UNIVERSITÀ DEGLI STUDI DI MILANO
Dipartimento di Matematica "Federigo Enriques"

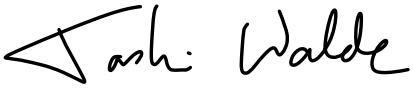
UNIVERSITÄT REGENSBURG
Fakultät für Mathematik

Master's Degree in Mathematics
ALGANT Master Program

A Type Theory for Synthetic Categories

Supervisor:
PROF. DR. DENIS-CHARLES CISINSKI

Co-Supervisor:
DR. TASHI WALDE



Candidate:
MICHELE RIVA
26903A

ALGANT Master's Thesis - Academic Year 2024/2025

Acknowledgements

The first person I wish to express my deepest gratitude to is *Denis-Charles*. I consider myself lucky to have had the chance to collaborate, as each of our meetings was, for me, a new window into the world of mathematics. Even more importantly, I could not thank enough for the passion I drew from our timeless retreats. This profoundly changed my view on mathematics, which became a passion before a duty. I look forward to keeping collaborating on the future of this project.

I would also like to express my most sincere gratitude to *Tashi*, above all for becoming such a great mathematical reference for me. It is absurd how our endless discussions became a routine in such a short time. I enjoyed every minute, and deeply appreciated the invested time and dedication. This made me realise how mathematics cannot be done alone: finding such a strong affinity and communication is extremely precious, and now I can say it exists. I am really glad we will keep in touch for the future of this work.

A heartfelt thank you also goes to *prof Vassena*. It's quite funny that, even though I've known you since I was born and finished high school long ago, I still call you that, probably to your dismay. However, deep down, you'll always be my professor, the one that made all this possible and gave me a dream to follow.

Thank you *Riki*, for being the person who I can always be myself with, and for all the love and support you constantly give me. Sometimes I realise how unbelievable it is that, after all the changes in our persons and in our lives, we have always been what we are right now since we were six. It is one of those bonds that I can bet will last forever.

To *Lorenzo*, for being the most important person in my studies from the very beginning. You were the reason why I got so involved in mathematics and how I discovered its beauty in the first place. Working together, even on another subject, shaped the way I think today. This will always remain one of my best memories in my academic path, and the reason why I became so passionate about studying.

To *Paolo*, for being by my side through all these university years since day one, literally. Thanks to you, university immediately felt like home, and as a place where I could share and cultivate my interests and passions. I could not be luckier to have you as a friend I can always count on.

To *Giovanni*, for being my guide during a dark time, both personally and mathematically. I will always be amazed by how well we worked together, despite our different approaches at the root. Yet, a big part of my passion in mathematics would not be the same without you. Working together has been both enlightening and stimulating, and I am sure it will always be.

Thank you, *Chiara*, from the bottom of my heart, for sharing the best memories of my life, which shaped the person I am today. Thank you for encouraging me to start this journey, and for making it possible in the first place. You put my desires and dreams before everything else, and I could never forget that. Now it opens the way to a new future towards my dream that wouldn't have been possible otherwise. I will keep in my heart forever where all this comes from.

If there's anything in this work harder than the mathematics, it's putting into words the gratitude towards my family. Thank you, *Mum* and *Dad*, for always being there for me throughout these years away from home, even in those moments when I was pulling away. One of the few things I know for sure is that your love and support will accompany me wherever I go. If there's someone who made this journey possible, and helped me overcome the obstacles, it's you, and I could never be grateful enough for this. Thank you, *Franci*, for being the best sister I could ask for, and for always knowing how to be there: whether I needed support or just a moment to breathe. In these years away, after forgetting what "home" even meant to me, you were a big part of how I found the answer again.

Contents

Introduction

The goal of this master thesis is to provide a formalisation of the theory of ∞-categories. This language has gained increasing traction in recent years, proving to be a suitable framework for establishing the actual foundations of modern mathematics. For this reason, the theory of ∞-categories stands in competition with classical set-theoretic foundations. In order to fully grasp the meaning of this, we will first provide an overview comparing the two approaches.

1.1 Set theory and ∞-category theory

Set theory In set theory, everything we can talk about is regarded as a set. The first fundamental relation between sets is the membership relation $\in$, stating that a set is an element of another set. The second relation we want to dig in, when we study a theory, is that of equality. In general, equality in a theory is an identification that makes two objects undistinguishable by any well-formed predicate. When it comes to understanding this in the framework of set theory, we discover that the theory of sets is grounded in two fundamental ideas which we want to despise, namely the concept of *strict equality* and the interpretation of sets as *collections*.

(1) Given any two elements $a, b \in X$, the theory allows us to formulate that $a = b$. This is a boolean relation, establishing whether two elements are equal or not. Despite its simplicity, this equality it is not coherent with the philosophy of modern mathematics, where we want to compare two objects in more general ways than a "yes or no" question. For instance, isomorphisms and homotopy equivalences are all *structured* notions of equality. This means that they come equipped with the provision of data, as we do not just care about a and b being isomorphic: instead we must provide the isomorphism map.

(2) The membership relation provides an extensional criterion for identifying sets: two sets are equal if and only if they have the exact same elements. In this sense, every set is uniquely determined by its elements. We refer to this phenomenon by saying that set theory describes every set as a "collection". However, within a collection, the only available notion of comparison is equality. This limitation implies that a theory cannot be represented as a mere collection. Indeed, a theory aims to describe much more than equality between its objects: it wants to capture the whole structure of their interactions. As a consequence, this imposes a strong restriction on the expressive power of the theory of sets: it will not be able to talk about theories, especially about set theory itself. More precisely, although every set is a collection, the theory organises them into a richer structure, which is not a collection any longer. This is due to the fact that set theory contains the whole language of functions, allowing us to compare sets without expressing an identification. In particular, the collection of all sets will not be the whole theory of sets, instead it would only retain information about equalities between them. The piece of information that is missing is, in fact, extremely important, as exhibiting a theory means to provide all the possible ways two objects interact. More concretely, the inability of set theory to describe itself reflects the fact that the theory of sets forms a category, not a set.

∞-category theory ∞-category theory aims to overcome these two limitations of the language of set theory when used to describe modern mathematics:

(1) The theory of ∞-categories serves as a homotopical foundation of mathematics. The philosophy of homotopy theory is to reject the strict notion of equality and replace it with a more structured form of identification. In this setting, any claim of equality is not taken as absolute but must be accompanied by a proof. Moreover, these data we provide can be required to satisfy higher coherences. This is a drastic change of viewpoint with respect to the classical foundations, where equality is taken as a primitive strict notion.

(2) Within an ∞-category, we can compare objects in more sophisticated ways than in set theory, by constructing morphisms between them. Consequently, ∞-categories are not just collections, but much richer structures: two ∞-categories might have the same objects but differ significantly in their morphisms. This flexibility allows ∞-categories to talk about entire theories, such as the theory of sets and the theory of groups. In particular, ∞-category theory speaks about itself: the ∞-category of ∞-categories encodes the full theory of ∞-categories.

This is a much more convenient framework for the development of modern mathematics. However, we need to face various issues coming from the nature of ∞-category theory:

(a) There exist multiple foundations for the language of ∞-categories, the so-called models. Since each model is intended to describe the same underlying theory, it becomes crucial to understand what it means to work model-independently. Indeed, the important features of ∞-category theory should be those that are invariant across all models.

(b) Working with these foundations is often highly technical. However, in practice, when doing ∞-category theory one does not make use of this language explicitly. Instead, one works with those properties and constructions that are shared by all models. This leads to some mismatch between the formal framework in which higher categories are defined and the practical applications.

(c) The used models of ∞-category theory often rely on set-theoretical foundations. This is against the homotopical spirit that ∞-category theory aims to capture. Ideally, we should avoid any instance of strict set-theoretical equality in the basic language.

These considerations motivate the need of new, unified, foundations for ∞-category theory, which do not rely on any set-theoretical model. Not only this would align better with the philosophy of homotopy theory, but it would also provide a more conceptual understanding of the theory itself. In conclusion, the goal of this project is to propose a logical theory of the so-called *synthetic* ∞*-categories* responding to these necessities. In particular, since we do not rely on the pre-existing language of ∞-categories, we will need to face both (1) and (2) in Set theory alternatively. A good framework to overcome (1), which embodies the spirit of homotopy, is that of type theory.

1.2 Fundamentals of type theory

Type theory Type theory is a branch of logic and computer science originally introduced by Russell in [Rus08], in order to solve set-theoretic paradoxes. It was later developed further by Church, with the introduction of λ-calculus, [Chu40]. A major turning point in the development of the theory came with the work of Martin-Löf, who gave rise to the so-called *dependent type theory* (see, for instance, [ML75]). Throughout this work, whenever we refer to "type theory," we will always mean this iteration. The central idea behind type theory is to provide a foundation for constructive mathematics, a framework in which asserting the truth of a statement always requires giving an explicit proof. As observed previously, this stands in deep contrast with set-theoretical foundations of mathematics, where we do not care about the provision of witnesses, and just determine whether a statement is true or false. We will soon discuss this more extensively to show how the formalism of type theory allows us to overcome the issue (1) in Set theory. Before that, however, let us give an informal overview of type theories.

Type theory considers *types* as primitive notions. The way types are introduced in the theory is by means of the language of judgements. These appear of the form $\Gamma \vdash A$ type, where Γ is called the *context*. Another form of judgement is $\Gamma \vdash a : A$, realising a as a *term* of A. Contexts and types are strictly related, in the sense that given Γ a context and $\Gamma \vdash A$ type, we can form the extended context $\Gamma, x : A$. A type over it, would then be denoted as $\Gamma, x : A \vdash B(x)$ type, and should be thought as a family of types over the terms of A, in the same way we may consider set-indexed families of sets. In this situation, we say that B is a *dependent type*

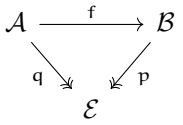
over A. In particular, we could write contexts Γ as a chain of dependent types and variables varying within them, depicted as $x_1 : A_1, \dots x_n : A(x_1, \dots, x_{n-1})$. On the top of this structure, type theory allows the formation of other two kinds of judgements, which are of the form $\Gamma \vdash A \equiv B$ type and $\Gamma \vdash a \equiv b : A$. These introduce meta-theoretical equality relations into the picture, that get the name of *judgemental equalities*. In particular, these behave strictly just as set-theoretic equalities. As motivated, we are interested in avoiding these kinds of equalities as much as possible. In conclusion, giving a type theory means to give a series of deduction rules between judgements. For instance, out of $\Gamma \vdash A$ type and $\Gamma \vdash B$ type, there will be a rule imposing the existence of a type $\Gamma \vdash A \times B$ type, and further rules will characterise its terms as the expected ones.

Homotopy and type theory In ordinary type theories, though it will not be the case for *our* theory, types can be thought as statements. In particular, a type A is a statement, and an inhabitant $a : A$ is regarded a proof of it. We will come back to the discussion of this idea and its limitations shortly. For the moment let us give an instance of this idea. Given A type and $x, y : A$, we want to formulate the statement "x is equal to y" internally to the theory. In other words, there is a type $x =_A y$ whose terms are proofs of the fact that x is equal to y. This endows the theory with an internal notion of equality, called *propositional equality*. In particular, in intensional type theories, this proof of equality will not be unique, meaning that "x is equal to y" will not just be a property of the pair (x, y). This is aligned with the spirit of homotopy: in order to identify x and y we need to provide a path between them, a homotopy. In particular, homotopies can be compared through higher homotopies and so on. In the same ways, terms of $x =_A y$ can be further compared through higher-order equalities, by taking advantage of the fact that $x =_A y$ is itself a type. This makes us understand why the language of type theory is intrinsically linked to that of homotopy. Although there is a strict judgemental equality as well, we want to despise it in favour of the so-called propositional equality. Under this perspective, type theory is a language that overcomes, by its nature, the issue (1) from Set theory.

Syntax and semantics The reader not familiar with type theory may find unsatisfying the brief overview we just gave. This is because, in order to properly present the rules and understanding them, a lot of time and patience is required. However, there are two ways of presenting a type theory, the one we mentioned being called the *syntax*. On the other hand, one could present a type theory via the *semantics*, i.e. a categorical presentation. Due to a bias of the author, and the fact that this allows a much more concise description, we will now quickly show what a type theory is from the point of view of semantics. This will help build a more concrete intuition for type theory, which we will refer back to throughout the present work.

