}) = \mathsf{refl}_{\mathsf{inr}_{A\vee B}(y)}.$$

By theorem 3.26 this means to show:

$$\mathsf{ap}_{\Psi\circ\Phi}(\mathsf{glue}_{A\vee B})^{-1} \cdot \mathsf{glue}_{A\vee B} = \mathsf{refl}_{\mathsf{inr}_{A\vee B}(y)}.$$

From the computation rules it is immediate that

$$\mathsf{ap}_{\Psi\circ\Phi}(\mathsf{glue}_{A\vee B})^{-1} = \mathsf{glue}^{-1}_{A\vee B}$$

and hence the claim follow from lemma 3.1. The other direction is completely analogue. □

In the main result of this section we establish that the wedge sum is the coproduct in the category of pointed types and base-point preserving maps

(see 5.8). The coproduct of two objects $X$ and $Y$ in some category is the object $X \sqcup Y$ together with two morphisms $\mathsf{inj}_l : X \to X \sqcup Y$ and $\mathsf{inj}_r : Y \to X \sqcup Y$, such that for any object $Z$ in the category and any two morphisms $f : X \to Z$ and $g : Y \to Z$, there is a unique morphism $f \sqcup g : X \sqcup Y \to Z$ for which $(f \sqcup g) \circ \mathsf{inj}_l = f$ and $(f \sqcup g) \circ \mathsf{inj}_r = g$. We continue with our discussion of the wedge sum by considering a special instance of pushout functions.

**Lemma 5.4.** *Assume we are given two pointed types $A, B : \mathcal{U}$ together with base-point preserving maps $(\alpha, p_\alpha) : \mathsf{Map}^*(A, A')$ and $(\beta, p_\beta) : \mathsf{Map}^*(B, B')$. We let:*

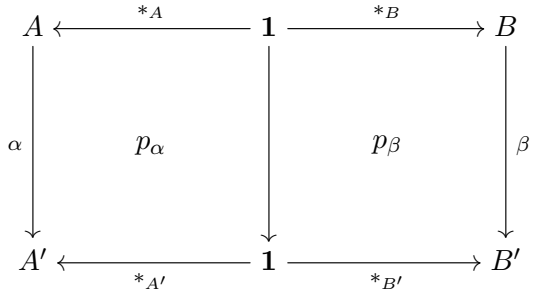
- $\mathsf{inl} : A \to (A \vee B)$
- $\mathsf{inr} : B \to (A \vee B)$
- $\mathsf{glue} : \mathsf{inl}(*_A) =_{(A \vee B)} \mathsf{inr}(*_B)$
- $\mathsf{inl}' : A' \to (A' \vee B')$
- $\mathsf{inr}' : B' \to (A' \vee B')$
- $\mathsf{glue}' : \mathsf{inl}'(*_{A'}) =_{(A' \vee B')} \mathsf{inr}(*_{B'})$

*There is a canonical function $\alpha \vee \beta : A \vee B \to A' \vee B'$, called the **wedge** of $\alpha$ and $\beta$, such that:*

- $\alpha \vee \beta \circ \mathsf{inl} \equiv \mathsf{inl}' \circ \alpha$
- $\alpha \vee \beta \circ \mathsf{inr} \equiv \mathsf{inr}' \circ \beta$
- $\mathsf{ap}_{(\alpha \vee \beta)}(\mathsf{glue}) = \mathsf{ap}_{\mathsf{inl}'}(p_\alpha)^{-1} \cdot \mathsf{glue}' \cdot \mathsf{ap}_{\mathsf{inr}'}(p_\beta)$

*What is more, $\mathsf{ap}_{\mathsf{inl}'}(p_\alpha)$ proves that $\alpha \vee \beta$ is base-point preserving.*

*Proof.* We define $\alpha \vee \beta$ to be the pushout function of the two commutative squares:




This function has the appropriate computation rules by lemma 4.21. Lastly we check that

$$p_\alpha : *_{A'} = \alpha(*_A)$$

and hence

$$\mathsf{ap}_{\mathsf{inl}'}(p_\alpha) : \mathsf{inl}'(*_{A'}) = \mathsf{inl}'\big(\alpha(*_A)\big).$$

Since by definition

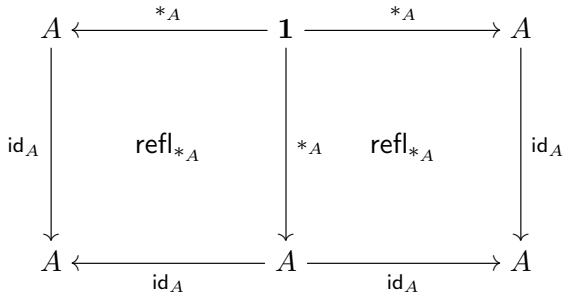
$$*_{A'\vee B'} \equiv \mathsf{inl}'(*_{A'})$$

and

$$(\alpha \vee \beta)(*_{A\vee B}) \equiv (\alpha \vee \beta)\big(\mathsf{inl}(*_A)\big) \equiv \mathsf{inl}'\big(\alpha(*_A)\big),$$

it holds indeed that $\mathsf{ap}_{\mathsf{inl}'}(p_\alpha)$ proves that $(\alpha\vee\beta)$ is base-point preserving. □

Next, we consider the pushout function $\delta : A \vee A \to A \sqcup^A A$ of the commutative squares:




We make $A \sqcup^A A$ a pointed type by setting $\mathsf{inl}_{*_A}$ as its base-point. By remark 4.2 there is an equivalence $\Phi : A \sqcup^A A \to A$, which is base-point preserving by $\mathsf{refl}_{*_A}$ and this gives us a function

$$\mathsf{Col} :\equiv \Phi \circ \delta \tag{11}$$

of type

$$A \vee A \to A,$$

such that:

- $\mathsf{Col} \circ \mathsf{inl} \equiv \mathsf{id}_A$
- $\mathsf{Col} \circ \mathsf{inr} \equiv \mathsf{id}_A$
- $\mathsf{ap}_{\mathsf{Col}}(\mathsf{glue}) = \mathsf{refl}_{*_A}$

What is more, $\mathsf{Col}$ is base-point preserving by $\mathsf{refl}_{\mathsf{inl}_{*_A}}$. With this at hand, to define a function $h : A \vee B \to C$, it suffices to give two base-point preserving functions $f : A \to C$ and $g : B \to C$, since we may wedge $f$ and $g$ and

compose with $\mathsf{Col}$. We introduce a special notation and set:

$$f \vee^c g :\equiv \mathsf{Col} \circ (f \vee g)$$

**Lemma 5.5.** *Assume we are given two pointed types $A, B : \mathcal{U}$, then we can define the left and right projections $\pi_l : A \vee B \to A$ and $\pi_r : A \vee B \to B$, such that:*

- $\pi_l\big(\mathsf{inl}(x)\big) \equiv x$
- $\pi_l\big(\mathsf{inr}(y)\big) \equiv *_A$
- $\mathsf{ap}_{\pi_l}(\mathsf{glue}) = \mathsf{refl}_{*_A}$
- $\pi_r\big(\mathsf{inl}(x)\big) \equiv *_B$
- $\pi_r\big(\mathsf{inr}(y)\big) \equiv y$
- $\mathsf{ap}_{\pi_l}(\mathsf{glue}) = \mathsf{refl}_{*_B}$

*What is more $\pi_l$ is base-point preserving by $\mathsf{refl}_{*_A}$ and $\pi_r$ is base-point preserving by $\mathsf{refl}_{*_B}$.*

*Proof.* We can define $\pi_l$ as the collapsed pushout function of the commutative squares

$$\begin{array}{ccccc}
A & \xleftarrow{*_A} & \mathbf{1} & \xrightarrow{*_B} & B \\
{\scriptstyle \mathsf{id}_A}\downarrow & \mathsf{refl}_{*_A} & \downarrow{\scriptstyle *_A} & \mathsf{refl}_{*_A} & \downarrow{\scriptstyle *_A} \\
A & \xleftarrow[\mathsf{id}_A]{} & A & \xrightarrow[\mathsf{id}_A]{} & A
\end{array}$$

and $\pi_r$ as the collapsed pushout function of the commutative squares

$$\begin{array}{ccccc}
A & \xleftarrow{*_A} & \mathbf{1} & \xrightarrow{*_B} & B \\
{\scriptstyle *_B}\downarrow & \mathsf{refl}_{*_B} & \downarrow{\scriptstyle *_B} & \mathsf{refl}_{*_B} & \downarrow{\scriptstyle \mathsf{id}_B} \\
B & \xleftarrow[\mathsf{id}_B]{} & B & \xrightarrow[\mathsf{id}_B]{} & B
\end{array}$$

These functions have the appropriate computation rules by lemma 4.21. Lastly we compute

$$\pi_l(*_{A\vee B}) \equiv \pi_l\big(\mathsf{inr}(*_B)\big) \equiv *_A$$

and

$$\pi_r(*_{A\vee B}) \equiv \pi_r\big(\mathsf{inr}(*_B)\big) \equiv *_B,$$

therefore $\mathsf{refl}_{*_A}$ and $\mathsf{refl}_{*_B}$ proof that $\pi_l$ and $\pi_r$ are base-point preserving respectively. □

**Lemma 5.6.** *Assume we are given two pointed types $A, B : \mathcal{U}$. We let*

$$\mathsf{inl}_{A\vee B} \vee^c \mathsf{inr}_{A\vee B}$$

*be the collapsed pushout function of the commutative squares with proofs of commutativity given by*

$$p_{\mathsf{inl}_{A\vee B}} :\equiv \mathsf{refl}_{\mathsf{inl}_{A\vee B}(*_A)}$$

*and*

$$p_{\mathsf{inr}_{A\vee B}} :\equiv \mathsf{glue}_{A\vee B}.$$

*Then it holds that:*

$$\mathsf{inl}_{A\vee B} \vee^c \mathsf{inr}_{A\vee B} \sim \mathsf{id}_{A\vee B}$$

*Proof.* We use the induction principle of $A \vee B$ to proof the claim. To this end, we let:

$$P(w) :\equiv \mathsf{inl}_{A\vee B} \vee^c \mathsf{inr}_{A\vee B}(w) = w$$

We compute

$$(\mathsf{inl}_{A\vee B} \vee^c \mathsf{inr}_{A\vee B})\big(\mathsf{inl}_{A\vee B}(x)\big) \equiv \mathsf{Col} \circ \mathsf{inl}_{(A\vee B)\vee(A\vee B)} \circ \mathsf{inl}_{A\vee B}(x) \equiv \mathsf{inl}_{A\vee B}(x)$$

and

$$(\mathsf{inl}_{A\vee B} \vee^c \mathsf{inr}_{A\vee B})\big(\mathsf{inr}_{A\vee B}(y)\big) \equiv \mathsf{Col} \circ \mathsf{inl}_{(A\vee B)\vee(A\vee B)} \circ \mathsf{inr}_{A\vee B}(y) \equiv \mathsf{inr}_{A\vee B}(y)$$