1.3 Semantics in terms of clans and tribes

Clans as semantics of type theory The semantics of a type theory can be expressed by means of a natural model of type theory, consisting of base category $\mathcal{E}$, whose objects and arrows are regarded respectively as contexts and context substitutions, together with a representable morphism of discrete fibrations



over $\mathcal{E}$. In particular, (dependent) types are objects of the category $\mathcal{B}$ and terms are objects of the category $\mathcal{A}$. The representability condition, namely the fact f admits a right adjoint, precisely mimics the context extension phenomenon. More precisely, for every dependent type B in $\mathcal{B}$, its context is defined to be $\Gamma := p(B)$, and representability assures that there exists an extended context $\Gamma.B$ equipped with a context substitution $\Gamma.B \to B$. The arrows of $\mathcal{E}$ of these form are called display maps and enjoy various properties. First of all, one can identify a term b of the dependent type B over Γ with a section $\mathrm{id}.b$ of the display map $\Gamma.B \to \Gamma$ in $\mathcal{E}$. In this picture, it is a wise idea to study fibrations: namely those arrows that are isomorphic to a display

map. One surprising result is that, as shown by Awodey in [Awo18], the categorical properties of the fibrations in the category $\mathcal{E}$ fully capture the type-theoretical aspects we care about. In particular, a good class of fibrations in the category $\mathcal{E}$ guarantees the existence of a natural model of type theory over the category $\mathcal{E}$ compatible with the structure of fibrations. The property defining a class of fibrations is simply the stability of fibrations under pullback. This means that whenever we provide a category $\mathcal{E}$ with a class of maps which is stable under pullbacks, we in fact present a type theory. In particular, in [Joy17], Joyal studies the structure of a *clan*, which is an ordinary category equipped with a class of fibrations as mentioned, where it is further required the existence of a terminal object, with all the terminal maps being fibrations, and the closure of fibrations under composition. At a type-theoretical level, this assumption translates the existence of a terminal type and dependent sums, which we will in fact assume for our purposes.

The reader may note that such a presentation of the semantics of a type theory by means of a clan looks much simpler than that of a natural model of type theory, despite the fact that they represent the same logic. Whereas it is indeed a much more immediate approach, some clarity to be made: some concept that previously were distinct, here collapse. For instance, the language of clans considers all contexts up to dependent sums, to the point that contexts and types become undistinguishable. However, up to equivalence, that is exactly how things should work, as we will patiently observe throughout the theory.

Tribes as semantics of intensional type theory We now wish to formalise the mentioned idea that, whenever we are doing intensional type theory, we are in fact doing homotopy theory. Indeed, the way we model intensionality in a type theory represented by a clan $\mathcal{E}$, is by equipping it with a tribe structure. In particular, here every arrow admits a factorisation into an anodyne map followed by a fibration, where anodyne maps $\rightarrowtail$ are those maps with the left lifting property with respect to fibrations. In particular, factorising the diagonal map $A \to A \times A$, yields the path object of A, which we will call *identity type of* A and denote it with Id_A. This comes equipped with maps $A \rightarrowtail \mathrm{Id}_A \twoheadrightarrow A \times A$ that compose to the diagonal map. Following the formalism of model categories, identity types carry a notion of homotopy between maps. More explicitly, equality, or homotopy, of two terms $a, a' : A$ is a factorisation through the second map of $(a, a') : * \to A \times A$. To sum up, we can encode a natural model of a type theory with dependent sums, terminal type and intensional identity types inside a tribe. Within this framework, we will soon formulate homotopy type theory, which will be a starting point for this work. However, we first wish to point out a structure that types naturally have, in the current setting.

Types as groupoids It was shown by Hofmann and Streicher in [HS98] that types in an intensional type theory are naturally endowed with a groupoidal structure. Under this light, terms should be regarded as objects and equalities of terms as internal morphisms in a type. The intuition behind why the obtained structure is endowed with a groupoidal flavour is that the morphisms in a groupoids shall be invertible, i.e. a good notion of identification between objects. Unsurprisingly, by means of manipulations of the path object, equality of terms can be equipped with a composition operation (concatenation of paths) which is unital and associative, with respect to which every equality is invertible. Intuitively, this corresponds to the fact that a path from a to b could be traversed in the opposite direction. We want to remark, though, that an intensional type theory is not the theory of groupoids, as for instance we do not even have a mapping space in general. The types in an intensional type theory could have a much more sophisticated structure, e.g. the structure of categories, and here we only claim that the equalities between terms are part of a groupoidal structure. Since our goal is to construct a type theory for ∞-categories eventually, which will be a particular intensional type theory, a good intermediate step is to understand the theory of ∞-groupoids within this framework first. This will lead to the well-know homotopy type theory.

1.4 Homotopy type theory

Homotopy type theory Homotopy type theory is a special kind of intensional type theory, which describes the theory of ∞-groupoids, or of spaces. In particular, the rules of intensional type theories with dependent sums and terminal type are enriched by the existence of dependent products. It will be surprising to notice how such a simple condition will lead to a groupoid-theoretic theory.

Definition 1.4.0.1 (Homotopy type theory)**.** Let $\mathcal{E}$ be a tribe.

(1) A fibration $f : A \twoheadrightarrow B$ is *coarsely exponentiable* is the pullback functor $f^* : \mathcal{E}/B \to \mathcal{E}/A$ has a partial right adjoint over $\mathcal{E}(B)$. In particular the pullback $f^* : \mathcal{E}(B) \to \mathcal{E}(A)$ has a right adjoint $f_* : \mathcal{E}(A) \to \mathcal{E}(B)$ called *pushforward along* f.

(2) A fibration $f : A \twoheadrightarrow B$ is *exponentiable* if it is coarsely exponentiable and the pushforward $f_* : \mathcal{E}(B) \to \mathcal{E}(A)$ is a morphism of tribes, i.e. it preserves the tribe structure.

(3) $\mathcal{E}$ is a *homotopy type theory* if every fibration is exponentiable.

Syntactically, the right adjoint to the pullback functor along f is called *dependent product* along f. In particular, dependent products encode internal Homs in the theory, which are indeed an essential feature of the theory of ∞-groupoids.

Local tribe of a homotopy type theory Now, we want to point out what is one of the key features of homotopy type theory

Theorem 1.4.0.2 (Slice principle of homotopy type theory)**.** Let $\mathcal{E}$ be a homotopy type theory, and let B be a type. Then $\mathcal{E}(B)$ is a homotopy type theory, and the pullback functor along $B \twoheadrightarrow *$ preserves the type-theoretical constructions.

The fact that every local tribe of a homotopy type theory is itself a homotopy type theory translates, syntactically, into the fact that the rules governing homotopy type theory are the same in all contexts. In other words, the theory of ∞-groupoids in context Γ is totally agnostic about the context itself. This principle, found in any iteration of dependent type theory, is what really allows us to interpret types as statements.

1.5 Statements and theories

Types as statements As mentioned previously, usually in type theory we can interpret types as statements. This interpretation is valid as long as the types of our theory are, in fact, collections of proofs. Given $\vdash X$ type, we may say it is a collection, or statement, if the rules over X are the same as the empty context compatibly with pullback. Indeed, coherently with the philosophy of (2) in Set theory, two collections $\vdash X, Y$ type have the properties that knowing $\mathrm{Hom}_{\mathcal{E}}(*, X) \simeq \mathrm{Hom}_{\mathcal{E}}(*, Y)$ lifts to a homotopy equivalence of types $X \simeq Y$. This is because we can regard indifferently terms $* \to A$ of a type or maps $S \to A$, thus we may lift the equivalence above to the local tribe over X and over Y and here construct the desired maps exhibiting $X \simeq Y$. This means that, if we accept the principle in Theorem 1.4.0.2, like in ordinary dependent type theory and homotopy type theory, this forces all types to behave like collections, hence statements. In particular, they admit naive manipulations, such as for sets, to provide equivalences. Furthermore, the fact that we regard every type as a statement means that also the universe type, if there is, will in fact be a statement. As in the case of the theory of sets, this cannot represent the full theory. For instance, saying that homotopy type theory only describes statements is not completely true: there is the whole language of functors between ∞-groupoids that allows to compare them without necessarily identifying. However, when we look at the ∞-groupoid of ∞-groupoids, we realise that all this information is lost, since this behaves like a collection: the only relation between its objects that it can talk about is equality. In conclusion, homotopy type theory cleverly overcomes (1) in Set theory, however, it still shares the problem (2) with the set-theoretical foundations of mathematics. In order to fix this, for our theory to be richer than a theory of statements, we need crucial changes.

Types as theories In our theory, if we want types to describe categories, we must get rid of the statement-interpretation. This is because categories are much richer than just collections. Indeed, providing an equivalence between two categories is much more sophisticated than doing it for groupoids. This will reflect into the fact that not all types will satisfy the principle in Theorem 1.4.0.2. To support this, even in the ordinary case, slicing over an arbitrary category changes the logic of the category of categories profoundly. In particular, it creates a distinction between an object $* \to A$ and an arbitrary $S \to A$. As opposed to statements,

the types in the theory of categories could be seen, instead, as structures in which we can organise and compare statements. This is what allow types to be, in particular, theories. For instance, the theory of sets and the theory of groups are not collections: they are categories. In particular, the theory of categories can itself be represented through a type in the theory. In conclusion, in order for our theory to overcome (2) in Set theory we will need to despise the principle in Theorem 1.4.0.2.

However, we will partially recover Theorem 1.4.0.2 in our framework: there will be a class of contexts over which the rules are all the same. Ordinarily, we know that slicing the theory of categories over a groupoid keeps the same logic. This means that, among all types, we may regard animae as the collections, or statements in our theory. This philosophy will be formally expressed through Meta Yoneda Lemma for animae. In particular, this suggests that there should be a primitive notion of animae as well-behaved contexts in a type theory, which are those types over which the slicing operation satisfies Theorem 1.4.0.2. Homotopy type theory, then, is just a very special case where every type is an anima, hence a statement. However, we do not have a statement interpretation of types a priori, in absence of the mentioned assumption. The new key feature of our theory will exactly be the existence of types that do not admit this interpretation.

1.6 Towards a type theory of categories

In this work, our goal is to construct a type theory whose types deserve to be called higher categories. A first attempt in this direction comes from Riehl and Shulman, in [RS17]. In particular, they developed the formalism of simplicial type theory, which is a framework suitable for the formalisation of some concepts from higher category theory. However, it is unable to talk about objectwise statements and point out the fundamental role animae have in the theory, under the vests of contexts. These are essential features for a theory of ∞-categories. Furthermore, we believe that the reversal of perspective in Types as theories really embodies the conceptual difference between a theory of categories and a theory of groupoids, and we wish our theory to reflect it when compared with homotopy type theory. For this purpose, we will propose a new formalism with widely different features.

The project In the spirit of the natural models of homotopy type theory introduced by Awodey ([Awo18]), we may present a type theory in a categorical way, by exhibiting a tribe. Cisinski determined axioms to be imposed on a tribe for it to be a semantic interpretation of a theory of synthetic categories, as opposed to homotopy type theory which describes higher groupoids. However, there is an unsatisfying aspects in it: there should not exist an ordinary category of ∞-category, whose arrows are functors of ∞-categories. In other words, these should not have a strictly associative composition as in an ordinary category. In this project, we will generalise this work by dispensing the strict structure of an ordinary type theory, in which we have a concept of strict equality. This cannot be done in the semantics, as its starting point is an ordinary category. For this goal, we need to present the syntax of the theory, and having care of replacing the judgemental, strict, equalities with the internal intensional equalities. This makes the theory agnostic about the concept of equality, and could be describing both a strict or a higher structure. Presenting the syntax has a further advantage, that is, the ability to implement the theory in a computer, which may be one of the final goals of this project. However, we will also need to carefully translate all the axioms in the semantics into well-formed type-theoretical rules. This will require particular attention. The resulting structure will be a much richer syntax that can talk about theories, in the sense of Types as theories, in addition to statements. In particular, the theory will be introduced with an intrinsic distinction between types and statements, which will play a role at every step. This is, in essence, the difference between the theory of categories and the theory of groupoids.

A Type Theory beyond Animae

2.1 Strategy

Let us sum up the key points we will obey to in the development of the theory.

(a) We will try to make our theory as agnostic as possible about the concept of equality. Therefore, we will replace every instance of judgemental equality with a propositional equality. This forces us to formulate the present theory syntactically. This corresponds to overcoming (1) in Set theory.

(b) As motivated in Types as theories, we want our theory not to describe only statements. What we will regard as statements will be a primitive concept of "anima". One of the main goals of this chapter is to make this notion of anima fit into the rules of type theory. We will not specify what animae are to begin with, we will just make sure that they are a class of well-behaved contexts over which all rules hold. In particular, this will imply that naive manipulations of types, as collections, from ordinary type theory will be allowed for animae only. This is one of the most crucial aspects of the theory, and corresponds to overcoming (2) in Set theory.

(c) The first rules of the theory will actually be introduced in all contexts. In particular, the main differences in the first part of the present chapter will lie in the philosophy and in the unstrictification procedure. However, things will change drastically when we will need to introduce a type of functions and a dependent product in our setting. These will be instances of types that cannot be formed in general, but they can always be formed over animae. To support this, the reader may note that even in ordinary category theory, the slice category $\mathrm{CAT}_{/\Gamma}$ has no internal Hom for Γ arbitrary. However, because it will have internal Homs in absolute context, it will have an internal Hom at least over all animae following (b).

The goal of the present chapter will be to construct a type theory about entities that are not animae, i.e. statements, but more general types. Types could be anything, they will be ways of organising statements into an internal structures. This will be a good substrate to then characterise types as categories (in context) in the following chapter.

2.2 Judgements

Let us introduce the types of judgements that are allowed in our syntax. Every judgement will be of one of the following forms:

(1) $\vdash \Gamma$ anima, read as "Γ *is an anima*".

(2) $\vdash \Gamma$ ctx, read as "Γ *is a context*".

(3) $\Gamma \vdash A$ type, read as "A *is a type in context* Γ".

(4) $\Gamma \vdash a : A$, read as "a *is a term of* A *in context* Γ".

(5) $\vdash \gamma : \Delta \to \Gamma$ ctx, read as "γ *is a context substitution from* Δ *to* Γ".

(6) $\Gamma \vdash A \equiv B$ type, read as "A *and* B *are judgementally equal types in context* Γ".

(7) $\Gamma \vdash a \equiv a' : A$, read as "$a$ *and* a' *are judgementally equal terms of* A *in context* Γ".

(8) $\vdash \gamma \equiv \delta : \Delta \to \Gamma$ ctx, read as "γ *and* δ *are judgementally equal context substitutions from* Δ *to* Γ".

The first five take the name of *basic judgements* whereas the last three are called *equality judgements*.

As in classical dependent type theory, a judgement of the form $\Gamma \vdash A$ type has, as a premise, the judgement $\vdash \Gamma$ ctx. In the same way, a judgement of the form $\Gamma \vdash a : A$ has, as premises, the judgements $\vdash \Gamma$ ctx and $\Gamma \vdash A$ type. Finally, a judgement of the form $\vdash \gamma : \Delta \to \Gamma$ ctx has premises $\vdash \Delta, \Gamma$ ctx. In the same way, the equality judgements talk respectively about types, terms of a type and substitutions, and thus need obvious premises. Note that judgemental equality behaves strictly as a truth value, thus we will try to avoid making use of it as much as possible, in order to have a good description of higher structures.

The form of judgement in (1) will be the first new feature of our syntax, namely the introduction of animae. Despite the fact that we claim the existence of entities of five different natures, we should regard $\vdash \Gamma$ anima as a predicate on contexts. In other words, the first rule of this theory is

$$\frac{\vdash \Gamma \text{ anima}}{\vdash \Gamma \text{ ctx}}.$$

This realises animae as a subclass of contexts. In particular, given $\vdash \Gamma$ anima we can consider types $\Gamma \vdash A$ type over it. The way this should be regarded is that we are looking at the theory of categories over an anima context Γ. As mentioned previously, it will be our responsibility to make sure that the theory of categories over an anima has the same rules of the theory in empty context. In some sense, the theory is agnostic about the context which we work over, as long as it is an anima. The first big difference with the theory of higher groupoids is that in homotopy type theory, the rules are stated in every arbitrary context. In other words, homotopy type theory will be a degenerate case of the first part of this agnostic syntax in the case where every context is an anima.

Remark 2.2.0.1. Since we want to regard the types in our theory as categories, one could legitimately think of introducing the predicate of being an anima, or groupoid, on types, and not on contexts. Instead, we introduce animae in our theory purely to characterise contexts over which we can do category theory. This happens before we can even say what it means for a type to be a groupoid. In particular, we prefer to distinguish the notions of animae and groupoids in principle. The former ones are some well-behaved contexts given a priori, whereas the latter ones will be defined as those types with all invertible maps. We will then formalise how we can regard animae as groupoids. In particular, because groupoids will be defined through a checkable property, we will determine how to *prove* that a context is, in fact, an anima.

2.3 The calculus of substitutions

First of all, in the development of this agnostic syntax, we need a calculus of substitutions. In order to be more formal and to be closer to the semantics, we prefer to follow a more categorical notation for substitutions, which treats them as abstract arrows inducing an action on types and their terms. Every rule involving the calculus of context substitutions will be strict. This is what makes the calculus equivalent to variables manipulation, which will be a more intuitive formalism that we will soon meet.

Remark 2.3.0.1. As anticipated, we wish our theory to be as agnostic as possible. For instance, we will have maps whose composition is not strictly associative, but only up to some *homotopy*. However, in order to have a notion of homotopy, we will need to introduce an intensional identity type, which we can only formulate once we have a good calculus of substitutions. In particular, substitution will be determined as a strict process.

2.3.1 Formation of contexts

The first two rules express the formation of basic contexts. The first is a terminal context, which happens to be an anima, whereas the second is the extension of a context along a type over it.

$$\frac{}{\vdash * \text{ anima}} \qquad \frac{\vdash \Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type}}{\vdash \Gamma.A \text{ ctx}}.$$

Note that we automatically deduce $\vdash * \,\mathrm{ctx}$ and that we can extend any $\vdash \Gamma$ anima with a $\Gamma \vdash A$ type. As a convention, every construction on the top of two judgementally equal types (or terms) will give rise to judgementally equal outcomes. For instance, if $\Gamma \vdash A \equiv A' \,\mathrm{type}$, we will identify the context $\Gamma.A$ with $\Gamma.A'$ despite them not being syntactically equal. We further impose the terminality of $*$ by means of the following rules:

$$\frac{\vdash \Gamma \,\mathrm{ctx}}{\vdash ! : \Gamma \to *} \qquad \frac{\vdash \gamma : \Gamma \to *}{\vdash \gamma \equiv ! : \Gamma \to * \,\mathrm{ctx}}.$$

By convention, we will write $\vdash A$ type exactly to mean $* \vdash A$ type, and in that case we denote with $\vdash A \,\mathrm{ctx}$ the extension $\vdash *.A \,\mathrm{ctx}$. We will refer to $*$ as the *empty* or *absolute context*, and to types over it as *absolute types*.

2.3.2 The ordinary category of contexts

We want to be able to perform basic operations with context substitutions. In particular, we want an identity substitution and a composition operation, expressed by

$$\frac{\vdash \Gamma \,\mathrm{ctx}}{\vdash \mathrm{id} : \Gamma \to \Gamma \,\mathrm{ctx}} \qquad \frac{\vdash \gamma_1 : \Gamma_2 \to \Gamma_1 \,\mathrm{ctx} \quad \vdash \gamma_0 : \Gamma_1 \to \Gamma_0 \,\mathrm{ctx}}{\vdash \gamma_0 \circ \gamma_1 : \Gamma_2 \to \Gamma_0 \,\mathrm{ctx}}.$$

Furthermore, we want composition to be (strictly) unital and associative:

$$\frac{\vdash \gamma : \Delta \to \Gamma \,\mathrm{ctx}}{\vdash \mathrm{id} \circ \gamma \equiv \gamma \equiv \gamma \circ \mathrm{id} : \Delta \to \Gamma \,\mathrm{ctx}} \qquad \frac{\vdash \gamma_2 : \Gamma_3 \to \Gamma_2 \,\mathrm{ctx} \quad \vdash \gamma_1 : \Gamma_2 \to \Gamma_1 \,\mathrm{ctx} \quad \vdash \gamma_0 : \Gamma_1 \to \Gamma_0 \,\mathrm{ctx}}{\vdash \gamma_2 \circ (\gamma_1 \circ \gamma_0) \equiv (\gamma_2 \circ \gamma_1) \circ \gamma_0 : \Gamma_3 \to \Gamma_0 \,\mathrm{ctx}}.$$

We can summarise these rules as follows: contexts and context substitutions fit into an ordinary category with terminal object the empty context. This happens in ordinary dependent type theory as well. Although this seems to be against our purpose to detect non-strict behaviours, we will instead focus on studying *functors* of types, that will be defined later on. These will be related to substitutions only up to equality terms, introducing non-strictness in the theory.

Remark 2.3.2.1. Let Ctx denote the ordinary category of contexts and context substitutions. Despite the fact that Ctx will not be a tribe any longer, since intensional identity types and dependent sums will be introduced non-strictly, it plays the same role as the tribe of an ordinary intensional type theory. Conceptually, we can think to have some sort of localisation functor $\mathrm{Ctx} \to \mathbb{L}(\mathrm{Ctx})$, where $\mathbb{L}(\mathrm{Ctx})$ will be a non-strict structure consisting of types and functors of types, which is the place where we will develop the theory of synthetic categories. If we already assumed the language of ∞-categories, $\mathbb{L}(\mathrm{Ctx})$ should be an ∞-category with pullbacks. Though, since we do not assume it, we will impose syntactic rules determining a non-strict structure for $\mathbb{L}(\mathrm{Ctx})$, so that it could be, in particular, an ∞-category. The goal is to provide a rich enough structure in order to perform the constructions of interest in $\mathbb{L}(\mathrm{Ctx})$, so that its objects deserve to be called ∞-categories. However, some basic operations related to the calculus of substitutions can be performed in the ordinary category Ctx.

2.3.3 Base change along a substitution

In the usual way of introducing dependent type theory, substitutions are not introduced as an abstract entity. Instead, they are encoded in the use of explicit variables, which can be substituted with expressions inside type and term dependences. In our setting, we recover this by stating the existence of an action of substitutions on types and terms as follows:

$$\frac{\vdash \gamma : \Delta \to \Gamma \,\mathrm{ctx} \quad \Gamma \vdash A \,\mathrm{type}}{\Delta \vdash A[\gamma] \,\mathrm{type}} \qquad \frac{\vdash \gamma : \Delta \to \Gamma \,\mathrm{ctx} \quad \Gamma \vdash A \,\mathrm{type} \quad \Gamma \vdash a : A}{\Delta \vdash a[\gamma] : A[\gamma]}.$$

We refer to $A[\gamma]$ and $a[\gamma]$ as the *base change along* γ of A and a respectively. Since we want substitutions to be a strict process, we further ask base change to be judgementally compatible with composition of

substitutions and identity:

$$\frac{\Gamma \vdash A\ \mathrm{type}}{\Gamma \vdash A[\mathrm{id}] \equiv A\ \mathrm{type}} \qquad \frac{\Gamma \vdash A\ \mathrm{type} \quad \Gamma \vdash a : A}{\Gamma \vdash a[\mathrm{id}] \equiv a : A}$$

$$\frac{\vdash \gamma_1 : \Gamma_2 \to \Gamma_1\ \mathrm{ctx} \quad \vdash \gamma_0 : \Gamma_1 \to \Gamma_0\ \mathrm{ctx} \quad \Gamma_0 \vdash A\ \mathrm{type}}{\Gamma_2 \vdash A[\gamma_0 \circ \gamma_1] \equiv A[\gamma_0][\gamma_1]\ \mathrm{type}}$$

$$\frac{\vdash \gamma_1 : \Gamma_2 \to \Gamma_1\ \mathrm{ctx} \quad \vdash \gamma_0 : \Gamma_1 \to \Gamma_0\ \mathrm{ctx} \quad \Gamma_0 \vdash A\ \mathrm{type} \quad \Gamma_0 \vdash a : A}{\Gamma_2 \vdash a[\gamma_0 \circ \gamma_1] \equiv a[\gamma_0][\gamma_1] : A[\gamma_0 \circ \gamma_1]}.$$

Again, these relations are implicit in the use of the notation with variables, as the types (or terms) which we claim to be judgementally equal would be also be written in the same way.