Therefore we can set

$$\mathsf{refl}_{\mathsf{inl}_{A\vee B}(*_A)} : P\big(\mathsf{inl}_{A\vee B}(*_A)\big)$$

and

$$\mathsf{refl}_{\mathsf{inr}_{A\vee B}(*_B)} : P\big(\mathsf{inr}_{A\vee B}(*_B)\big).$$

Then, we need to show:

$$\mathsf{transport}^P(\mathsf{glue}_{A\vee B}, \mathsf{refl}_{\mathsf{inl}_{A\vee B}(*_A)}) = \mathsf{refl}_{\mathsf{inr}_{A\vee B}(*_B)}$$

By theorem 3.26 this means to show:

$$\mathsf{ap}_{(\mathsf{inl}_{A\vee B}\vee^c \mathsf{inr}_{A\vee B})}(\mathsf{glue}_{A\vee B})^{-1} \cdot \mathsf{glue}_{A\vee B} = \mathsf{refl}_{\mathsf{inr}_{A\vee B}(*_B)} \tag{12}$$

By definition, $\mathsf{inl}_{A\vee B}$ and $\mathsf{inr}_{A\vee B}$ are base-point preserving by

$$p_{\mathsf{inl}_{A\vee B}} :\equiv \mathsf{refl}_{\mathsf{inl}_{A\vee B}(*_A)}$$

and

$$p_{\mathsf{inr}_{A\vee B}} :\equiv \mathsf{glue}_{A\vee B}$$

respectively. Therefore by the computation rules for pushout functions from lemma 4.21, we have that $\mathsf{ap}_{\mathsf{inl}_{A\vee B}\vee\mathsf{inr}_{A\vee B}}(\mathsf{glue}_{A\vee B})$ equals:

$$\mathsf{glue}_{(A\vee B)\vee(A\vee B)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\vee B)\vee(A\vee B)}}(\mathsf{glue}_{A\vee B}) \tag{13}$$

Lastly we note that by definition of $\mathsf{Col}$ in 11 we have that

$$\mathsf{ap}_{\mathsf{Col}}(\mathsf{glue}_{(A\vee B)\vee(A\vee B)}) = \mathsf{refl}_{*_{(A\vee B)}}$$

and since $\mathsf{Col} \circ \mathsf{inr}_{(A\vee B)\vee(A\vee B)}$ is the identity function we have that

$$\mathsf{ap}_{\mathsf{Col}\circ\mathsf{inr}_{(A\vee B)\vee(A\vee B)}}(\mathsf{glue}_{A\vee B}) = \mathsf{glue}_{A\vee B}.$$

But this means that

$$\begin{aligned}
&\mathsf{ap}_{\mathsf{Col}}\Big(\mathsf{glue}_{(A\vee B)\vee(A\vee B)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\vee B)\vee(A\vee B)}}(\mathsf{glue}_{A\vee B})\Big) = \\
&= \mathsf{ap}_{\mathsf{Col}}\Big(\mathsf{glue}_{(A\vee B)\vee(A\vee B)}\Big) \cdot \mathsf{ap}_{\mathsf{Col}}\Big(\mathsf{ap}_{\mathsf{inr}_{(A\vee B)\vee(A\vee B)}}(\mathsf{glue}_{A\vee B})\Big) = \\
&= \mathsf{refl}_{*_{(A\vee B)}} \cdot \mathsf{glue}_{A\vee B} = \mathsf{glue}_{A\vee B},
\end{aligned}$$

where in the first step we used lemma 3.4 and in the last step we used lemma 3.1. But this means, that the left hand side of (12) is equal to a concatenation of inverse paths and therefore equal to $\mathsf{refl}_{\mathsf{inr}_{A\vee B}(*_B)}$, again by lemma 3.1. □

**Lemma 5.7.** *Assume we are given two pointed types $A, B : \mathcal{U}$ and a base-point preserving map $(f, p_f) : \mathsf{Map}^*(A, B)$. Then we have:*

$$f \circ \mathsf{Col}_A \sim (f \vee^c f)$$

*Proof.* We aim to proof the claim by applying the induction principle for $A \vee A'$. So let us consider the type family

$$R : A \vee A' \to \mathcal{U}$$

defined by the equation

$$R(w) :\equiv \big(f \circ \mathsf{Col}_A(w)\big) = \big(\mathsf{Col}_B \circ (f \vee f)(w)\big).$$

By definition of $\mathsf{Col}$ in 11 we have that

$$\mathsf{Col}_A \circ \mathsf{inl}_{A \vee A} \equiv \mathsf{id}_A$$

and therefore

$$f \circ \mathsf{Col}_A \circ \mathsf{inl}_{A \vee A} \equiv f.$$

Similarly, by definition of $(f \vee f)$ in lemma 5.4, we have that

$$(f \vee f) \circ \mathsf{inl}_{A \vee A} \equiv \mathsf{inl}_{B \vee B} \circ f$$

and using again the definition of $\mathsf{Col}$ in 11 that

$$\mathsf{Col}_B \circ \mathsf{inl}_{B \vee B} \equiv \mathsf{id}_B.$$

Putting the last equations together gives us

$$\mathsf{Col}_B \circ (f \vee f) \circ \mathsf{inl}_{A \vee A} \equiv \mathsf{Col}_B \circ \mathsf{inl}_{B \vee B} \circ f \equiv f$$

and this means that for every $x : A$ we can set

$$\mathsf{refl}_{f(x)} : R\big(\mathsf{inl}_{A \vee A}(x)\big)$$

and similarly

$$\mathsf{Col}_B \circ (f \vee f) \circ \mathsf{inr}_{A \vee A} \equiv \mathsf{Col}_B \circ \mathsf{inr}_{B \vee B} \circ f \equiv f$$

and this means that for every $x : A$ we can set

$$\mathsf{refl}_{f(x)} : R\big(\mathsf{inr}_{A \vee A}(x)\big).$$

Then, what is left is to show that

$$\mathsf{transport}^R(\mathsf{glue}_{A \vee A}, \mathsf{refl}_{f(*_A)}) = \mathsf{refl}_{f(*_A)}.$$

By theorem 3.26 this means to show:

$$\mathsf{ap}_{f \circ \mathsf{Col}_A}(\mathsf{glue}_{A \vee A})^{-1} \cdot \mathsf{ap}_{(f \vee^c f)}(\mathsf{glue}_{A \vee A}) = \mathsf{refl}_{f(*_A)} \tag{14}$$

But the definition of $\mathsf{Col}$ in 11 gives

$$\mathsf{ap}_{\mathsf{Col}_A}(\mathsf{glue}_{A\vee A}) = \mathsf{refl}_{*_A}$$

and hence

$$\begin{aligned}\mathsf{ap}_{f\circ\mathsf{Col}_A}(\mathsf{glue}_{A\vee A})^{-1} \cdot \mathsf{ap}_{(f\vee^c f)}(\mathsf{glue}_{A\vee A}) &= \\ = \mathsf{ap}_f(\mathsf{refl}_{*_A})^{-1} \cdot \mathsf{ap}_{(f\vee^c f)}(\mathsf{glue}_{A\vee A}) &= \\ = \mathsf{ap}_{(f\vee^c f)}(\mathsf{glue}_{A\vee A}),&\end{aligned}$$

where we used lemmas 3.4 and 3.1. Lastly we check again the definition of $\mathsf{Col}$, by which we have

- $\mathsf{Col} \circ \mathsf{inl} \equiv \mathsf{id}_A$ ,
- $\mathsf{Col} \circ \mathsf{inr} \equiv \mathsf{id}_A$ and
- $\mathsf{ap}_{\mathsf{Col}}(\mathsf{glue}) = \mathsf{refl}_{*_A}$

and then we use computation rules for the wedge of functions, provided in 5.4, which gives us that

$$\mathsf{ap}_{f\vee f}(\mathsf{glue}_{A\vee A}) = \mathsf{ap}_{\mathsf{inl}_{B\vee B}}(p_f) \cdot \mathsf{glue}_{B\vee B} \cdot \mathsf{ap}_{\mathsf{inr}_{B\vee B}}(p_f)$$

and therefore

$$\begin{aligned}&\mathsf{ap}_{(f\vee^c f)}(\mathsf{glue}_{A\vee A}) = \\ &= \mathsf{ap}_{\mathsf{Col}}\Big(\mathsf{ap}_{\mathsf{inl}_{B\vee B}}(p_f) \cdot \mathsf{glue}_{B\vee B} \cdot \mathsf{ap}_{\mathsf{inr}_{B\vee B}}(p_f)\Big) = \\ &= \mathsf{pr}_f^{-1} \cdot \mathsf{pr}_f.\end{aligned}$$

This shows that the left hand side of (14) is equal to a concatenation of inverse paths and the proof is proven by lemma 3.1. □

We can now proof that the wedge sum is the coproduct in the category of pointed types and base-point preserving maps.

**Proposition 5.8.** *The wedge sum is the coproduct in the category of pointed types and base-point preserving maps*

*Proof.* For any two pointed maps $f : A \to C$ and $g : B \to C$, we have that $f \vee^c g : A \vee B \to C$ has appropriate type. It is left to show uniqueness. So assume, we are given pointed types $A, B, C : \mathcal{U}$ together with base-point

preserving functions functions $(f, p_f) : \mathsf{Map}^*(A, C)$, $(g, p_g) : \mathsf{Map}^*(B, C)$ and $(\alpha, p_\alpha) : \mathsf{Map}^*(A \vee B, C)$, such that:

- $(\alpha, p_\alpha) \circ (\mathsf{inl}, \mathsf{glue}) =_{\mathsf{Map}^*(A,C)} (f, p_f)$
- $(\alpha, p_\alpha) \circ (\mathsf{inr}, \mathsf{refl}_{*_{A\vee B}}) =_{\mathsf{Map}^*(B,C)} (g, p_g)$

Now firstly we recall that

$$p_{\mathsf{inl}} \equiv \mathsf{refl}_{*_{A\vee B}}$$

and

$$p_{\mathsf{inr}} \equiv \mathsf{glue}.$$

We then use 5.2, by which we have that

$$(\alpha, p_\alpha) \circ (\mathsf{inl}, \mathsf{refl}_{*_{A\vee B}}) \equiv \big(\alpha \circ \mathsf{inl}, p_\alpha\big)$$

and

$$(\alpha, p_\alpha) \circ (\mathsf{inr}, \mathsf{glue}) \equiv \big(\alpha \circ \mathsf{inr}, p_\alpha \centerdot \mathsf{ap}_\alpha(\mathsf{glue})\big).$$

By theorem 3.16 we get that

$$\big(\alpha \circ \mathsf{inl}, p_\alpha\big) =_{\mathsf{Map}^*(A,C)} (f, p_f)$$

is equivalent to

$$\sum_{p:\alpha\circ\mathsf{inl}=f} \mathsf{transport}^{h\mapsto *_C = h(*_A)}(p, p_\alpha) = p_f.$$