2.3.4 Weakening

We introduce the most important kind of context substitution, which is what semantically is called display map: the weakening.

$$\frac{\Gamma \vdash A\ \mathrm{type}}{p : \Gamma.A \to \Gamma\ \mathrm{ctx}}.$$

In a tribe, among all arrows, these substitutions are encoded by the notion of fibrations. Intuitively, this rule says that given two types $\Gamma \vdash A, B\ \mathrm{type}$ in the same context, we can regard B as a type over the context extension by forming $\Gamma.A \vdash B[p]$, which is called the *weakening of* B *at* A. The idea is that we are exhibiting $B[p]$ as a "free lift" of B to the extended context $\Gamma.A$, with a trivial type dependence over A. Semantically, $B[p]$ is just the projection fibration $B \times_\Gamma A \twoheadrightarrow A$ over Γ.

Before making more formal this intuition of $B[p]$ in our setting, we want to claim the existence of a *universal term*, a term of the weakening of a type at itself. More precisely, when we have a type A in a context Γ, we can weaken A at itself, and we want to equip the resulting type with a universal term

$$\frac{\Gamma \vdash A\ \mathrm{type}}{\Gamma.A \vdash q : A[p].}$$

Its universality property will be determined by means of the next rules we will see. The intuition behind the universal term is that we can "freely lift" the terms of A to the extended context $\Gamma.A$. Unsurprisingly, semantically, the universal term corresponds to the diagonal map $\Delta : A \to A \times_\Gamma A$, section of the projection $A \times_\Gamma A \twoheadrightarrow A$. In the semantics, the diagonal map is called universal since its pullback along a term $a : *_\Gamma \to A$ is the term $a : *_\Gamma \to A$ itself. In light of this we will formulate the next rules, which will conclude the calculus of substitutions.

2.3.5 Substitutions extensions

We conclude the calculus of substitutions by putting in relation substitutions and terms. This will formalise the intuition of a substitution as an expression of variables. In particular, we will be able to characterise terms of types via substitutions, namely the sections of the fibrations. As a consequence this will justify the interpretation of the weakening $B[p]$ at A as the type consisting of pointwise assignments from terms of A to terms of B. Under this light, the universal term of A will represent the identical assignment. More precisely, the idea is that we can describe context substitutions $\Gamma.A \to \Gamma.B$ compatible with the weakening projections exactly as the terms of the weakened type $\Gamma.A \vdash B[p]\ \mathrm{type}$. This will be our definition of function of types.

The first rule tells us that we can construct new substitutions by extending one with a term as follows:

$$\frac{\vdash \gamma : \Delta \to \Gamma\ \mathrm{ctx} \quad \Gamma \vdash A\ \mathrm{type} \quad \Delta \vdash a : A[\gamma]}{\vdash \gamma.a : \Delta \to \Gamma.A\ \mathrm{ctx}}$$

The next two rules, conversely, allow us to recover γ and a from $\gamma.a$. Intuitively, we can regard $\Gamma.A$ as

some sort of product, and we want to equip it with two "projections":

$$\frac{\vdash \gamma : \Delta \to \Gamma \,\mathrm{ctx} \quad \Gamma \vdash A \,\mathrm{type} \quad \Delta \vdash a : A[\gamma]}{\vdash p \circ (\gamma.a) \equiv \gamma : \Delta \to \Gamma.A \,\mathrm{ctx}} \qquad \frac{\gamma : \Delta \to \Gamma \,\mathrm{ctx} \quad \Gamma \vdash A \,\mathrm{type} \quad \Delta \vdash a : A[\gamma]}{\Delta \vdash q[\gamma.a] \equiv a : A}.$$

Thanks to these rules, the substitution p is called *first projection* and the universal term q is called *second projection* in the language of categories with families, introduced in [CCD21]. The final rule of the calculus of substitutions says that every context substitution into an extended context is of the above form

$$\frac{\Gamma \vdash A \,\mathrm{type} \quad \vdash \gamma : \Delta \to \Gamma.A \,\mathrm{ctx}}{\vdash (p \circ \gamma).q[\gamma] \equiv \gamma : \Delta \to \Gamma.A \,\mathrm{ctx}}.$$

Given $\vdash \Gamma \,\mathrm{ctx}$ and $\Gamma \vdash A \,\mathrm{type}$, we have that the terms of A are in correspondence with substitutions of the form $\mathrm{id}\,.a : \Gamma \to \Gamma.A$. Semantically, this corresponds to the fact that we can regard terms of dependent types as sections of the fibrations (the display maps $p : \Gamma.A \to \Gamma$) as previously explained.

As anticipated, thanks to the rules for substitutions extensions, we can finally understand what we mean by universality when we talk about the universal term.

Remark 2.3.5.1. Let $\Gamma \vdash A \,\mathrm{type}$. For any $\Gamma \vdash a : A$, we have $\Gamma \vdash q[\mathrm{id}\,.a] \equiv a : A$. In other words, the base change of the universal term at the substitution induced by a is a itself. We call this property *universality*.

Finally, one can show computation tools for substitutions extensions, but we will not steal space and time to more interesting discussions. We will limit to point out the following tool, that would reveal useful if the reader checked that all the rules we will see are well-typed.

Remark 2.3.5.2. Let $\vdash \delta : \Xi \to \Delta \,\mathrm{ctx}$ and $\vdash \gamma : \Delta \to \Gamma \,\mathrm{ctx}$, and assume $\Delta \vdash a : A$. Then we have an equality of substitutions $\vdash \gamma.a \circ \delta \equiv (\gamma \circ \delta).a[\delta] : \Xi \to \Gamma.A \,\mathrm{ctx}$. This easily follows from the fact that equality of substitutions can be checked after the two projections.

Remark 2.3.5.3 (Invariance under context substitution)**.** Now that we fully developed the calculus of substitutions, we will construct the rest of the syntax by obeying to the principle that every construction shall be compatible with base change. This means that, given a rule where everything lives on the top of a context Γ, we will ask systematically that, for any substitution $\gamma : \Delta \to \Gamma$, the base change of the outcome along γ coincides with the outcome of the rule applied to the base changes of the premises along γ. To be more precise, in this context the base change along γ of a $\Gamma.A \vdash B \,\mathrm{type}$ means its base change $\Delta.A[\gamma] \vdash B[\gamma.A]$, and analogously for terms and iterated type dependences.

2.3.6 Functions and dependent functions

Before providing the next rules of the syntax, let us introduce some names and interpretations in the setting we have already developed. In particular, we will define the notion of (dependent) function of types and their evaluations. In order for this discussion to be meaningful, we will first introduce the concept of fibre of a dependent type at a term. This allows us to interpret a dependent type as a family of types.

Definition 2.3.6.1 (Fibre of a type)**.** Let $\Gamma \vdash A \,\mathrm{type}$ and $\Gamma.A \vdash B \,\mathrm{type}$ and consider a term $\Gamma \vdash a : A$. We define the *fibre of* B *at* a as $\Gamma \vdash B(a) :\equiv B[\mathrm{id}\,.a] \,\mathrm{type}$.

Note that given $\vdash \Gamma \,\mathrm{ctx}$ and $\Gamma \vdash A, B \,\mathrm{type}$, the fibre of the weakening at any term $\Gamma \vdash a : A$ is $\Gamma \vdash B[p][\mathrm{id}\,.a] \equiv B$. Under this light, the weakening can be regarded as the constant family at B over the terms of A. Semantically, this corresponds to the fact that pulling back $B \times_\Gamma A \twoheadrightarrow A$ along the point $*_\Gamma$ returns B. This construction takes place more in general by considering *generalised terms* instead of simple terms.

Remark 2.3.6.2. Let $\Gamma \vdash A \,\mathrm{type}$. Given $\Gamma \vdash S \,\mathrm{type}$, a *generalised term of* A *in context* S is a term $\Gamma.S \vdash a : A[p]$ of the weakening of A at S. In particular, for $\Gamma.A \vdash B \,\mathrm{type}$, the *generalised fibre of* B *at* a is the type $\Gamma.S \vdash B[p.a] \,\mathrm{type}$.

Thanks to the terminal type that we will be able to see terms as special cases of generalised terms. In particular, we will see that functorial statements will be those assignments that see no difference between

absolute and arbitrary generalised terms. Now that we saw how a dependent type can be interpreted as a family of types, namely its (generalised) fibres, we are ready to discuss the notion of dependent function.

Definition 2.3.6.3 (Dependent functions and evaluation)**.** Let $\Gamma \vdash A$ type and let $\Gamma.A \vdash B$ type. A *dependent function from* A *to* B is a term $\Gamma.A \vdash f : B$.

(1) For every $\Gamma \vdash a : A$, we define the *evaluation of* f *at* a as the term $\Gamma \vdash f(a) :\equiv f[\mathrm{id}\,.a] : B(a)$.

(2) For every $\Gamma \vdash S$ type and generalised term $\Gamma.S \vdash a : A[p]$ we define the *evaluation of* f *at* a as the generalised term $\Gamma.S \vdash f(a) :\equiv f[p.a] : B[p.a]$.

The above definition allows us to reinterpret terms of dependent types as assignments to any term of A of a term of the fibre of B at it. This is the reason why we call them dependent functions, as they are assignments where the target is allowed to vary over the terms of the source. In other words, the codomain is a family of types, the fibres, and the evaluation at every term will lie in the corresponding fibre. A case of particular interest is when B has a trivial type dependence. In other words, B is of the form $C[p]$ type for some $\Gamma \vdash C$ type, so that its fibres are constantly C. Therefore, a term $\Gamma.A \vdash f : C[p]$ provides an assignment to each term $a : A$ of a term $f(a) : C$. Unsurprisingly, we define:

Definition 2.3.6.4 (Functions)**.** Let $\Gamma \vdash A, C$ type. A *function of types from* A *to* C is a term $\Gamma.A \vdash f : C[p]$ type.

In particular, given $\Gamma \vdash A, C$ type and a function f from A to C, we inherit a notion of evaluation at all terms and generalised terms:

(1) For every $\Gamma \vdash a : A$, the *evaluation of* f *at* a is the term $\Gamma \vdash f(a) :\equiv f[\mathrm{id}\,.a] : C$.

(2) For every $\Gamma \vdash S$ type and generalised term $\Gamma.S \vdash a : A[p]$, the *evaluation of* f *at* a is the generalised term $\Gamma.S \vdash f(a) :\equiv f[p.a] : C[p]$.

As the reader might expect, functions of types will have a key importance throughout the theory, as they are the first tool we have to compare types over the same contexts. Indeed, they will be close to what we will define to be a functor of types later on.

Remark 2.3.6.5. Let $\Gamma \vdash A, C$ type. By the rules about extension of substitutions, functions of types from A to C are in correspondence with substitutions $\Gamma.A \to \Gamma.C$ which commute with the first projections. In particular, $\Gamma.A \vdash f : C[p]$ corresponds to $p.f : \Gamma.A \to \Gamma.C$.

As substitutions fit into an ordinary category, functions of types inherit a rich structure as well. In particular, because generalised terms are practically the same as dependent functions, we can interpret the evaluation operation of a function at a generalised term as a composition operation. Indeed we introduce the following notation for evaluations of functions at generalised terms.

Definition 2.3.6.6 (Composition of functions)**.** Let $\Gamma \vdash A, B, C$ type and consider two functions $\Gamma.A \vdash f : B[p]$ and $\Gamma.B \vdash g : C[p]$. We define the *composition of* f *and* g as the function $\Gamma.A \vdash g \circ f :\equiv g(f) \equiv g[p.f] : C[p]$.

As pointed out, $g \circ f$ can be equivalently referred to as $g(f)$, when f is regarded as a generalised term. Type theoretically, there is no distinction between the two interpretations, but conceptually, we might prioritise to think of f either as a generalised term or as a function. Because we defined a composition operation on functions, we are interested in its unitality and associativity properties properties.

Remark 2.3.6.7. As the reader might expect, composition of functions resembles composition of context substitutions, in the sense that $p.(g \circ f) = p.g \circ p.f$. Thus, it mirrors the behaviour of composition of context substitutions, which fit into an ordinary category by Remark 2.3.2.1:

(1) Composition of functions resembles composition of context substitutions is strictly unital with respect to the universal term $\Gamma.A \vdash q : A[p]$, which is thus named the *identity function of* A.

(2) Composition of functions of types is strictly associative because composition of substitutions is.

A key step we are missing, in order to regard functions of types as functors of ∞-categories, is that we want to replace these judgemental unitality and associativity constraints with relations that are true up to some equality data. This is, in some sense, the essence of ∞-category theory.

Convention. Let $\vdash \Gamma\,\mathrm{ctx}$ and let $\Gamma \vdash A, B$ type with a function $f : A \to B$. Given $\Gamma.B \vdash E$ type, we may denote with $\Gamma.A \vdash f^*(E) :\equiv E[p.f]$ type the base change of E along f, resembling a pullback notation. This emphasises the idea, for the categorically-oriented reader, that base change along a function of types is a pullback construction. This will be formalised later type-theoretically.

2.3.7 Variables

The rules we introduced for the calculus of substitutions allow us to represent contexts, types and terms via manipulation of variables. This can be convenient when the contexts get more complicated, and in order to have a better intuition of the type dependences. In particular, we may denote context extensions $\Gamma.A$ by $\Gamma, x : A$, dependent types $\Gamma.A \vdash B$ type by $\Gamma, x : A \vdash B(x)$ type emphasising their interpretation as family of fibres, and dependent terms $\Gamma.A \vdash f : B$ by $\Gamma, x : A \vdash f(x) : B(x)$ emphasising their interpretation as dependent functions. In this picture, the weakening of a type $\Gamma \vdash C$ type at A is denoted as the constant family $\Gamma, x : A \vdash B$ type, in line with the interpretation of weakening as family of types with constant fibres. From this viewpoint, the universal term of A is denoted as the identical assignment $\Gamma, x : A \vdash x : A$. Here, the interpretation of this special term is much more concrete: the universal term is the lift of the free variable. In particular, it is clear why it embodies the identical function, with respect to which composition is unital by Remark 2.3.6.7. Finally, a context extension of the form $\Gamma.A.B$ will be denoted as $\Gamma, x : A, y : B(x)$. In general, we can translate one notation into the other with no loss of generality. Therefore, it is important to understand how to go in the converse direction as well: from manipulations of variables to the formalism we have been using up to now. For instance, a key ingredient is that we wish to have access to all variables appearing in the context, in order to express judgements such as $\Gamma, x : A, y : B(x) \vdash x : A$ in the formalism we set up. The way this is done is the following: given a context $\Gamma.A.B$, the variable of B is recovered in the conclusions by the universal term $\Gamma.A.B \vdash q : B[p]$, and the one of A by $\Gamma.A.B \vdash q[p] : A[p^2]$. This can be iterated for longer chains. This provides a recipe to describe the following two common procedures:

(1) Reordering of a context: for instance, when $\Gamma \vdash A, B$ type, there is no true difference between the contexts $\Gamma, x : A, y : B$ and $\Gamma, y : B, x : A$. This is the context substitution $\mathrm{id}\,.q.q[p]$, that sends a dependent type $\Gamma, x : A, y : B \vdash C(x, y)$ type into the swapped type $\Gamma, y : B, x : A \vdash C(x, y)$ type

(2) Manipulation of variables in a dependent type: for instance, given a dependent type $\Gamma, x : A, y : A \vdash C(x, y)$ type, we can describe the diagonal parametrisation $\Gamma, x : A \vdash C(x, x)$ type. This is realised by the substitution $\mathrm{id}\,.q[p] : \Gamma.A \to \Gamma.A.A[p]$.

In these cases, and more in general for manipulation of rules, we prioritise the notation with variables, being it more concrete. For the sake of coherence, though, the axioms will always be introduced in the formalism we used up to here. Sometimes, it will allow us to be more precise with less ambiguous notation.

Remark 2.3.7.1 (Base change as reparametrisation)**.** Let $\Gamma \vdash A, B$ type and consider a function $\Gamma, x : A \vdash f(x) : B$. For every dependent type $\Gamma, x : A \vdash E(x)$ type, the base change $f^*(E)$ is the type $\Gamma, x : A \vdash f^*(E)(x) \equiv E(f(x))$ type, i.e. it is literally the reparametrisation of the type family E along f.

2.3.8 Functorial statements

We end this discussion by pointing out the key idea of functoriality, that is intrinsic in judgements.

Remark 2.3.8.1 (Functoriality of judgements)**.** The variables interpretation is indeed useful, but might be misunderstood. Indeed, it seems like a dependent type $\Gamma, x : A \vdash B(x)$ type is fully determined by the fibres $\Gamma \vdash B(a)$ type over the terms $\Gamma \vdash a : A$, and the same for their terms. This is *not* the way this notation should be interpreted, and the reason lies in the fact that variables in the context $\Gamma, x : A$ do not just stand for arbitrary terms $\Gamma \vdash a : A$, but instead for arbitrary generalised terms of A. The reason why we claim it, is that every dependent type $\Gamma, x : A \vdash B(x)$ type stands for the collection of all its generalised fibres. Indeed, from the family of $\Gamma.S \vdash B[p.a]$ with $\Gamma.S \vdash a : A[p]$ we recover B itself, in the case $S \equiv A$ and $a \equiv q$, and conversely all the fibres are obviously constructed from B. Moreover, we saw that a dependent term $\Gamma.A \vdash f : B$ does not only provide an assignment on terms to $a : A$ of $f(a) : B(a)$, but it further provides an assignment on generalised terms. In particular, when B is of the form $C[p]$ for some $\Gamma \vdash C$ type, f is a function of types from A to C, and it induces an assignment to any function from S to A of a function from

S to C. We refer to this phenomenon as *functoriality*. The reason will be clearer later, when types will be regarded as synthetic categories, but intuitively it means that f is not just an assignment on objects, but also on arrows and on all classes functors from a fixed category.

The reader familiar with ordinary type theory might be confused here. Indeed, in homotopy type theory, this subtlety does not subsist: although the distinction between the judgement $\Gamma, x : A \vdash B(x)$ type and the family of fibres $\Gamma \vdash B(a)$ type over the terms $\Gamma \vdash a : A$ exists in principle, it is solved by the rules being the same in all contexts, as we will soon observe. This is a consequence of the interpretation of Types as statements. In our setting, this distinction is far from being solved, and it is the essence of the distinction between an objectwise statement and a functorial statements. The ability to distinguish functorial statements from objectwise ones, core idea even in ordinary category theory, is a feature of the presented theory, that does not even belong to simplicial type theory [RS17]. For the moment we will focus our attention on functorial statements, and later to the case of homotopy type theory.

Remark 2.3.8.2 (Functorial statements)**.** We can take Remark 2.3.8.1 further: a judgement $\Gamma, x : A \vdash f(x) : B(x)$ is literally *the same* as a rule

$$\frac{\Gamma \vdash S \text{ type} \quad \Gamma.S \vdash a : A[p]}{\Gamma.S \vdash f(a) : B[p.a]}$$

compatible with base change over Γ, meta-theoretically. Indeed, for $S \equiv A$ and $a \equiv q$ the universal term, the rule provides a judgement $\Gamma.A \vdash f(q) : B$, that is $\Gamma, x : A \vdash f(x) : B(a)$, and the converse is trivial. Thus, we will talk about *functorial statements* to mean a judgement or a rule of the form above. We refer to these two forms as the *internal formulation* and the *meta-theoretical formulation* of the functorial statement, respectively. The internal formulation will be preferred for its elegance and flexibility. However, the reason why we also care about meta-theoretical formulation is that, sometimes, we will have a *partial* functoriality. For instance, a rule as above might hold only for some special $\Gamma \vdash S$ type, such as for $\vdash \Gamma.S$ anima, and thus cannot (yet) be assembled to a judgement because it is not fully functorial. One of the purposes of the groupoid core type constructor will be to internalise these statements, which we will call *objectwise statements*.

Now, as anticipated, let us focus on the case of homotopy type theory. We will understand how this solves the subtleties we presented, and we will look into the consequences of this difference between homotopy type theory and our theory. In particular, this will affect the way we will formulate statements in the present theory: two formulations might coincide in homotopy type theory and differ in our setting.

Remark 2.3.8.3 (A profound difference with homotopy type theory)**.** In homotopy type theory, all rules are the same in all contexts and they are preserved by base change. In other words, the meta-theoretical counterpart of a judgement $\Gamma, x : A \vdash f(x) : B(x)$ could be written as

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash f(a) : B[\mathrm{id}\,.a]},$$

because its invariance under weakening would yield the one seen in Remark 2.3.8.2. Because in our theory of categories this principle will not be true anymore, this game will not work any longer. For this reason, in homotopy type theory, some definitions can be given meta-theoretically in a naive way, as above. In our setting, though, the judgemental form will be strictly stronger than the naive one above. Thus, if we want to define concepts that satisfy a functorial conditions in our setting, we need to write them in the form of judgements, or meta-theoretically including all possible context extensions as in Remark 2.3.8.2. As the latter is less natural and we prefer to manipulate non-meta theoretical concepts, we will prioritise the judgemental form. We will see instances of this phenomenon, where our definition will be formulated judgementally as opposed to [BGL+17]: in their setting everything is automatically functorial.

On the other hand, in our setting, when Γ is an anima, the naive formulation above will in fact be equivalent to a so-called *objectwise statement*. We will define this concept and prove it is equivalent to the naive meta-theoretical formulation above later on. What we care about now, though, is that this formulation will be used to express only *some* statements: the ones about "objectwise" conditions. As one could expect from the language of category theory, these will represent the objectwise phenomena in ordinary category theory, as opposed to the functorial ones. In particular, they will have an essential role throughout the theory.

2.3.9 Compatibility with base change

We saw that the process of base change allows to change of context every type and term appearing in a rule. However, we would like to say, as a principle of the theory, that the rules are compatible with base change. In order for this to make sense, we need to specify what we mean to base change a rule. The idea is that every rule will have a fixed base context Γ such that every type or term will be introduced over it or over a context extension of it. The first step, then, will be to understand how to extend a substitution via a type, so that it makes sense to base change $\Gamma.A \vdash B$ type along some $\vdash \gamma : \Delta \to \Gamma$.

Remark 2.3.9.1 (Extension of substitutions with types)**.** Given $\vdash \gamma : \Delta \to \Gamma$ and $\Gamma \vdash A$ type, we define the substitution $\gamma.A : \Delta.A[\gamma] \to \Gamma.A$ as the extension of the substitution $\gamma \circ p : \Delta.A[\gamma] \xrightarrow{p} \Delta \xrightarrow{\gamma} \Gamma$ with the term $\Delta.A[\gamma] \vdash q : A[\gamma \circ p] \equiv A[\gamma][p]$.

Extension of context substitutions satisfies various unsurprising properties that the reader might check, which we will be implicit in the fact that the rules in the theory will be well-typed, via straightforward computations that we will omit. Therefore, once we fix a rule with base context Γ, we might consider another context Δ (for which the rule is well-formed) with a substitution $\vdash \Delta \to \Gamma$ ctx, and systematically apply base change to all the types and terms along γ (for what lives over Γ) or the appropriate extension of γ (for what lives over some $\vdash \Gamma.\Xi$ ctx) appearing in the premises. In particular, this process returns a rule where everything is defined over (a context extension of) Δ, and the rule in base context Δ will yields an outcome that lives over (a context extension of) Δ. On the other hand, we could compute the outcome of the rule over Γ and then, since it lives over (a context extension of) Γ apply base change along γ. Therefore, we might compare the two results, which live over the same context, i.e. a context extension of Δ. When these results are judgementally equal, we say that the rule is *compatible* with base change. As a general principle, we will adopt the following:

Fundamental principle of base change. All the rules that introduce a type or a term are judgementally compatible with base change along contexts over which the rule is well-defined.