Hence we may assume that we are given an inhabitant $p_l : \alpha \circ \mathsf{inr} = f$, such that we have that

$$\mathsf{transport}^{h\mapsto h(*_A)=*_C}(p_l, p_\alpha) = p_f.$$

We use theorem 3.26, lemma 3.11 and remark 3.5, by which we have

$$\begin{aligned}\mathsf{transport}^{h\mapsto h(*_A)=*_C}(p_l, p_\alpha) &= \\ = \mathsf{ap}_{\mathsf{eval}(*_A)}(p_l) \centerdot p_\alpha &= \\ = \mathsf{happly}(p_l)(*_A)^{-1} \centerdot p_\alpha.&\end{aligned}$$

Therefore, we get that

$$p_\alpha \centerdot \mathsf{happly}(p_l)(*_A) = p_g. \tag{15}$$

We note that

$$\mathsf{happly}(p_l) : \prod_{x:A} \alpha \circ \mathsf{inl}(x) = f(x).$$

Similarly

$$\big(\alpha \circ \mathsf{inr}, p_\alpha \centerdot \mathsf{ap}_\alpha(\mathsf{glue})\big) =_{\mathsf{Map}^*(B,C)} (g, p_g)$$

is equivalent to

$$\sum_{p:\alpha\circ\mathsf{inr}=g} \mathsf{transport}^{h\mapsto *_C = h(*_B)}(p, p_\alpha \centerdot \mathsf{ap}_\alpha(\mathsf{glue})) = p_g$$

and therefore we may assume that we are given $p_r : \alpha \circ \mathsf{inr} = g$, such that

$$\mathsf{transport}^{h\mapsto *_C = h(*_B)}(p_r, p_\alpha \centerdot \mathsf{ap}_\alpha(\mathsf{glue})) = p_g.$$

We use again theorem 3.26, lemma 3.11 and remark 3.5, by which we have

$$\begin{aligned}\mathsf{transport}^{h\mapsto *_C = h(*_B)}(p_r, p_\alpha \centerdot \mathsf{ap}_\alpha(\mathsf{glue})) &= \\ = p_\alpha \centerdot \mathsf{ap}_\alpha(\mathsf{glue}) \centerdot \mathsf{ap}_{\mathsf{eval}(*_B)}(p_r) &= \\ = p_\alpha \centerdot \mathsf{ap}_\alpha(\mathsf{glue}) \centerdot \mathsf{happly}(p_r)(*_B)&\end{aligned}$$

Therefore, when we abbreviate $p_{\alpha\circ\mathsf{inr}} \equiv p_\alpha \centerdot \mathsf{ap}_\alpha(\mathsf{glue})$, we have

$$p_{\alpha\circ\mathsf{inr}} \centerdot \mathsf{happly}(p_r)(*_B) = p_f \tag{16}$$

and we note that

$$\mathsf{happly}(p_r) : \prod_{y:B} \alpha \circ \mathsf{inr}(y) = g(y).$$

By lemma 5.6, $\mathsf{inl} \vee^c \mathsf{inr}$ is the identity function and using lemma 5.7, we get:

$$\alpha \sim \alpha \circ (\mathsf{inl} \vee^c \mathsf{inr}) \sim (\alpha \circ \mathsf{inl}) \vee^c (\alpha \circ \mathsf{inr})$$

Lastly, by 4.24, equations (16) and (15) suffice to inhabit:

$$(\alpha \circ \mathsf{inl}) \vee^c (\alpha \circ \mathsf{inr}) \sim f \vee^c g$$

Then, function extensionality shows uniqueness. □

## 6. The smash product

Next we consider the *smash product* of pointed types. The definition of the smash product relies on the product $A \times B$ from section 2 of two pointed

types $A$ and $B$. We make $A \times B$ a pointed type by choosing $(*_A, *_B)$ as its base-point.

*Remark* 6.1. We note that, if we are given base-point preserving maps $(\alpha, p_\alpha) : \mathsf{Map}^*(A, A')$ and $(\beta, p_\beta) : \mathsf{Map}^*(B, B')$, then, by corollary 3.18, we have that $\mathsf{pair}^=(p_\alpha, p_\beta)$ proves that $(\alpha, \beta)$ is base-point preserving.

**Definition 6.2.** Assume we are given two pointed types $A, B : \mathcal{U}$. We let $\mathsf{inj}_l : A \to A \times B$ be the left injection, defined by the equation

$$\mathsf{inj}_l(x) :\equiv (x, *_B),$$

which is base-point preserving by $\mathsf{refl}_{(*_A, *_B)}$ and similarly $\mathsf{inj}_r : B \to A \times B$ the right injection, defined by the equation

$$\mathsf{inj}_r(y) :\equiv (*_A, y),$$

which is base-point preserving by $\mathsf{refl}_{(*_A, *_B)}$ as well. The smash product $A \wedge B$ of $A$ and $B$ is the pushout of the following square:

$$\begin{array}{ccc} A \vee B & \xrightarrow{\mathsf{inj}_l \vee^c \mathsf{inj}_r} & A \times B \\ \downarrow & \mathsf{Glue} & \downarrow{\scriptstyle [\_,\_]} \\ \mathbf{1} & \xrightarrow[\mathsf{inl}]{} & A \wedge B \end{array}$$

We make $A \wedge B$ a pointed type, by setting $*_{A \vee B} :\equiv \mathsf{inl}(\star)$

**Proposition 6.3.** *Assume we are given two pointed types $A, B : \mathcal{U}$. Then there is an equivalence*

$$A \wedge B \simeq B \wedge A$$

*Proof.* We denote by

- $\mathsf{inj}_l : A \to A \times B$,
- $\mathsf{inj}_r : B \to A \times B$,
- $\mathsf{inj}'_l : B \to B \times A$ and
- $\mathsf{inj}'_r : A \to B \times A$

the respective injections, by

$$\delta : A \vee B \to B \vee A$$

the equivalence from lemma 5.3 and lastly define

$$\gamma : A \times B \to B \times A$$

by the equation

$$\gamma(x, y) :\equiv (y, x),$$

which obviously is an equivalence. By theorem 4.29, in order to proof the above proposition, it suffices to show that

$$\prod_{w:A\vee B} (\mathsf{inj}'_l \vee^c \mathsf{inj}'_r) \circ \delta(w) = \gamma \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)(w).$$

We compute

$$(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r) \circ \delta \circ \mathsf{inl}_{A\vee B}(x) \equiv (\mathsf{inj}'_l \vee^c \mathsf{inj}'_r)\big(\mathsf{inr}_{B\vee A}(x)\big) \equiv (*_B, x),$$
$$(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r) \circ \delta \circ \mathsf{inr}_{A\vee B}(y) \equiv (\mathsf{inj}'_l \vee^c \mathsf{inj}'_r)\big(\mathsf{inl}_{B\vee A}(y)\big) \equiv (y, *_A),$$
$$\gamma \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)\big(\mathsf{inl}_{A\vee B}(x)\big) \equiv \gamma(x, *_B) \equiv (*_B, x),$$

and

$$\gamma \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)\big(\mathsf{inr}_{A\vee B}(y)\big) \equiv \gamma(*_A, y) \equiv (y, *_A).$$

Therefore, over the points we can proof the claim by reflexivity. Then it is left to show that

$$\mathsf{transport}^{w\mapsto(\mathsf{inj}'_l\vee^c\mathsf{inj}'_r)\circ\delta(w)=\gamma\circ(\mathsf{inj}_l\vee^c\mathsf{inj}_r)(w)}(\mathsf{glue}_{A\vee B}, \mathsf{refl}_{(*_B,x)}) = \mathsf{refl}_{(y,*_A)}.$$

But since the injections are base-point preserving by reflexivity and

$$\mathsf{ap}_\delta(\mathsf{glue}_{A\vee B}) = \mathsf{glue}_{B\vee A},$$

this follows immediately from theorem 3.26. □

*Remark* 6.4. The smash product can be thought of the product where all pairs for which the first coordinate is the base-point and all pairs for which the second coordinate is the base-point are connected by a path to an external point (the base-point of the smash product). The ambiguity of this description lies in the fact that, as it stands, it gives two paths when both coordinates are the base-point. We want to establish that these paths have to be homotopic to each other. So assume we are in the situation of definition 6.2. We consider the type family $P : A \vee B \to \mathcal{U}$, defined by the equation

$$P(w) :\equiv \big(*_{A\wedge B} = [\_, \_] \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)(w)\big).$$

Then it holds that

$$\mathsf{Glue} : \prod_{w:A\vee B} P(w)$$

and lemma 3.7 gives us an inhabitant

$$\mathsf{apd}_{\mathsf{Glue}}(\mathsf{glue}) : \mathsf{transport}^P\big(\mathsf{glue}, \mathsf{Glue}\big(\mathsf{inl}(*_A)\big)\big) = \mathsf{Glue}\big(\mathsf{inr}(*_B)\big).$$

From theorem 3.26 we get that the left hand side of the above equality is equal to

$$\mathsf{Glue}\big(\mathsf{inl}(*_A)\big) \cdot \mathsf{ap}_{[\_,\_]\circ(\mathsf{inj}_l \vee^c \mathsf{inj}_r)}(\mathsf{glue}) = \mathsf{Glue}\big(\mathsf{inr}(*_B)\big).$$

We recall that $\mathsf{inj}_l$ and $\mathsf{inj}_r$ are base-point preserving by reflexivity and therefore, by the definitions of $\mathsf{Col}$ in 11 and the wedge of functions from lemma 5.4, we have that

$$\mathsf{ap}_{\mathsf{inj}_l \vee^c \mathsf{inj}_r}(\mathsf{glue}_{A\vee B}) = \mathsf{refl}_{(*_A,*_B)}.$$

But, using lemma 3.1, this means that

$$\mathsf{apd}_{\mathsf{Glue}}(\mathsf{glue}) : \mathsf{Glue}\big(\mathsf{inl}(*_A)\big) = \mathsf{Glue}\big(\mathsf{inr}(*_B)\big)$$

*Remark* 6.5. We want to take a closer look at the recursion principle for the smash product. We recall from definition 4.1 that, in order to define a function of type $A \wedge B \to C$ for some given type $C : \mathcal{U}$, we need to give a function

$$f : A \times B \to C$$

together with a dependent function

$$\mathsf{Com} : \prod_{w:A\vee B} *_{A\wedge B} = f \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)(w).$$

In other words, given some function $f : A \times B \to C$, we need to provide a dependent function $\mathsf{Com}$ that makes the square

$$\begin{array}{ccc} A \vee B & \xrightarrow{\mathsf{inj}_l \vee^c \mathsf{inj}_r} & A \times B \\ \Big\downarrow & \mathsf{Com} & \Big\downarrow f \\ \mathbf{1} & \longrightarrow & C \end{array}$$

commute. We claim that this can be done by giving the following data:

- For every $x : A$ a proof $\mathsf{BPP}(x) : *_C = f(x, *_B)$ that the function $\lambda y.f(x, y)$ is base-point preserving (in particular this means that $f$ is base-point preserving).
- For every $y : B$ a proof $\mathsf{CON}(y) : *_C = f(*_A, y)$, i.e. a proof that the constant function $(*_C)_B$ is pointwise equal to the function $\lambda y.f(*_A, y)$ (by function extensionality this also means that the functions are equal).
- A path of type $\mathsf{BPP}(*_A) = \mathsf{CON}(*_B)$ between the proof that $\lambda y.f(*_A, y)$ is base-point preserving and the proof that it is constant at $*_B$.