This means that whenever we write a rule, this will come equipped with its rule of compatibility with base change which we will omit by convention. Note that the rules that have a judgemental equality as an outcome do not need any condition of compatibility with base change. To give an example, the rules on the left are compatible with base change if the rules on the right hold:

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Gamma \vdash R(A,B) \text{ type}} \qquad \frac{\vdash \gamma : \Delta \to \Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Delta \vdash R(A,B)[\gamma] \equiv R(A[\gamma], B[\gamma.A]) \text{ type}}$$

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type} \quad \Gamma.A \vdash b : B}{\Gamma \vdash r(b) : R(A,B)} \qquad \frac{\vdash \gamma : \Delta \to \Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type} \quad \Gamma.A \vdash b : B}{\Delta \vdash r(b)[\gamma] \equiv r(b[\gamma.A]) : R(A,B)[\gamma]}.$$

By the fundamental principle of base change, we will always assume that base change preserves everything, thus every rule that introduces a type or a term will implicitly come equipped with a rule of compatibility with base change as the ones on the right. This compatibility holds strictly in order for the type-theoretical syntax to behave well. Furthermore, it allows the usage of the notation with variables without creating any issues. Note that we implicitly assume that we can base change only over those contexts for which the rule is well-formed. For instance, the first rule above could be well-defined only over animae, so that its rule of compatibility with composition will make sure that we apply base change along an anima:

$$\frac{\vdash \Gamma \text{ anima} \quad \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Gamma \vdash R(A,B) \text{ type}} \qquad \frac{\vdash \Gamma, \Delta \text{ anima} \quad \vdash \gamma : \Delta \to \Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Delta \vdash R(A,B)[\gamma] \equiv R(A[\gamma], B[\gamma.A]) \text{ type}}.$$

2.4 Intensional identity type

After the introduction of the strict calculus of substitutions, we now want to introduce type constructors that behave non-strictly. In other words, we want to introduce an internal notion of equality, as opposed to judgemental equality, that consists of providing equality data, so that the types we will define later on

will be allowed to obey a non-strict behaviour. In order to make precise what we mean by equality data, we need to introduce an identity type $a =_A b$ for any pair of terms $a, b : A$, whose terms shall be the proofs of an equality between them. These equality data will be called *propositional equalities*, and, being terms of a type, can further be compared and asked to coincide via a higher-order equality. Indeed, we would like these equalities to behave like homotopies, which can be required to satisfy further coherences. For this purpose, we need to introduce an intensional identity type, as opposed to an extensional equality type. In order to make this sentence meaningful, let us explain briefly what extensionality is, and why it does not suit our goals. Extensionality is characterised by the *equality reflection* rule, which means that if $a =_A b$ is inhabited, we can deduce that $a \equiv b : A$, and furthermore if an inhabitant exists it is *judgementally* unique. Semantically, in a clan, extensional equality of terms $(a, b) : * \to A \times A$ is a factorisation along the diagonal map: this exists if and only if a and b are strictly equal maps in the clan, and in that case the lift is unique. In other words, extensional equality fully reflects the behaviour of judgemental, strict, equality, realising it as a "yes or no" kind of question. This is deeply against the spirit of homotopy theory, where we regard equality data as paths between points: having a path between two points does not imply that they coincide, and furthermore any two points can be connected by means of various paths. Philosophically, our theory could be thought as "extensional" in the sense that judgemental equality will be put aside, and replaced by propositional equality; therefore, it is tautologically true that whenever the equality type $a =_A b$ is inhabited, the terms a and b are "equal" in the best way our theory wants to say it, which is not judgementally anymore. This is essentially the philosophy of extensionality. On the other side, it would still not be true that an inhabitant of the identity type, if it exists, is unique, even up to higher-order propositional equality. This can also be seen homotopically, as there can be non-trivial loops that are not homotopic to the constant loop.

These considerations lead us to considering an intensional theory, such as homotopy type theory or any type theory that can be semantically encoded in a tribe. Intensional type theory intrinsically constructs the identity type as a path object, and its terms resemble paths or homotopies. Indeed, we saw that, in a tribe, two terms are defined to be homotopic if and only if they factor through the identity type. In this setting we really care about the explicit lift we exhibit, and not just about the existence of one, which happens not to be unique anymore. This is analogous to the difference between working in an ∞-category and in its homotopy category. Providing an equality will always mean to provide a proof of it, and not claiming that there exists one: these data can be then imposed to satisfy further compatibilities.

As seen in the first chapter, one of the key features of the intensional identity type in homotopy type theory is that it allows us to recover all the higher coherences we wish by means of the J-elimination rule, and this is what endows types with the structure of ∞-groupoids with internal morphisms given by equalities. Although in synthetic category theory we will have a different notion of morphisms in a type, the equalities will represent the isomorphisms, thus they will still need the full coherences as in homotopy type theory. For instance, they shall be composable, with a unital and associative composition. Therefore, the construction of the intensional identity type will mimic the classical one. However, we will carefully need a change to the J-elimination rule and the computation rule. The first will be subdued to modifications due to the lack of arbitrary dependent products in the theory, the second will be stated in an unstrictified way, aligned to the formalism of objective type theory from [vdBdB21]. This unstrictification procedure will take place for every type constructor we will see.

2.4.1 Homotopy type theory as the theory of statements

In homotopy type theory, the identity type is a first instance on the perspective of regarding types as statements. Indeed, in Types as statements, we mentioned that in ordinary type theory every type can be interpret as a statement containing its proofs. Now that we developed a proper syntax to rely on, we can explain better this perspective. However, we also pointed out in Types as theories that, in our theory, only some types will be statements. This is crucially important, but for a matter of intuition this interpretation will be useful nevertheless. Indeed, various types we will introduce are borrowed from homotopy type theory, where every type is a statement. To fully understand their homotopy type theoretical meaning, it will be a good idea to have this interpretation. In particular, we will often make sure that those types that are thought as statements will still be statements in our theory.

Let us assume we are in homotopy type theory. A type A could be though as a statement, and its terms $a : A$ as its proofs. In particular, A could also be regarded as a parametrising domain of another statement: this is the idea of dependent types $a : A \vdash B(a)$ type. Here, B is regarded as a statement parametrised over A, i.e. a predicate. In particular, for every $a : A$ we have a statement $B(a)$ about a, which can be true or false depending on $B(a)$ being inhabited or not. Given A type, the identity type will be defined as the predicate "x is equal to y" parametrised over $x : A, y : A$, and will be a first instance of this interpretation.

Going on with this interpretation, a function of types $a : A \vdash f(a) : C$ becomes an implication between statements, since to any proof of A we get a proof of B. In particular, within homotopy type theory, we can immediately define what it means for two types to be logically equivalent: given $\Gamma \vdash A, B$ type, they are logically equivalent if there exist functions $\Gamma.A \vdash f : B[p]$ and $\Gamma.B \vdash g : A[p]$ in the opposite directions. A dependent function $a : A \vdash f(a) : C(a)$, instead, should be regarded as a conditional proof. In other words, it is a proof of $C(a)$ being true about every $a : A$.

2.4.2 Rules of the intensional identity type

First of all, we have the *formation rule*, that allows us to form the identity type of any type:

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma.A.A[p] \vdash \mathrm{Id}_A \text{ type}}.$$

In particular, given $\Gamma \vdash A$ type and $\Gamma \vdash a, b : A$, we have the fibre $\Gamma \vdash \mathrm{Id}_A(a, b) :\equiv \mathrm{Id}_A[\mathrm{id}\,.a.b]$ type. More concretely, the identity type is a dependent type $\Gamma, x : A, y : A \vdash \mathrm{Id}_A(x, y)$ type.

Definition 2.4.2.1. Let $\Gamma \vdash A$ type and let $\Gamma \vdash a, b : A$. We say that two terms a and b are *equal* or *homotopic* if such a type is inhabited. An inhabitant is called a *homotopy* or *equality* between a and b.

As pointed out, we do not want to work up to saying two terms are equal, as we care about the homotopy itself. However, sometimes, it will still be useful to study terms up to the relation of being equal, namely by working in the homotopy category. The above rule comes equipped with compatibility with base changes along $\vdash \gamma : \Delta \to \Gamma$ ctx as all the other rules. In particular, also its fibres $\mathrm{Id}_A(a, b)$ are judgementally compatible with base change.

The *introduction rule* states that every term is canonically equal to itself by means of a reflexive term. As the name suggests, it corresponds to the reflexivity of the relation of being equal:

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma.A \vdash \mathrm{refl} : \mathrm{Id}_A[\mathrm{id}\,.q]}.$$

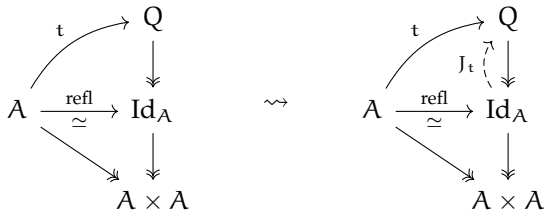
Fibrewise, this induces an inhabitant of the identity type of a term with itself: given $\Gamma \vdash A$ type with $\Gamma \vdash a : A$ we write $\Gamma \vdash \mathrm{refl}_a :\equiv \mathrm{refl}[\mathrm{id}\,.a] : \mathrm{Id}_A(a, a)$. Categorically, this should be though as an identity morphism of a inside A. In terms of variables, we can form a canonical term $\Gamma, x : A \vdash \mathrm{refl}_x : \mathrm{Id}_A(x, x)$. The reflexive term, and consequently each fibre, is itself compatible with base change.

As for any kind of well-behaved equality relation, we wish that the relation of being equal is in fact an equivalence relation. However, symmetry and transitivity will not be forcefully imposed, as a lot of coherences and further constructions should then be forced by hand in the same way. Instead, symmetry and transitivity will be elegantly deduced from the next two rules, that will exhibit the identity type as a first instance of an *inductive type*. As opposed to extensional equality types, which states that refl_a is the only inhabitant (even judgementally) of the equality type between a and itself, for intensional identity we will state that the theory "believes" that refl_a is the only inhabitant of $\mathrm{Id}_A(a, a)$. This is expressed by means of a so-called mapping-out property, where instead of characterising explicitly the terms (mapping-in property) of Id_A, we describe how to induce terms for dependent types over it.

Remark 2.4.2.2 (Inductive types)**.** Every inductive type we will meet throughout the theory will follow the pattern we will see now, of an induction principle followed by a computation rule. The idea is to specify a type by saying how to induce terms of dependent types over it. More explicitly, every inductive type will come equipped with term constructors (in this case the reflexive term), and in order to induce a term of a

dependent type over it, it will suffice to specify a term in the fibre at the term constructors. The induced term will then be an "extension" of it through the computation rule.

Roughly speaking, for the identity type, given $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x, y) \vdash Q(x, y, \alpha)$ type, if we produce a term of its fibre $\Gamma, x : A \vdash t : Q(x, x, \mathrm{refl}_x)$, we can canonically extend it to a term of the type Q. Semantically, in a tribe, we say that given a lift as in the left diagram, we can extend it to a section as in the diagram on the right:



Note that the first is a term of the fibre of Q over refl, whereas the second is a term of Q. The J-rule will formalise the given assignment, and the computation rule will state that the upper triangle containing J_t commutes, realising J_t as an extension of t.