To see that this data really suffices, we note that the first two points are just the proof of commutativity over the points, that is we can set

- $\mathsf{Com}\big(\mathsf{inl}(x)\big) :\equiv \mathsf{BPP}(x)$ and
- $\mathsf{Com}\big(\mathsf{inr}(y)\big) :\equiv \mathsf{CON}(y)$.

To finish the definition of $\mathsf{Com}$, it is then left to show that

$$\mathsf{transport}^{w \mapsto *_C = f \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)(w)}(\mathsf{glue}, \mathsf{BPP}(*_A)) = \mathsf{CON}(*_B)$$

and by theorem 3.26 this means to show

$$\mathsf{BPP}(*_A) \cdot \mathsf{ap}_{f \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)}(\mathsf{glue}) = \mathsf{CON}(*_B). \tag{17}$$

As in the previous remark, since $\mathsf{inj}_l$ and $\mathsf{inj}_r$ are base-point preserving by reflexivity, the definitions of $\mathsf{Col}$ in 11 and the wedge of functions from lemma 5.4 give us that

$$\mathsf{ap}_{\mathsf{inj}_l \vee^c \mathsf{inj}_r}(\mathsf{glue}_{A \vee B}) = \mathsf{refl}_{(*_A, *_B)}.$$

Therefore, using lemma 3.1 and the computation rule from lemma 3.2, we get that

$$\mathsf{ap}_{f \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)}(\mathsf{glue}_{A \vee B})^{-1} \cdot \mathsf{BPP}(*_A) = \mathsf{refl}^{-1}_{f(*_A, *_B)} \cdot \mathsf{BPP}(*_A) = \mathsf{BPP}(*_A)$$

and hence, in order to show (17), we need to inhabit

$$\mathsf{BPP}(*_A) = \mathsf{CON}(*_B).$$

This is however inhabited by assumption. To summarize, the above data suffices to define a proof of commutativity

$$\mathsf{Com} : \prod_{w:A\vee B} *_C = f \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)(w)$$

and thereby a function $f' : A \wedge B \to C$, such that

- $f'(*_{A\wedge B}) \equiv *_C$
- $f'([x, y]) \equiv f(x, y)$
- $\mathsf{ap}_{f'}\big(\mathsf{Glue}(w)\big) = \mathsf{Com}(w)$

In particular, $f'$ is a pointed function, where $\mathsf{refl}_{*_C}$ proves that it is base-point preserving.

*Remark* 6.6. Let us assume that we are in the same situation as in remark 6.5, i.e. that we are given pointed types $A, B, C : \mathcal{U}$ and some function $f : A \times B \to C$ such that we have:

- For every $x : A$ a proof $\mathsf{BPP}(x) : *_C = f(x, *_B)$ that the function $\lambda y.f(x, y)$ is base-point preserving (in particular this means that $f$ is base-point preserving).
- For every $y : B$ a proof $\mathsf{CON}(y) : *_C = f(*_A, y)$, i.e. a proof that the constant function $(*_C)_B$ is pointwise equal to the function $\lambda y.f(*_A, y)$ (by function extensionality this also means that the functions are equal).
- A path of type $\mathsf{BPP}(*_A) = \mathsf{CON}(*_B)$ between the proof that $\lambda y.f(*_A, y)$ is base-point preserving and the proof that it is constant at $*_B$.

We claim that this data suffices to define an inhabitant of

$$\mathsf{Map}^*\big(A, \mathsf{Map}^*(B, C)\big).$$

It is obvious that for every $x : A$ it holds that

$$\big(\lambda y.f(x, y), \mathsf{BPP}(x)\big) : \mathsf{Map}^*(B, C).$$

Then, what is left is to give a proof that the *pointed* function $\lambda y.f(*_A, y)$ (where $\mathsf{BPP}(*_A)$ proves that it is base-point preserving) is equal to the base-point in $\mathsf{Map}^*(B, C)$. But to this end we can use remark 4.14, where $\mathsf{functExt}(\mathsf{CON})$ proves that $\lambda y.f(*_A, y)$ is constant together with the fact that

happly is left-inverse to functExt and then we can use the last point in the above list.

**Proposition 6.7.** *Assume we are given three pointed types $A, B, C : \mathcal{U}$. Then there is an equivalence:*

$$\mathsf{Map}^*(A \wedge B, C) \simeq \mathsf{Map}^*\big(A, \mathsf{Map}^*(B, C)\big)$$

*Proof.* Step 1: We firstly assume that we are given some base-point preserving map

$$(f, p_f) : \mathsf{Map}^*(A \wedge B, C).$$

Then we can compose

$$f : A \wedge B \to C$$

with the bracket

$$[\_, \_] : A \times B \to A \wedge B$$

and this gives a function

$$f \circ [\_, \_] : A \times B \to C.$$

Next we consider the type family

$$P : A \vee B \to \mathcal{U},$$

given by the defining equation

$$P(w) :\equiv \big(*_{A\wedge B} = [\_, \_] \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)(w)\big)$$

and note that for every $w : A \vee B$ we have an inhabitant

$$\mathsf{Glue}(w) : P(w).$$

Additionally we have an inhabitant

$$p_f : *_C = f(*_{A\wedge B})$$

and therefore we can define a dependent function $\mathsf{F}$ of type

$$\prod_{w:A\vee B} \big(*_C = f \circ [\_, \_] \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)(w)\big)$$

by the defining equation

$$\mathsf{F}(w) :\equiv p_f \centerdot \mathsf{ap}_f\big(\mathsf{Glue}(w)\big). \tag{18}$$

In particular, since it holds for every $x : A$ that

$$(\mathsf{inj}_l \vee^c \mathsf{inj}_r)\big(\mathsf{inl}(x)\big) \equiv (x, *_B),$$

we have for every $x : A$ that

$$\mathsf{F}\big(\mathsf{inl}(x)\big) : *_C = f([x, *_B])$$

proves that the function

$$\lambda y.f([x, y])$$

is base-point preserving and this gives an inhabitant of

$$A \to \mathsf{Map}^*(B, C).$$

Next we note that since it holds for every $y : B$ that

$$(\mathsf{inj}_l \vee^c \mathsf{inj}_r)\big(\mathsf{inr}(y)\big) \equiv (*_A, y),$$

we have for every $y : B$ that

$$\mathsf{F}\big(\mathsf{inr}(y)\big) : *_C = f([*_A, y])$$

proves that the constant function $(*_C)_B$ is pointwise equal to

$$\lambda y.f([*_A, y]).$$

Lastly we note that

$$\mathsf{apd}_{\mathsf{F}}(\mathsf{glue}) : \mathsf{transport}^P(\mathsf{glue}, \mathsf{F}\big(\mathsf{inl}(*_A)\big)) = \mathsf{F}\big(\mathsf{inr}(*_B)\big)$$

and by theorem 3.26 this is equivalent to

$$\mathsf{F}\big(\mathsf{inl}(*_A)\big) \cdot \mathsf{ap}_{f \circ [\_,\_] \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)}(\mathsf{glue}) = \mathsf{F}\big(\mathsf{inr}(*_B)\big).$$

But since $\mathsf{inj}_l$ and $\mathsf{inj}_r$ are base-point preserving by reflexivity and therefore, by the definitions of $\mathsf{Col}$ in 11 and the wedge of functions from lemma 5.4, we have that

$$\mathsf{ap}_{\mathsf{inj}_l \vee^c \mathsf{inj}_r}(\mathsf{glue}_{A \vee B}) = \mathsf{refl}_{(*_A, *_B)},$$

this gives that

$$\mathsf{F}\big(\mathsf{inl}(*_A)\big) = \mathsf{F}\big(\mathsf{inr}(*_B)\big).$$

By 6.6, this shows that the function $\lambda y.f([*_A, y])$ is equal to the base-point in $\mathsf{Map}^*(B, C)$. Putting everything together, we showed that

$$f \circ [\_, \_] : A \times B \to C$$

gives rise to an inhabitant of

$$\mathsf{Map}^*\big(A, \mathsf{Map}^*(B, C)\big).$$

Step 2: Conversely, assume that we are given

$$(g, p_g) : \mathsf{Map}^*\big(A, \mathsf{Map}^*(B, C)\big).$$

We firstly note that

$$g : A \to \mathsf{Map}^*(B, C),$$

which means that the projection to the first coordinate gives for every $x : A$ a function

$$\pi_1\big(g(x)\big) : B \to C$$

and the projection to the second coordinate proves that this function is base-point preserving

$$\pi_2\big(g(x)\big) : *_C = \pi_1\big(g(x)\big)(*_B).$$

Secondly we note that

$$p_g : g(*_A) =_{\mathsf{Map}^*(B,C)} \big((*_C)_B, \mathsf{refl}_{*_C}\big)$$

and by theorem 3.16, from $p_g$ we get a path

$$p_g^l : \pi_1\big(g(*_A)\big) = (*_C)_B$$

such that

$$p_g^r : \mathsf{transport}^{h \mapsto *_C = h(*_B)}(p_g^l, \pi_2\big(g(*_A)\big)) = \mathsf{refl}_{*_C}.$$

By theorem 3.26 we have that the left side of this equality is equal to

$$\pi_2\big(g(*_A)\big) \cdot \mathsf{ap}_{\mathsf{eval}(*_B)}(p_g^l)$$

and then we can use 3.11, which gives us that

$$\pi_2\big(g(*_A)\big) \cdot \mathsf{ap}_{\mathsf{eval}(*_B)}(p_g^l) = \pi_2\big(g(*_A)\big) \cdot \mathsf{happly}(p_g^l)(*_B)$$

Putting everything together, we get an inhabitant of

$$\pi_2\big(g(*_A)\big) \centerdot \mathsf{happly}(p_g^l)(*_B) = \mathsf{refl}_{*_C}$$

and concatenating with $\mathsf{happly}(p_g^l)(*_B)^{-1}$ on both sides gives

$$\pi_2\big(g(*_A)\big) = \mathsf{happly}(p_g^l)(*_B)^{-1}.$$

We summarize, what we established so far in this step.

- We have for every $x : A$ a proof $\pi_2\big(g(x)\big) : *_C = \pi_1\big(g(x)\big)(*_B)$ that $\lambda y.\pi_1\big(g(x)\big)(y)$ is base-point preserving.
- For every $y : B$ a proof $\mathsf{happly}(p_g^l)(y)^{-1} : *_C = \pi_1\big(g(*_A)\big)(y)$, i.e. a proof that the constant function $(*_C)_B$ is pointwise equal to the function $\lambda y.\pi_1\big(g(*_A)\big)(y)$.
- A path of type $\pi_2\big(g(*_A)\big) = \mathsf{happly}(p_g^l)(*_B)^{-1}$ between the proof that $\lambda y.\pi_1\big(g(*_A)\big)(y)$ is base-point preserving and the proof that it is constant at $*_B$.

By remark 6.5 (and using the recursion principle for product types), this gives rise to a function $f'$ of type $A \wedge B \to C$.

Step 3: Next we want to show that what we defined is indeed an equivalence. So let us assume that we start of with an inhabitant

$$(f, p_f) : \mathsf{Map}^*(A \wedge B, C),$$

as in the first step. There we defined

$$\mathsf{F}(w) :\equiv p_f \centerdot \mathsf{ap}_f\big(\mathsf{Glue}(w)\big)$$

and then established that it holds for every $x : A$ that

$$\mathsf{F}\big(\mathsf{inl}(x)\big) : *_C = f([x, *_B])$$

proves that the function

$$\lambda y.f([x, y])$$

is base-point preserving and that

$$\lambda y.\mathsf{F}\big(\mathsf{inr}(y)\big) : \prod_{y:B} *_C = f([*_A, y])$$

proves that the constant function $(*_C)_B$ is pointwise equal to

$$\lambda y.f([*_A, y]).$$

Together with $\mathsf{apd}_{\mathsf{F}}(\mathsf{glue})$ this gave rise to an inhabitant of

$$\mathsf{Map}^*\big(A, \mathsf{Map}^*(B, C)\big),$$

by remark 6.6. Then, in step 2, we used remark 6.5, by which the very same data suffices to define a function $f' : A \wedge B \to C$ by the recursion principle of the smash product, such that:

- $f'(*_{A\wedge B}) \equiv *_C$
- $f'([x, y]) \equiv f(x, y)$
- $\mathsf{ap}_{f'}\big(\mathsf{Glue}(w)\big) = \mathsf{F}(w)$

We want to show that this functions is the same one that we started with. So let us consider the type family

$$P : A \wedge B \to \mathcal{U},$$

defined by the equation

$$P(s) :\equiv f(s) = f'(s).$$

We aim to inhabit the type

$$\prod_{x:A\wedge B} P(s).$$

Firstly we check the definition of $f'$ to compute

$$P(*_{A\wedge B}) \equiv f(*_{A\wedge B}) = *_C.$$

Therefore we can set

$$p_f^{-1} : P(*_{A\wedge B}).$$

Similarly we have

$$f'([x, y]) \equiv f([x, y])$$

and therefore we can set

$$\mathsf{refl}_{f([x,y])} : P([x, y]).$$

It is then left to show for every $w : A \vee B$ that

$$\mathsf{transport}^P(\mathsf{Glue}(w), p_f^{-1}) = \mathsf{refl}_{f([x,y])}.$$

By theorem 3.26 this means to show that

$$\mathsf{ap}_f\big(\mathsf{Glue}(w)\big)^{-1} \cdot p_f^{-1} \cdot \mathsf{ap}_{f'}\big(\mathsf{Glue}(w)\big) = \mathsf{refl}_{f([x,y])}.$$

But we have again by definition that

$$\mathsf{ap}_{f'}\big(\mathsf{Glue}(w)\big) = \mathsf{F}(w)$$

and

$$\mathsf{F}(w) \equiv p_f \centerdot \mathsf{ap}_f\big(\mathsf{Glue}(w)\big).$$

Step 4: In the converse direction we started of with

$$(g, p_g) : \mathsf{Map}^*\big(A, \mathsf{Map}^*(B, C)\big)$$

and then derived data as in remark 6.5 to define a function $f' : A \wedge B \to C$. In particular we constructed a path of type $\delta : \pi_2\big(g(*_A)\big) = \mathsf{happly}(p_g^l)(*_B)^{-1}$ (we did not write it down explicitly but we showed that such a path can be constructed). Next we defined a dependent function by the equation

$$\mathsf{F}'(w) :\equiv p_{f'} \centerdot \mathsf{ap}_{f'}\big(\mathsf{Glue}(w)\big),$$

but as noted in the remark, $f'$ is pointed by reflexivity and therefore we have for every $w : A \vee B$ that

$$\mathsf{F}'(w) = \mathsf{ap}_{f'}\big(\mathsf{Glue}(w)\big).$$

In the first step we used that composition of $\mathsf{F}'$ with $\mathsf{inl}$ and $\mathsf{inr}$ gave proofs that

$$f' \circ [\_, \_] : A \times B \to C$$

is base-point preserving for every $x : A$ and constant if $x \equiv *_A$. Together with $\mathsf{apd}_{\mathsf{F}'}(\mathsf{glue})$, remark 6.6 gave us an inhabitant of

$$\mathsf{Map}^*\big(A, \mathsf{Map}^*(B, C)\big).$$

But by construction we have firstly that

$$\mathsf{ap}_{f'}\Big(\mathsf{Glue}\big(\mathsf{inl}(x)\big)\Big) = \pi_2\big(g(x)\big),$$

secondly that

$$\mathsf{ap}_{f'}\Big(\mathsf{Glue}\big(\mathsf{inr}(y)\big)\Big) = \mathsf{happly}(p_g^l)(y)^{-1}$$

and that lastly that

$$\mathsf{apd}_{\mathsf{F}'}(\mathsf{glue}) = \delta$$

Therefore this inhabitant is determined by the same data as $(g, p_g)$. □

**Lemma 6.8.** *Assume we are given two pointed types $A, B : \mathcal{U}$ together with base-point preserving maps $(\alpha, p_\alpha) : \mathsf{Map}^*(A, A')$ and $(\beta, p_\beta) : \mathsf{Map}^*(B, B')$. We let*

- $\mathsf{inl} : A \to A \vee B$
- $\mathsf{inr} : B \to A \vee B$
- $\mathsf{inj}_l : A \to A \times B$
- $\mathsf{inj}_r : B \to A \times B$
- $\mathsf{Glue} : \prod_{(w:A\vee B)} *_{A\wedge B} = (\mathsf{inj}_l \vee^c \mathsf{inj}_r)(w)$
- $\mathsf{inl}' : A' \to A' \vee B'$
- $\mathsf{inr}' : B' \to A' \vee B'$
- $\mathsf{inj}'_l : A' \to A' \times B'$
- $\mathsf{inj}'_r : B' \to A' \times B'$
- $\mathsf{Glue}' : \prod_{(w':A'\vee B')} *_{A'\wedge B'} = (\mathsf{inj}'_l \vee^c \mathsf{inj}'_r)(w')$

*There is a canonical function $\alpha \wedge \beta : A \wedge B \to A' \wedge B'$, called the **smash** of $\alpha$ and $\beta$, such that*

- $(\alpha \wedge \beta)(*_{A\wedge B}) \equiv *_{A'\wedge B'}$
- $(\alpha \wedge \beta)([x, y]) \equiv [\alpha(x), \beta(y)]$
- $\mathsf{ap}_{(\alpha\wedge\beta)}\big(\mathsf{Glue}\big(\mathsf{inl}(x)\big)\big) = \mathsf{Glue}'\big(\mathsf{inl}'\big(\alpha(x)\big)\big) \cdot \mathsf{ap}_{[\_,\_]}\big(\mathsf{pair}^=(\mathsf{refl}_{\alpha(x)}, p_\beta)\big)$
- $\mathsf{ap}_{(\alpha\wedge\beta)}\big(\mathsf{Glue}\big(\mathsf{inr}(y)\big)\big) = \mathsf{Glue}'\big(\mathsf{inr}'\big(\beta(y)\big)\big) \cdot \mathsf{ap}_{[\_,\_]}\big(\mathsf{pair}^=(p_\alpha, \mathsf{refl}_{\beta(x)})\big)$

*What is more, $\alpha \wedge \beta$ is base-point preserving by reflexivity.*

*Proof.* We want to define a proof of commutativity $\mathsf{Com}$ for the right square in the diagram:

$$\begin{array}{ccccc}
\mathbf{1} & \longleftarrow & A \vee B & \xrightarrow{(\mathsf{inj}_l \vee^c \mathsf{inj}_r)} & A \times B \\
\Big\downarrow{\scriptstyle \star} & \mathsf{refl}_\star & \Big\downarrow{\scriptstyle \alpha\vee\beta} & & \Big\downarrow{\scriptstyle (\alpha,\beta)} \\
\mathbf{1} & \longleftarrow & A' \vee B' & \xrightarrow[(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r)]{} & A' \times B'
\end{array}$$

To this end we consider the type family

$$P : A \vee B \to \mathcal{U},$$

defined by the equation

$$P(w) :\equiv (\mathsf{inj}'_l \vee^c \mathsf{inj}'_r) \circ (\alpha \vee \beta)(w) = (\alpha, \beta) \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)(w).$$

We compute:

$$(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r) \circ (\alpha \vee \beta)\big(\mathsf{inl}(x)\big) \equiv (\mathsf{inj}'_l \vee^c \mathsf{inj}'_r)\Big(\mathsf{inl}'\big(\alpha(x)\big)\Big) \equiv (\alpha(x), *_{B'})$$

$$(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r) \circ (\alpha \vee \beta)\big(\mathsf{inr}(y)\big) \equiv (\mathsf{inj}'_l \vee^c \mathsf{inj}'_r)\Big(\mathsf{inr}'\big(\beta(y)\big)\Big) \equiv \big(*_{A'}, \beta(y)\big)$$

$$(\alpha, \beta) \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)\big(\mathsf{inl}(x)\big) \equiv (\alpha, \beta)(x, *_B) \equiv \big(\alpha(x), \beta(*_B)\big)$$

$$(\alpha, \beta) \circ (\mathsf{inj}_l \vee^c \mathsf{inj}_r)\big(\mathsf{inr}(y)\big) \equiv (\alpha, \beta)(*_A, y) \equiv \big(\alpha(*_A), \beta(y)\big)$$

Therefore we can define

$$\mathsf{pair}^{=}(\mathsf{refl}_{\alpha(x)}, p_\beta) : P\big(\mathsf{inl}(x)\big)$$

and

$$\mathsf{pair}^{=}(p_\alpha, \mathsf{refl}_{\beta(y)}) : P\big(\mathsf{inr}(y)\big).$$

Then it is left to show that

$$\mathsf{transport}^P(\mathsf{glue}, \mathsf{pair}^{=}(\mathsf{refl}_{\alpha(x)}, p_\beta)) = \mathsf{pair}^{=}(p_\alpha, \mathsf{refl}_{\beta(y)}).$$

By theorem 3.26 this means to show

$$\mathsf{ap}_{(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r) \circ (\alpha \vee \beta)}(\mathsf{glue})^{-1} \cdot \mathsf{pair}^{=}(\mathsf{refl}_{\alpha(x)}, p_\beta) = \mathsf{pair}^{=}(p_\alpha, \mathsf{refl}_{\beta(y)}) \tag{19}$$

and hence we need to compute

$$\mathsf{ap}_{(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r) \circ (\alpha \vee \beta)}(\mathsf{glue}).$$