The J-*elimination rule*, or *path induction* is subject to a modification compared to the ordinary formulation. The reason is that, in homotopy type theory, the process under which the J-rule provides all the coherences we wish relies on the existence of arbitrary dependent products. In our framework, these will not always exist, as we motivated. Thus, we will need to allow an arbitrary context to appear in the formulation. In some sense, this is the most complete formulation of the J-rule, which happens to be equivalent to a weaker formulation in homotopy type theory thanks to the presence of dependent products. We state it as follows:

$$\frac{\Gamma \vdash A \text{ type} \quad \vdash \Gamma.A.A[p].\mathrm{Id}_A.\Xi \,\mathrm{ctx} \quad \Gamma.A.A[p].\mathrm{Id}_A.\Xi \vdash Q \text{ type} \quad \Gamma.A.\Xi[\mathrm{id}.q.\mathrm{refl}] \vdash t : Q[\mathrm{id}.q.\mathrm{refl}.\Xi]}{\Gamma.A.A[p].\mathrm{Id}_A.\Xi \vdash J_\Xi(t) : Q}$$

This clearly mimics the assignment in semantics above, although it might look very abstract at a first sight. In more concrete notation, it says that given a dependent type $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x, y), \xi : \Xi(x, y, \alpha) \vdash Q(x, y, \alpha)$ type, if we produce a term of its fibre $\Gamma, x : A, \xi : \Xi(x, x, \mathrm{refl}_x) \vdash t : Q(x, x, \mathrm{refl}_x, \xi)$, we can canonically "extend" it to a term of the type Q (where the term extension will be formalised by the computation rule). Note that, because Ξ is a context extension of arbitrary length of $\Gamma.A.A[p].\mathrm{Id}_A$, when we write $\xi : \Xi(x, y, \alpha)$ we possibly mean an array of variables in the components of Ξ. This rule allows to induce terms of dependent types over the identity type by pretending the only inhabitant is the reflexive term. If we think of Q as a statement in the assumptions of x and y being equal via α, this rule says that to provide a proof of this statement it suffices to provide a proof in the case $x \equiv y$ and where we pretend the equality term is refl_x. Note that even in the case $x \equiv y$, the equality term in $\mathrm{Id}_A(x, x)$ can be far from being the reflexive term. Therefore, such a rule is incredibly powerful and will allow us to induce constructions over equalities in a much simpler way, as we will always be able to assume the equality to be the reflexive term. In this sense, the theory *behaves as* if the only term of $\mathrm{Id}_A(x, x)$ was refl_x.

The adjustment of this rule, compared to the classical one, is the appearance of an arbitrary context Ξ over $\Gamma.A$. In the usual formulation of homotopy type theory, Ξ is taken as the empty extension, thus ours is a stronger version of the rule. This change will allow us to deduce the desired constructions and compatibilities about equalities, which the second part of this section will be about. Despite the fact that homotopy type theory possesses the "weak" version of this rule, equalities are still coherent and compatible with every construction of the theory thanks to the presence of arbitrary dependent products. Later on, we will allow the formation of dependent products only over some types, therefore we cannot rely on them to deduce the coherences, and thus we are forced to introduced this stronger form. It is not surprising that dependent products would make the rule above equivalent to the one with empty Ξ. Indeed, as we will see, dependent product is the operation that allows to carry Ξ on the right, so that then we may apply the "weak" rule on the dependent product to get the same outcome of the stronger formulation. Note that, in the intuitive interpretation given above, the context Ξ did not seem to play any role: in fact, it will be crucial only to exhibit some specific constructions. We will point out for which applications this strengthening of the rule will not be necessary. Finally, since this elimination rule provides a term of a type, it will also come

equipped with a compatibility rule with base change of Γ.

Finally, we want to conclude the construction of the identity type with the computation rule. After giving an assignment for every term t of the fibre of Q over refl of a term J_t^{Ξ} of the type Q, we cannot still say anything about how it relates to the starting term. The computation rule says that J_t^{Ξ} restricts to the original term t, in the sense that its fibre over refl will give back t. In other words, J_t is an extension of the term t to the whole type Q. In the semantic picture we provided above, the J-rule guarantees the assignment to every lift of Q along $\mathrm{refl} : A \to \mathrm{Id}_A$ of a section of Q, and the computation rule assures that the provided section is an extension of the lift $A \to Q$ to $\mathrm{Id}_A \to Q$. Although in the classical setting the commutativity of the triangle is strict, and thus the section $\mathrm{Id}_A \to Q$ is a proper extension of the lift $A \to Q$ along $\mathrm{refl} : A \to \mathrm{Id}_A$, here we will modify this rule by asking this triangle to commute only up to homotopy, meaning that the section will extend the original lift only up to homotopy. This is the first rule in this syntax where we remove the strictness from the way it is presented classically. The *computation rule* is stated as:

$$\frac{\Gamma \vdash A\ \mathrm{type} \quad \vdash \Gamma.A.A[p].\mathrm{Id}_A\ .\Xi\ \mathrm{ctx} \quad \Gamma.A.A[p].\mathrm{Id}_A\ .\Xi \vdash Q\ \mathrm{type} \quad \Gamma.A.\Xi[\mathrm{id}\ .q.\ \mathrm{refl}] \vdash t : Q[\mathrm{id}\ .q.\ \mathrm{refl}\ .\Xi]}{\Gamma.A.\Xi[\mathrm{id}\ .q.\ \mathrm{refl}] \vdash \mathrm{comp}^{\mathrm{Id}}_{\Xi}(t) : \mathrm{Id}_{Q[\mathrm{id}\ .q.\ \mathrm{refl}\ .\Xi]}(J_{\Xi}(t)[\mathrm{id}\ .q.\ \mathrm{refl}\ .\Xi], t)}.$$

The computation rule does not usually require a rule determining its compatibility with context substitution, since the standard formulation claims a judgemental equality instead of a propositional equality. Since we made it non-strict, there will be a further rule making sure that this computation term is compatible with base change.

This concludes the rules of the intensional identity type. Although they might look hard to be used in practice, we will now explore various consequences. Instead, it will be surprisingly natural how various construction we want to perform follow elegantly from them. On one side, these applications will provide a better intuition on how the rules of the identity type can be used, and on the other side we will be able to deduce key properties of equality and the coherences we want in our agnostic theory. In other words, the rest of this section about the identity type will be devoted to showing we have a nice theory of equalities.

Notation 2.4.2.3. We will frequently study base changes of the identity type, where the terms we equate are not exactly the variables in the context. Indeed, given $\Gamma \vdash A, B$ type and a function of types $\Gamma.A \vdash f : B[p]$, we define:

(1) $\Gamma.A.A[p] \vdash f^*\,\mathrm{Id}_B :\equiv \mathrm{Id}_B(f(q[p]), f(q))$, more explicitly given by $\Gamma, a : A, a' : A \vdash (f^*(\mathrm{Id}_B))(a, a') \equiv \mathrm{Id}_B(f(a), f(a'))$.

(2) $\Gamma.A.B[p] \vdash (f\times 1)^*\,\mathrm{Id}_B :\equiv \mathrm{Id}_B(f(q[p]), q)$, more explicitly given by $\Gamma, a : A, b : B \vdash ((f\times q)^*(\mathrm{Id}_B))(a, b) \equiv \mathrm{Id}_B(f(a), b)$.

(3) $\Gamma.B.A[p] \vdash (1\times f)^*\,\mathrm{Id}_B :\equiv \mathrm{Id}_B(q[p], f(q))$, more explicitly given by $\Gamma, b : B, a : A \vdash ((q\times f)^*(\mathrm{Id}_B))(b, a) \equiv \mathrm{Id}_B(b, f(a))$.

All these standard base changes of the identity type will frequently appear.

Before moving to a detailed discussion of the identity type and the relations satisfies by equalities, we will give an interpretation of what equality will mean, categorically.

Remark 2.4.2.4. In homotopy type theory, types have the structure of ∞-groupoids, with objects given by their terms, and morphisms given by equalities of terms. This interpretation is intuitive since all morphisms in an ∞-groupoid should be isomorphisms, thus identifications of objects. Because homotopy is not the strict equality of terms, it is allowed to describe the notion of (iso)morphism in ∞-groupoids. One of the goals of this section is to show all the fundamental properties that equalities enjoy, realising them as a good notion of morphisms in ∞-groupoids. In particular, through the rules of the identity type, we will show that they compose, they are invertible, and their composition has all the wished coherences.

However, in our theory we will regard types over animae as ∞-categories, with objects given by (some of) their terms. We want a *directed* notion of morphism between objects, that is not an identification, i.e. that is not invertible with respect to composition in general. Indeed, it is obviously not true that every morphism in a category is an isomorphism, unlike groupoids. Therefore, in our theory, equalities of objects will *not* be all morphisms between them. Instead, they will be *some* of the morphisms: the invertible ones.

Remark 2.4.2.5. Because the theory of ∞-groupoids is agnostic about the context, talking about generalised terms is literally the same as (and more natural than) talking about the absolute terms of a type, thus its absolute objects. In particular, generalised terms are functions of types, which assume the role of functors of ∞-groupoids in homotopy type theory. Therefore, everything we said in Remark 2.4.2.4 could be equivalently reinterpreted by replacing objects of a type A with functors into A. Thus, the notion of equality of these would resemble that of natural transformation, hence isomorphism, of functors of ∞-groupoids.

This dual interpretation partially breaks in our setting about the theory of categories. First of all, only types over animae will be regarded as proper categories, and for them, only their (absolute) terms and some of their generalised terms will provide a good notion of objects. Instead, more arbitrary generalised terms will only be interpreted as functors into them. Note that, in order to fix the strictness of their composition, though, this interpretation will be true only up to homotopy: later on we will define actual functors of types that overcome this problem. However, currently, thinking functions of types as functors fits our purposes. As a consequence of the considerations above, we will need to distinguish between objects of a category A and functors into A. In Remark 2.4.2.4, we saw how the object-interpretation of equalities is linked to the idea of isomorphisms in a category. However, the same discussion can be done for generalised terms of a category, as the theory did not differentiate animae from arbitrary contexts yet. Under this viewpoint, generalised terms of a category A will be functors into A, and equalities between these will be *some* natural transformations of functors into A, namely the invertible ones. In particular, the identity type of a category A will not only describe the isomorphisms between objects of A, but, more in general, natural isomorphisms between functors into A. Indeed, when we will meet a context $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x, y)$, the variables x and y may stand for any pair of functors into A, and α for what will be a natural isomorphism of functors into A.

2.4.3 Compatibility of functions with equality

A first application of path induction will be to show that functions of types automatically preserve equalities, coherently with the fact that the whole theory should behave nicely with respect to homotopy. In other words, given $\Gamma \vdash A, B$ type and a function of types $\Gamma, x : A \vdash f(x) : B$, it is a natural question to ask if two equal terms $\alpha : \mathrm{Id}_A(a, b)$ induce equal terms $f(a)$ and $f(b)$ of the type B, and the answer is indeed always positive.

This takes further the discussion of Remark 2.3.8.1, where we saw that every (dependent) function is automatically functorial on all terms. Indeed, as seen in Remark 2.4.2.4, in homotopy type theory equalities are morphisms in ∞-groupoids, whereas in this theory they will be *some* of the morphisms in a type. Here, we show that functions, which are automatically functorial on all generalised objects, also have an action on equalities, i.e. on internal morphisms in homotopy type theory, and on "isomorphisms" in the present theory. It is to be specified, though, that this interpretation is more relevant in homotopy type theory than in our setting, as the more general definition of directed morphism in a type A will in particular be a specific form of generalised term in A. Therefore, the functorial action of functions on morphisms will directly follow from Remark 2.3.8.1. Still, it is a very important perspective in the theory of ∞-groupoids and we will make a wide use of this result in our framework.

Lemma 2.4.3.1. Assume $\Gamma \vdash A, B$ type and consider a function $\Gamma, x : A \vdash f(x) : B$. Then we can construct the following terms:

(1) $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x, y) \vdash \mathrm{ap}_f(\alpha) : \mathrm{Id}_B(f(x), f(y))$.

(2) $\Gamma, x : A \vdash \mathrm{comp}^f(x) :\equiv \mathrm{comp}^{\mathrm{Id}}(\mathrm{refl})(x) : \mathrm{Id}_{\mathrm{Id}_B(f(x), f(x))}(\mathrm{ap}_f(\mathrm{refl}_x), \mathrm{refl}_{f(x)})$.

Proof. By the J-elimination rule, we can construct such a term over the fibre at $y \equiv x$ and $\alpha \equiv \mathrm{refl}_x$. The reflexive term $\Gamma, x : A \vdash \mathrm{refl}_{f(x)} : \mathrm{Id}_B(f(x), f(x))$ induces a term $\mathrm{ap}_f(\alpha) :\equiv J(\mathrm{refl}_{f(-)})(x, y, \alpha)$ as desired. The last claim in the statement is the computation rule. □

Note that, for this first application, the softer version of the J-rule with $\Xi = \emptyset$ would be enough. This result makes us deduce that functions of types are functorial in an even stronger way than just on generalised terms: they respect equalities. In particular, if we regard equalities as (invertible) morphisms in a type, the

reflexive term would be the identical morphism, and (2) states that every function preserves this identical morphism.