By lemma 4.27 we get that

$$(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r) \circ (\alpha \vee \beta) \sim (\mathsf{inj}'_l \circ \alpha) \vee^c (\mathsf{inj}'_r \circ \beta).$$

We then use that

$$(\mathsf{inj}'_l \circ \alpha)$$

is base-point preserving by

$$\mathsf{ap}_{\mathsf{inj}'_l}(p_\alpha)$$

and using the equivalence from corollary 3.18, lemma 3.21 and remark 3.5, we get

$$\mathsf{ap}_{\mathsf{inj}'_l}(p_\alpha) = \mathsf{pair}^{=}(p_\alpha, \mathsf{refl}_{*_{B'}}).$$

Similarly we have that

$$(\mathsf{inj}'_r \circ \beta)$$

is base-point preserving by

$$\mathsf{ap}_{\mathsf{inj}'_r}(p_\beta) = \mathsf{pair}^{=}(\mathsf{refl}_{*_A}, p_\beta).$$

Checking the definition of a wedge of functions in lemma 5.4, we conclude that

$$\mathsf{ap}_{(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r)\circ(\alpha\vee\beta)}(\mathsf{glue})$$

is equal to

$$\mathsf{pair}^{=}(p_\alpha, \mathsf{refl}_{*_{B'}}) \cdot \mathsf{pair}^{=}(\mathsf{refl}_{*_A}, p_\beta).$$

We use lemma 3.20, by which we have that

$$\mathsf{pair}^{=}(p_\alpha, \mathsf{refl}_{*_{B'}}) \cdot \mathsf{pair}^{=}(\mathsf{refl}_{*_A}, p_\beta) = \mathsf{pair}^{=}(p_\alpha, p_\beta)$$

and when we plug this in (19), we get

$$\mathsf{pair}^{=}(p_\alpha, p_\beta)^{-1} \cdot \mathsf{pair}^{=}(\mathsf{refl}_{\alpha(x)}, p_\beta) = \mathsf{pair}^{=}(p_\alpha, \mathsf{refl}_{\beta(y)}).$$

Yet again, this can be proven by lemma 3.20 and this gives us the required proof $\mathsf{Com}$ of commutativity and we define $\alpha \wedge \beta$ to be the pushout function of the two commutative squares:

$$\begin{array}{ccccc}
\mathbf{1} & \longleftarrow & A \vee B & \xrightarrow{(\mathsf{inj}_l \vee^c \mathsf{inj}_r)} & A \times B \\
{\scriptstyle\star}\downarrow & \mathsf{refl}_\star & \downarrow{\scriptstyle \alpha\vee\beta} & \mathsf{Com} & \downarrow{\scriptstyle(\alpha,\beta)} \\
\mathbf{1} & \longleftarrow & A' \vee B' & \xrightarrow[(\mathsf{inj}'_l \vee^c \mathsf{inj}'_r)]{} & B'
\end{array}$$

Lastly we check that

$$(\alpha \wedge \beta)(*_{A\wedge B}) \equiv *_{A'\wedge B'},$$

hence it holds indeed that reflexivity proves that $(\alpha \wedge \beta)$ is base-point preserving. □

*Remark* 6.9. The following proof is incomplete. In step three and four we will want to show an equality between pointed functions

$$f, g : \mathsf{Map}^*(A, \mathsf{Map}^*(B, C)),$$

but we came only so far as to show that there is an

$$\mathsf{H} : \pi_1(f) = \pi_1(g).$$

What is missing is to show that

$$\pi_2(f) \cdot \mathsf{H}(*_A) = \pi_2(g),$$

according to remark 4.14.

**Proposition 6.10.** *Assume we are given two pointed types $A, B$ and $C$. Then there is an equivalence:*

$$(A \vee B) \wedge C \simeq (A \wedge C) \vee (B \wedge C)$$

*Proof.* Step 1: Defining a function of type

$$(A \wedge C) \vee (B \wedge C) \to (A \vee B) \wedge C$$

is easy, since we have at hand the pointed functions:

- $\mathsf{inl}_{A \vee B} : A \to A \vee B$ and
- $\mathsf{inr}_{A \vee B} : B \to A \vee B$

and therefore we can use lemmas 6.8 and 5.4 to define

$$\Phi :\equiv (\mathsf{inl}_{A \vee B} \wedge \mathsf{id}_C) \vee^c (\mathsf{inr}_{A \vee B} \wedge \mathsf{id}_C),$$

which has appropriate type

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

Step 2: Next, we want to give for any $z : C$ a base-point preserving function $f_z$ of type:

$$A \vee B \to (A \wedge C) \vee (B \wedge C)$$

To this end, we aim to define for every $z : C$ a pointed map $f_z^l$ in

$$\mathsf{Map}^*\big(A, (A \wedge C) \vee (B \wedge C)\big)$$

and a pointed map $f_z^r$ in

$$\mathsf{Map}^*\big(B, (A \wedge C) \vee (B \wedge C)\big).$$

We start with $f_z^l$, which is defined by the equation

$$f_z^l(x) :\equiv \mathsf{inl}_{(A \wedge C) \vee (B \wedge C)}([x, z]).$$

We recall that we chose

$$*_{(A\wedge C)\vee(B\wedge C)} :\equiv \mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}(*_{A\wedge C})$$

as base-point. Now, since

$$\mathsf{Glue}_{A\wedge C}\big(\mathsf{inr}_{A\vee C}(z)\big) : *_{A\wedge C} = [*_A, z],$$

we get that

$$\mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inr}_{A\vee C}(z)\big)\Big)$$

has type

$$\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}(*_{A\wedge C}) = \mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}([*_A, z])$$

and therefore proves that $f^l_z \equiv \lambda x.\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}([x, z])$ is base-point preserving for every $z : C$. Similarly we can define a function by the equation

$$f^r_z(y) :\equiv \mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}([y, z])$$

and then we can use that

$$\mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\big(\mathsf{Glue}_{B\wedge C}(\mathsf{inr}_{B\vee C}(z)\big)$$

has type

$$\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}(*_{B\wedge C}) = \mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}([*_B, z])$$

and that

$$\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} : \mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}(*_{A\wedge C}) = \mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}(*_{B\wedge C})$$

to define a proof that $f^r_z \equiv \lambda y.\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}([y, z])$ is base-point preserving for every $z : C$ by

$$\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\big(\mathsf{Glue}_{B\wedge C}(\mathsf{inr}_{B\vee C}(z)\big).$$

Now we are able to define for every $z : C$ an inhabitant of

$$A \vee B \to \big((A \wedge C) \vee (B \wedge C)\big)$$

by

$$f_z :\equiv f^l_z \vee^c f^r_z.$$

We note that for every $z : C$, by definition of $f_z$ we have

$$f_z(*_{A\vee B}) \equiv f_z\big(\mathsf{inl}_{A\vee B}(*_A)\big) \equiv f^l_z(*_A)$$

and hence $f_z$ is pointed by

$$\mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inr}_{A\vee C}(z)\big)\Big).$$

Next we want to show that the constant function $(*_{(A\wedge C)\vee(B\wedge C)})_A$ is pointwise equal to $f_{*_C}$ and we do so by applying the induction principle of $A \vee B$. So let us consider the type family

$$P : A \vee B \to \mathcal{U},$$

defined by the equation

$$P(w) :\equiv *_{(A\wedge C)\vee(B\wedge C)} = f_{*_C}(w).$$

We note that for every $x : A$ we have that

$$\mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(x)\big)\Big)$$

has type

$$\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}(*_{A\wedge C}) = \mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}([x, *_C]),$$

i.e. proves that the constant function $(*_{(A\wedge C)\vee(B\wedge C)})_A$ is pointwise equal to

$$f^l_{*_C} \equiv \lambda x.\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}([x, *_C]).$$

Similarly, for any $y : B$, the term

$$\mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inl}_{B\vee C}(y)\big)\Big)$$

has type

$$\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}(*_{B\wedge C}) = \mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}([y, *_C])$$

and

$$\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)}$$

has type

$$\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}(*_{A\wedge C}) = \mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}(*_{B\wedge C}).$$

Hence we can define for every $y : B$ an inhabitant of type

$$\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}(*_{A\wedge C}) = \mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}([y, *_C])$$

i.e. a proof that the constant function $(*_{(A\wedge C)\vee(B\wedge C)})_A$ is pointwise equal to

$$f^r_{*_C} \equiv \lambda y.\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}([y, *_C])$$

by

$$\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inl}_{B\vee C}(y)\big)\Big).$$

Then we want to show that

$$\mathsf{transport}^P\Big(\mathsf{glue}_{A\vee B}, \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(*_A)\big)\Big)\Big) =$$
$$= \mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inl}_{B\vee C}(*_B)\big)\Big).$$

By theorem 3.26 this means to show that

$$\begin{aligned}(20)\qquad &\mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(*_A)\big)\Big) \cdot \mathsf{ap}_{f_{*_C}}(\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)}) = \\ &\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inl}_{B\vee C}(*_B)\big)\Big)\end{aligned}$$

But by definition of the wedge of functions (see lemma 5.4) we have that

$$\mathsf{ap}_{f_{*_C}}(\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)}) =$$
$$= \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inr}_{A\vee C}(*_C)\big)\Big)^{-1} \cdot$$
$$\cdot \mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inr}_{B\vee C}(*_C)\big)\Big)$$

and by remark 6.4 that

$$\mathsf{apd}_{\mathsf{Glue}_{A\wedge C}}(\mathsf{glue}_{A\vee C}) : \mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(*_A)\big) = \mathsf{Glue}_{A\wedge C}\big(\mathsf{inr}_{A\vee C}(*_C)\big),$$

as well as

$$\mathsf{apd}_{\mathsf{Glue}_{B\wedge C}}(\mathsf{glue}_{B\vee C}) : \mathsf{Glue}_{B\wedge C}\big(\mathsf{inl}_{B\vee C}(*_B)\big) = \mathsf{Glue}_{B\wedge C}\big(\mathsf{inr}_{B\vee C}(*_C)\big).$$

Hence, after canceling inverses, we can proof (20) by

$$\mathsf{refl}_{\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}(\mathsf{Glue}_{B\wedge C}(\mathsf{inl}_{B\vee C}(*_B)))} \cdot$$

This shows that that the constant function $(*_{(A\wedge C)\vee(B\wedge C)})_A$ is pointwise equal to $f_{*_C}$. We now put everything together. Firstly we can define

$$\lambda w.\lambda c.f_z(w) : (A \vee B) \to C \to (A \wedge C) \vee (B \wedge C)$$

and then, by the preceding discussion, the constant function $(*_{(A\wedge C)\vee(B\wedge C)})_C$ is pointwise equal to $\lambda z.f_z(*_{A\vee B})$. In particular, evaluated at $*_C$, this is proven by

$$\mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inr}_{A\vee C}(*_C)\big)\Big).$$

Secondly we showed that $\lambda z.f_z(w)$ is base-point preserving for every $w : A \vee B$ and, evaluated at $*_{A\vee B} \equiv \mathsf{inl}_{A\vee C}(*_A)$, this is proven by

$$\mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(*_A)\big)\Big).$$

By remark 6.4 we have that

$$\mathsf{apd}_{\mathsf{Glue}_{A\wedge C}}(\mathsf{glue}_{A\vee C}) : \mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee B}(*_A)\big) = \mathsf{Glue}_{A\wedge C}(\mathsf{inr}_{A\vee C}(*_C)$$

and hence, from lemma 3.3, we get a proof

$$\mathsf{ap}^2_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}(\mathsf{apd}_{\mathsf{Glue}_{A\wedge C}}(\mathsf{glue}_{A\vee C}))$$

that

$$\mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(*_A)\big)\Big) =$$
$$= \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inr}_{A\vee C}(*_C)\big)\Big).$$

By remark 6.6, this data suffices to define an inhabitant of

$$\mathsf{Map}^*\Big(A \vee B, \mathsf{Map}^*\big(C, (A \wedge C) \vee (B \wedge C)\big)\Big)$$

and by lemma 6.7, this corresponds to a function $\Psi$ of type

$$\mathsf{Map}^*\big((A \vee B) \wedge C, (A \wedge C) \vee (B \wedge C)\big),$$

such that over the points we have the following computation rules:

- $\Psi(*_{(A\vee B)\wedge C)}) :\equiv \mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}(*_{(A\wedge C)}) \equiv *_{(A\wedge C)\vee(B\wedge C)}$
- $\Psi([w, z]) :\equiv f_z(w)$

In particular, $\Psi$ is base-point preserving by reflexivity. Additionally we have

$$\mathsf{ap}_\Psi\Big(\mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inr}_{(A\vee B)\vee C}(z)\big)\Big) = \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inr}_{A\vee C}(z)\big)\Big),$$

$$\mathsf{ap}_\Psi\Big(\mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inl}_{(A\vee B)\vee C}(\mathsf{inl}_{A\vee B}(x))\big)\Big) =$$
$$= \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(x)\big)\Big)$$

and lastly

$$\mathsf{ap}_\Psi\Big(\mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inl}_{(A\vee B)\vee C}(\mathsf{inr}_{A\vee B}(y))\big)\Big) =$$
$$= \mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inl}_{B\vee C}(y)\big)\Big).$$

Step 3: Next, we aim to show that

$$\Psi \circ \big((\mathsf{inl}_{A\vee B} \wedge \mathsf{id}_C) \vee^c (\mathsf{inr}_{A\vee B} \wedge \mathsf{id}_C)\big) \sim \mathsf{id}_{(A\wedge C)\vee(B\wedge C)}.$$

By lemma 4.27, we have:

$$\Psi\circ\big((\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)\big) \sim \big(\Psi\circ(\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)\big)\vee^c\big(\Psi\circ(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)\big)$$

By Corollary 5.6, it suffices to show that the last function is pointwise equal to:

$$\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)} \vee^c \mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}$$

We compute for every $x : A$ and $z : C$ that

$$\Psi \circ (\mathsf{inl}_{A\vee B} \wedge \mathsf{id}_C)([x, z]) \equiv \Psi([\mathsf{inl}_{A\vee B}(x), z]) \equiv$$
$$\equiv f_z\big(\mathsf{inl}_{A\vee B}(x)\big) \equiv f^l_z(x) \equiv \mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}([x, z]).$$

We want to avoid using the induction principle for the smash product, which is why we consider the images of

$$\Psi \circ (\mathsf{inl}_{A\vee B} \wedge \mathsf{id}_C)$$

and

$$\Psi \circ (\mathsf{inr}_{A\vee B} \wedge \mathsf{id}_C)$$

under the equivalence of proposition 6.7 instead. We compute for every $x : A$ and $z : C$ that

$$\Psi \circ (\mathsf{inl}_{A\vee B} \wedge \mathsf{id}_C)([x, z]) \equiv \Psi([\mathsf{inl}_{A\vee B}(x), z]) \equiv$$
$$\equiv f_z\big(\mathsf{inl}_{A\vee B}(x)\big) \equiv f^l_z(x) \equiv \mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}([x, z]).$$

Next we use that $\Psi$ and $(\mathsf{inl}_{A\vee B} \wedge \mathsf{id}_C)$ and therefore also $\Psi \circ (\mathsf{inl}_{A\vee B} \wedge \mathsf{id}_C)$, as well as $\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}$ are base-point preserving by reflexivity and proceed as in equation 18, i.e. for every $w : A \vee C$ we consider the terms

$$\mathsf{F}_l(w) :\equiv \mathsf{ap}_{\Psi\circ(\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)}\big(\mathsf{Glue}_{A\wedge C}(w)\big)$$

and

$$\mathsf{G}_l(w) :\equiv \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\big(\mathsf{Glue}_{A\wedge C}(w)\big).$$

We check the definition of the smash of functions, lemma 6.8 and then, since $\mathsf{id}_C$ is base-point preserving by reflexivity, we get for every $x : A$ that

$$\mathsf{ap}_{(\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(x)\big)\Big) =$$
$$= \mathsf{Glue}_{(A\vee B)\wedge C}\Big(\mathsf{inl}_{(A\vee B)\vee C}\big(\mathsf{inl}_{A\vee B}(x)\big)\Big)$$

and therefore

$$\mathsf{ap}_{\Psi\circ(\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(x)\big)\Big) =$$
$$= \mathsf{ap}_{\Psi}\Big(\mathsf{Glue}_{(A\vee B)\wedge C}\Big(\mathsf{inl}_{(A\vee B)\vee C}\big(\mathsf{inl}_{A\vee B}(x)\big)\Big)\Big) =$$
$$= \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{A\vee C}(x)\big)\Big).$$

Hence we get that for every $x : A$ it holds:

$$\mathsf{F}_l\big(\mathsf{inl}_{A\vee C}(x)\big) = \mathsf{G}_l\big(\mathsf{inl}_{A\vee C}(x)\big)$$

Similarly, since also $\mathsf{inl}_{A\vee B}$ is base-point preserving by reflexivity, we have for every $z : C$ that

$$\mathsf{ap}_{(\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)}\big(\mathsf{Glue}_{A\wedge C}(\mathsf{inr}_{A\vee C}(z))\Big) =$$
$$= \mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inr}_{(A\vee B)\vee C}(z)\big)$$

and therefore

$$\mathsf{ap}_{\Psi\circ(\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)}\big(\mathsf{Glue}_{A\wedge C}(\mathsf{inr}_{A\vee C}(z))\Big) =$$
$$= \mathsf{ap}_{\Psi}\big(\mathsf{Glue}_{(A\vee B)\wedge C}(\mathsf{inr}_{(A\vee B)\vee C}(z))\Big) =$$
$$= \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\big(\mathsf{Glue}_{A\wedge C}(\mathsf{inr}_{A\vee C}(z))\Big).$$

Hence we get that also for every $z : C$ it holds that:

$$\mathsf{F}_l\big(\mathsf{inr}_{A\vee C}(z)\big) = \mathsf{G}_l\big(\mathsf{inr}_{A\vee C}(z)\big)$$

Similarly for every $y : B$ and $z : C$ we compute

$$\Psi \circ (\mathsf{inr}_{A\vee B} \wedge \mathsf{id}_C)([y, z]) \equiv \Psi([\mathsf{inr}_{A\vee B}(y), z]) \equiv$$
$$\equiv f_z\big(\mathsf{inr}_{A\vee B}(y)\big) \equiv f^r_z(y) \equiv \mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}([y, z]).$$

Then we use that $\Psi$ and $(\mathsf{inr}_{A\vee B} \wedge \mathsf{id}_C)$ and therefore also $\Psi \circ (\mathsf{inr}_{A\vee B} \wedge \mathsf{id}_C)$ are base-point preserving by reflexivity and $\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}$ is base-point

preserving by $\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)}$ and proceed again as in equation 18, i.e. for every $w : B \vee C$ we consider the terms

$$\mathsf{F}_r(w) :\equiv \mathsf{ap}_{\Psi\circ(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)}\big(\mathsf{Glue}_{B\wedge C}(w)\big)$$

and

$$\mathsf{G}_r(w) :\equiv \mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\big(\mathsf{Glue}_{B\wedge C}(w)\big).$$

We check again lemma 6.8 and then, since $\mathsf{id}_C$ is base-point preserving by reflexivity, we get that

$$\mathsf{ap}_{(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)}\big(\mathsf{Glue}_{B\wedge C}(\mathsf{inl}_{B\vee C}(y))\big) =$$
$$= \mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inl}_{(A\vee B)\vee C}(\mathsf{inr}_{A\vee B}(y))\big)$$

and therefore

$$\mathsf{ap}_{\Psi\circ(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inl}_{B\vee C}(y)\big)\Big) =$$
$$= \mathsf{ap}_{\Psi}\big(\mathsf{Glue}_{(A\vee B)\wedge C}(\mathsf{inl}_{(A\vee B)\vee C}(\mathsf{inr}_{A\vee B}(y))\big)\big) =$$
$$= \mathsf{glue}_{(A\wedge C)\vee(B\wedge C)} \cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inl}_{B\vee C}(y)\big)\Big).$$

Hence we get that for every $y : B$ it holds:

$$\mathsf{F}_r\big(\mathsf{inl}_{B\vee C}(y)\big) = \mathsf{G}_r\big(\mathsf{inl}_{B\vee C}(y)\big).$$

Next, using again the definition of the smash of functions (see lemma 6.8) together with the fact that $\mathsf{inr}_{A\vee B}$ is base-point preserving by $\mathsf{glue}_{A\vee B}$, we get for every $z : C$ that

$$\mathsf{ap}_{\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inr}_{B\vee C}(z)\big)\Big) =$$
$$= \mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inr}_{(A\vee B)\vee C}(z)\big) \cdot \mathsf{ap}_{[\_,\_]}\big(\mathsf{pair}^{=}(\mathsf{glue}_{A\vee B}, \mathsf{refl}_z)\big).$$

and therefore

(21)

$$\mathsf{ap}_{\Psi\circ(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)}\big(\mathsf{Glue}_{B\wedge C}(\mathsf{inr}_{A\vee C}(y))\big) =$$
$$= \mathsf{ap}_{\Psi}\Big(\mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inr}_{(A\vee B)\vee C}(z)\big) \cdot \mathsf{ap}_{[\_,\_]}\big(\mathsf{pair}^{=}(\mathsf{glue}_{A\vee B}, \mathsf{refl}_z)\big)\Big) =$$
$$= \mathsf{ap}_{\Psi}\Big(\mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inr}_{(A\vee B)\vee C}(z)\big)\Big) \cdot \mathsf{ap}_{\Psi\circ[\_,\_]}\big(\mathsf{pair}^{=}(\mathsf{glue}_{A\vee B}, \mathsf{refl}_z)\big).$$

We use lemma 3.22, by which we have that

$$\begin{aligned}\mathsf{ap}_{\Psi\circ[\_,\_]}\big(\mathsf{pair}^{=}(\mathsf{glue}_{A\vee B},\mathsf{refl}_z)\big) &= \\ = \mathsf{ap}_{\lambda w.\Psi([w,z])}(\mathsf{glue}_{A\vee B}).&\end{aligned}$$

We check the definition $\lambda w.\Psi([w,z]) \equiv f_z$ and then

$$\begin{aligned}\mathsf{ap}_{f_z}(\mathsf{glue}_{A\vee B}) &= \\ = \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}(\mathsf{Glue}_{A\wedge C}(\mathsf{inr}_{A\vee C}(z))\Big)^{-1}\cdot& \\ \cdot\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)}\cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}(\mathsf{Glue}_{B\wedge C}(\mathsf{inr}_{B\vee C}(z))\Big)&\end{aligned}$$

and by definition of $\Psi$ we have that

$$\begin{aligned}\mathsf{ap}_{\Psi}\Big(\mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inr}_{(A\vee B)\vee C}(z)\big)\Big) &= \\ = \mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}(\mathsf{Glue}_{A\wedge C}(\mathsf{inr}_{A\vee C}(z))\Big).&\end{aligned}$$

Hence the rightmost term of equation (21) is in turn equal to

$$\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)}\cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inr}_{B\vee C}(z)\big)\Big)$$

and this shows that for every $z : C$ it holds:

$$\mathsf{F}_r\big(\mathsf{inr}_{B\vee C}(z)\big) = \mathsf{G}_r\big(\mathsf{inr}_{B\vee C}(z)\big)$$

Step 4: Next we want to show that also

$$\big((\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge \mathsf{id}_C)\big)\circ\Psi \sim \mathsf{id}_{(A\vee B)\wedge C}.$$

As in step 3, we consider the image of

$$\big((\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge \mathsf{id}_C)\big)\circ\Psi$$

under the equivalence from proposition 6.7 and then, for every $w : A\vee B$ and $z : C$ we compute:

$$\begin{aligned}\big((\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge \mathsf{id}_C)\big)\circ\Psi([w,z]) &\equiv \\ \equiv \big((\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge \mathsf{id}_C)\big)\circ f_z(w) &\equiv \\ \equiv \big((\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge \mathsf{id}_C)\big)\circ (f_z^l\vee^c f_z^r)(w).&\end{aligned}$$

We quickly check the definitions

$$f^l_z(x) \equiv \mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}([x,z])$$

and

$$f^r_z(y) \equiv \mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}([y,z]).$$

Then, using lemma 5.7, we have that:

$$\begin{aligned}&\left((\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge \mathsf{id}_C)\right)\circ(f^l\vee^c f^r)\sim\\ &\sim\left(\lambda x.\lambda z.[\mathsf{inl}_{A\vee B}(x),z]\right)\vee^c\left(\lambda y.\lambda z.[\mathsf{inr}_{A\vee B}(y),z]\right)\end{aligned}$$

We note that we have for every $x : A$ that

$$\mathsf{ap}_{\left((\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)\right)\circ\Psi}\left(\mathsf{Glue}_{(A\vee B)\wedge C}(\mathsf{inl}_{(A\vee B)\vee C}(\mathsf{inl}_{A\vee B}(x)))\right)$$

proves that

$$\lambda z.[\mathsf{inl}_{A\vee B}(x),z]$$

is base-point preserving and by definition of $\Psi$ and the smash of functions

$$\begin{aligned}&\mathsf{ap}_{\left((\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)\right)\circ\Psi}\left(\mathsf{Glue}_{(A\vee B)\wedge C}\left(\mathsf{inl}_{(A\vee B)\vee C}\left(\mathsf{inl}_{A\vee B}(x)\right)\right)\right)=\\ &=\mathsf{ap}_{\left((\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)\right)}\left(\mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\left(\mathsf{Glue}_{A\wedge C}\left(\mathsf{inl}_{A\vee C}(x)\right)\right)\right)=\\ &=\mathsf{ap}_{(\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)}\left(\mathsf{Glue}_{A\wedge C}\left(\mathsf{inl}_{A\vee C}(x)\right)\right)=\mathsf{Glue}_{(A\vee B)\wedge C}\left(\mathsf{inl}_{(A\vee B)\vee C}\left(\mathsf{inl}_{A\vee B}(x)\right)\right).\end{aligned}$$

Similarly for every $y : B$ we have that

$$\mathsf{ap}_{\left((\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)\right)\circ\Psi}\left(\mathsf{Glue}_{(A\vee B)\wedge C}(\mathsf{inl}_{(A\vee B)\vee C}(\mathsf{inr}_{A\vee B}(y)))\right)$$

proves that

$$\lambda z.[\mathsf{inr}_{A\vee B}(y),z]$$

is base-point preserving and by definition of $\Psi$ and the smash of functions

$$\begin{aligned}&\mathsf{ap}_{\left((\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)\right)\circ\Psi}\left(\mathsf{Glue}_{(A\vee B)\wedge C}(\mathsf{inl}_{(A\vee B)\vee C}(\mathsf{inr}_{A\vee B}(y)))\right)=\\ &=\mathsf{ap}_{\left((\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)\right)}\left(\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)}\cdot\mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\left(\mathsf{Glue}_{B\wedge C}(\mathsf{inl}_{B\vee C}(y))\right)\right)=\\ &=\mathsf{ap}_{\left((\mathsf{inl}_{A\vee B}\wedge\mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)\right)}\left(\mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\left(\mathsf{Glue}_{B\wedge C}(\mathsf{inl}_{B\vee C}(y))\right)\right)=\\ &=\mathsf{ap}_{(\mathsf{inr}_{A\vee B}\wedge\mathsf{id}_C)}\left(\mathsf{Glue}_{B\wedge C}(\mathsf{inl}_{B\vee C}(y))\right)=\\ &=\mathsf{Glue}_{(A\vee B)\wedge C}\left(\mathsf{inl}_{(A\vee B)\vee C}(\mathsf{inr}_{A\vee B}(y))\right).\end{aligned}$$

Next we use remark 5.2 and since $\mathsf{inl}_{A\vee B}$ is base-point preserving by reflexivity, we get for every $z : C$ that

$$(\mathsf{Glue}_{(A\vee B)\wedge C}\big(\mathsf{inr}_{(A\vee B)\vee C}(z)\big)$$

proves that

$$\lambda z.[\mathsf{inl}_{A\vee B}(*_A), z]$$

is equal to the base-point. Secondly, since $\mathsf{inr}_{A\vee B}$ is base-point preserving by $\mathsf{glue}_{A\vee B}$ we compute

$$\mathsf{ap}_{\big((\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge \mathsf{id}_C)\big)\circ(f_z^l\vee^c f_z^r)}(\mathsf{glue}_{A\vee B}) =$$

$$= \mathsf{ap}_{\big((\mathsf{inl}_{A\vee B}\wedge \mathsf{id}_C)\vee^c(\mathsf{inr}_{A\vee B}\wedge \mathsf{id}_C)\big)}\Big(\mathsf{ap}_{\mathsf{inl}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{A\wedge C}\big(\mathsf{inr}_{A\vee C}(z)\big)\Big)^{-1}\cdot$$

$$\cdot\mathsf{glue}_{(A\wedge C)\vee(B\wedge C)}\cdot \mathsf{ap}_{\mathsf{inr}_{(A\wedge C)\vee(B\wedge C)}}\Big(\mathsf{Glue}_{B\wedge C}\big(\mathsf{inr}_{B\vee C}(z)\big)\Big)\Big) =$$

$$= \mathsf{Glue}_{(A\vee B)\wedge C}\Big(\mathsf{inl}_{(A\vee B)\vee C}\big(\mathsf{inl}_{A\vee B}(x)\big)\Big)^{-1}\cdot$$

$$\cdot\mathsf{Glue}_{(A\vee B)\wedge C}\Big(\mathsf{inl}_{(A\vee B)\vee C}\big(\mathsf{inr}_{A\vee B}(y)\big)\Big)\cdot \mathsf{ap}_{[\_,\_]}\big(\mathsf{pair}^{=}(\mathsf{glue}_{A\vee B}, \mathsf{refl}_z)\big).$$

Therefore, the proof that

$$\lambda z.[\mathsf{inr}_{A\vee B}(*_B), z]$$

is equal to the base-point for every $z : C$ is given by

$$\mathsf{Glue}_{(A\vee B)\wedge C}\Big(\mathsf{inl}_{(A\vee B)\vee C}\big(\mathsf{inr}_{A\vee B}(y)\big)\Big)\cdot \mathsf{ap}_{[\_,\_]}\big(\mathsf{pair}^{=}(\mathsf{glue}_{A\vee B}, \mathsf{refl}_z)\big).$$

□

## 7. Conclusion

We hope that the reader developed some intuition about basic concepts in homotopy type theory and how one translates ideas from homotopy theory to homotopy type theory, while reading this thesis. As far as the author is aware, there was no work yet showing that pushouts are invariant under homotopy, without adding any axioms. The most compelling and interesting task for future work would be to implement the results in a proof assistant.

## References


[1] Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146:45–55, 2009. 23

[2] Erret Bishop. *Foundations of constructive analysis*. McGraw-Hill Book Co., New York, 1967. 2

[3] Guillaume Brunerie. *On the homotopy groups of spheres in homotopy type theory*. PhD thesis, Université Nice-Sophia-Antipolis – UFR Sciences, 2016. 4

[4] Guillaume Brunerie, Kuen-Bang Hou (Favonia), Evan Cavallo, Eric Finster, Jesper Cockx, Christian Sattler, Chris Jeris, Michael Shulman, et al. Homotopy type theory in Agda. https://github.com/HoTT/HoTT-Agda. 4

[5] Evan Cavallo. Synthetic cohomology in homotopy type theory. Master's thesis, Carnegie Mellon University, 2015. 4

[6] R. Graham. Synthetic Homology in Homotopy Type Theory. *ArXiv e-prints*, June 2017. 4

[7] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In Giovanni Sambin and Jan M. Smith, editors, *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford University Press, New York, 1998. 3, 23

[8] William A. Howard. The formulae-as-types notion of construction. In J. Roger Seldin, Jonathan P.; Hindley, editor, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980. original paper manuscript from 1969. 5

[9] Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975. 2

[10] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study, 2013. 4, 32, 51

[11] Vladimir Voevodsky. A very short note on the homotopy $\lambda$-calculus. http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/Hlambda_short_current.pdf, 2006. 23

**Eidesstattliche Erklärung**

Hiermit versichere ich an Eides statt, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht zu haben und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Andreas Franz, ....., August 2017

Betreuer: Dr. Iosif Petrakis, Mathematisches Institut der Ludwig-Maximilians-Universität München.