Remark 2.4.3.2 (Application of f as left-whiskering)**.** In Remark 2.4.2.5, we saw that variables in the context $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x, y)$ will also be interpreted, later on, as a pair of functors into A equipped with a natural isomorphism between them. Therefore, Lemma 2.4.3.1 will mean that given a natural isomorphism of functors between $x, y : C \to A$, we obtain a natural isomorphism between the composites $f(x), f(y) : C \to B$. For this reason, we may call $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x, y) \vdash \mathrm{ap}_f(\alpha) : \mathrm{Id}_B(f(x), f(y))$ the *whiskering of* α *with* f *on the left*, following the terminology of category theory.

As we expect, we can construct a right-whiskering as well. However, this is constructed and formulated differently compared to the left-whiskering: in fact, it is even more elementary and intrinsic in the basics.

Remark 2.4.3.3. Let $\Gamma \vdash A, B, C$ type, and assume we have functions of types $\Gamma.A \vdash f : B[p]$ and $\Gamma.B \vdash g, g' : C[p]$. Assume further that we have $\Gamma, y : B \vdash \gamma(y) : \mathrm{Id}_C(g(y), g'(y))$. A base change along f yields $\Gamma, x : A \vdash \gamma(f)(x) :\equiv \gamma(f(x)) : \mathrm{Id}_C(g(f(x)), g'(f(x)))$. We might call $\gamma(f)$ the *whiskering of* γ *with* f *on the right*.

2.4.4 Inversion of equalities

The first construction that we will provide is the inversion of equalities. This encodes the symmetry of the relation of being equal, and corresponds to the fact that homotopies are always invertible as they can be traversed in the opposite direction. In particular, we want to provide a function of types from $\mathrm{Id}_A(a, b)$ to $\mathrm{Id}_A(b, a)$ representing the inversion of paths. The name inversion will be more appropriate after we will define a composition operation on equalities and show the following inverts equalities with respect to it. In homotopy type theory, the invertibility of equalities reflects into the invertibility of morphisms in ∞-groupoids.

Proposition 2.4.4.1 (Inversion of equalities)**.** Let $\Gamma \vdash A$ type. We can construct the following terms.

(1) *Inversion:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x, y) \vdash \alpha^{-1} :\equiv \mathrm{J}(\mathrm{refl})(x, y, \alpha) : \mathrm{Id}_A(y, x)$.

(2) *Computation of inversion:* $\Gamma, x : A \vdash \mathrm{comp}_{\mathrm{Id}}(\mathrm{refl})(x) : \mathrm{Id}_{\mathrm{Id}_A(x,x)}(\mathrm{refl}_x^{-1}, \mathrm{refl}_x)$.

(3) *Compatibility of inversion with equality:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x, y), \alpha' : \mathrm{Id}_A(x, y), \sigma : \mathrm{Id}_{\mathrm{Id}_A(x,y)}(\alpha, \alpha') \vdash \mathrm{ap}_{(-)^{-1}}(\sigma) : \mathrm{Id}_{\mathrm{Id}_A(y,x)}(\alpha^{-1}, \alpha'^{-1})$.

Proof. For (1), by path induction it suffices to provide a term in the case $y \equiv x$ and $\alpha \equiv \mathrm{refl}_x$, and the obvious choice is $\Gamma, x : A \vdash \mathrm{refl}_x : \mathrm{Id}_A(x, x)$. (2) follows by the computation rule and (3) follows by Lemma 2.4.3.1. □

In particular, we provided a function of types from $\mathrm{Id}_A(x, y)$ to $\mathrm{Id}_A(y, x)$ over $\Gamma, x : A, y : A$, called *inversion*, which is compatible with equality and sends the reflexive term to itself, up to homotopy. Whenever we have an equality between two terms we will canonically get an equality in the converse direction. This will be tacitly used throughout the whole theory. We will soon endow equalities with a composition operation and show inversion does, in fact, invert equalities with respect to it.

2.4.5 Transport between fibres

A key aspect of the identity type is that the fibres over two propositionally equal terms have the same terms up to homotopy, compatibly with context substitution. More explicitly, given a dependent type B over A, and two equal terms a and a' of A, the J-rule allows us to construct a comparison function $B(a) \to B(a')$ over Γ. This will provide what we call an equivalence of types.

Proposition 2.4.5.1 (Transport between fibres)**.** Let $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. We can construct the following terms.

(1) *Transport:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x, y), z : B(x) \vdash \mathrm{tr}_\alpha(z) : B(y)$.

(2) *Transport along* refl*:* $\Gamma, x : A, z : B(x) \vdash \mathrm{refltr}(x, z) :\equiv \mathrm{comp}^{\mathrm{Id}}_B(q)(x, z) : \mathrm{Id}_{B(x)}(\mathrm{tr}_{\mathrm{refl}_x}(z), z)$.

(3) *Compatibility with equality in the variable z:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : B(x), z' : B(x'), \beta : \mathrm{Id}_{B(x)}(z,z') \vdash \mathrm{ap}_{\mathrm{tr}_\alpha}(\beta) : \mathrm{Id}_{B(y)}(\mathrm{tr}_\alpha(z), \mathrm{tr}_\alpha(z'))$.

(4) *Compatibility with the reflexive term in the variable z:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : B(x) \vdash \mathrm{comp}^{\mathrm{tr}_\alpha}(z) : \mathrm{Id}_{\mathrm{Id}_{B(y)}(\mathrm{tr}_\alpha(z), \mathrm{tr}_\alpha(z))}(\mathrm{ap}_{\mathrm{tr}_\alpha}(\mathrm{refl}_z), \mathrm{refl}_{\mathrm{tr}_\alpha(z)})$.

(5) *Compatibility with equality in α:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), \alpha' : \mathrm{Id}_A(x,y), \sigma : \mathrm{Id}_{\mathrm{Id}_A(x,y)}(\alpha, \alpha'), z : B(x) \vdash \mathrm{ap}_{\mathrm{tr}_{(-)}(z)}(\sigma) : \mathrm{Id}_{B(y)}(\mathrm{tr}_\alpha(z), \mathrm{tr}_{\alpha'}(z))$.

(6) *Compatibility with the reflexive term in α:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : B(x) \vdash \mathrm{comp}^{\mathrm{tr}_{(-)}(z)}(x, y, \alpha, z) : \mathrm{Id}_{\mathrm{Id}_{B(y)}(\mathrm{tr}_\alpha(z), \mathrm{tr}_\alpha(z))}(\mathrm{ap}_{\mathrm{tr}_{(-)}(z)}(\mathrm{refl}_\alpha), \mathrm{refl}_{\mathrm{tr}_\alpha(z)})$.

(7) *Chain of transports:* For every chain of dependent types $A_1 \dots A_n$ over Γ and for every $\Gamma.A_1 \dots A_n \vdash C$ type, we construct inductively $\Gamma, x_1 : A_1, x_1' : A_1, \alpha_1 : \mathrm{Id}_{A_1}(x_1, x_1'), \dots, x_n : A_n(x_1 \dots x_{n-1}), x_n' : A_n(x_1', \dots, x_{n-1}'), \alpha_n : \mathrm{Id}_{A_n(x_1' \dots x_n')}(\mathrm{tr}_{\alpha_1 \dots \alpha_{n-1}}(x_n), x_n'), z : C(x_1 \dots x_n) \vdash \mathrm{tr}_{\alpha_1 \dots \alpha_n}(z) : C(x_1' \dots x_n')$.

Proof. For (1), path induction on the universal term $\Gamma, x : A, z : B(x) \vdash z : B(x)$ induces $\mathrm{tr}_\alpha(z) :\equiv J_B(q)(x, z, \alpha)$, and the computation rule yields (2). Now, note that transport $\mathrm{tr}_\alpha(z)$ can be regarded either as a function of types in z, from $B(x)$ to $B(y)$ over $\Gamma, x, y : A, \alpha : \mathrm{Id}_A(x,y)$, or as a function of types in α, from $\mathrm{Id}_A(x,y)$ to $B(y)$ over $\Gamma, x, y : A, z : B(x)$. Lemma 2.4.3.1 applied to these two interpretations of tr as a function yield the pairs (3), (4) (function in z) and (5), (6) (function in α). (7) is proved with the J-rule as (1) inductively on the length of Ξ. □

Note that, in the proof, we saw that we can interpret transport either as a function between fibres, from $B(x)$ to $B(y)$ in context $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y)$, or as a function in α, from $\mathrm{Id}_A(x,y)$ to $B(y)$ over $\Gamma, x : A, y : A, z : B(x)$. This is what allowed to deduce the preservation of equalities in both z and α.

A first application of transport is that we can provide a generalisation of Lemma 2.4.3.1 to dependent functions. Indeed, given a dependent function $\Gamma.A \vdash f : B$, the evaluations at two equal terms $a, a' : A$ via $\alpha : \mathrm{Id}_A(a, a')$ live in two different types $f(a) : B(a)$ and $f(a') : B(a')$. By Proposition 2.4.5.1, we can transport $f(a)$ to the fibre $B(a')$ and ask whether it is equal to $f(a')$, i.e. if f respects equalities.

Lemma 2.4.5.2 (Dependent functions preserve equalities)**.** Let $\Gamma \vdash A$ type and let $\Gamma.A \vdash B$ type with a dependent function $\Gamma, x : A \vdash f(x) : B(x)$. We can construct the following terms.

(1) *Preservation of equalities:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{ap}_f(\alpha) : \mathrm{Id}_{B(y)}(\mathrm{tr}_\alpha(f(x)), f(y))$.

(2) *Preservation of* refl*:* $\Gamma, x : A \vdash \mathrm{Id}_{\mathrm{Id}_{B(x)}(\mathrm{tr}_{\mathrm{refl}_x}(f(x)), f(x))}(\mathrm{ap}_f(\mathrm{refl}_x), \mathrm{refltr}(x, f(x)))$.

Proof. By path induction it suffices to assume $y \equiv x$ and $\alpha \equiv \mathrm{refl}_x$, thus the term $\Gamma, x : A \vdash \mathrm{refltr}(x, f(x)) : \mathrm{Id}_{B(x)}(\mathrm{tr}_{\mathrm{refl}_x}(f(x)), f(x))$ provides the term in (1), and the computation rule constructs (2). □

Now we will study in detail a particular case of transport: composition of equalities.

2.4.6 Composition of equalities

A fundamental property of equalities is the fact that we can compose, as a particular case of transport, namely when the dependent type B is itself an identity type. In homotopy type theory, it is the composition of morphisms in an ∞-groupoid. In particular, it provides the transitivity of the relation of being equal, which is a property that any kind of identification should have. Being a particular case of transport, this will once again need the stronger version of the J-rule in absence of dependent products.

Proposition 2.4.6.1 (Composition of equalities)**.** Let $\Gamma \vdash A$ type. We can construct the following terms.

(1) *Composition:* $\Gamma, z : A, y : A, \beta : \mathrm{Id}_A(y,z), x : A, \alpha : \mathrm{Id}_A(x,y) \vdash \beta \circ \alpha :\equiv \mathrm{tr}_\beta(\alpha) : \mathrm{Id}_A(x,z)$.

(2) *Left unitality:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{lun}_\alpha :\equiv \mathrm{refltr}(x, \alpha) : \mathrm{Id}_{\mathrm{Id}_A(x,y)}(\mathrm{refl}_y \circ \alpha, \alpha)$.

(3) *Right unitality:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{run}_\alpha :\equiv J(\mathrm{lun}_{\mathrm{refl}})(x, y, \alpha) : \mathrm{Id}_{\mathrm{Id}_A(x,y)}(\alpha \circ \mathrm{refl}_x, \alpha)$.

(4) *Unitality on* refl*:* $\Gamma, x : A \vdash \mathrm{comp}_{\mathrm{Id}}(\mathrm{lun}_{\mathrm{refl}}) : \mathrm{Id}_{\mathrm{Id}_{\mathrm{Id}_A(x,x)}(\mathrm{refl}_x \circ \mathrm{refl}_x, \mathrm{refl}_x)}(\mathrm{run}_{\mathrm{refl}_x}, \mathrm{lun}_{\mathrm{refl}_x})$.

(5) *Alternative composition:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : A, \beta : \mathrm{Id}_A(x,y) \vdash \mathrm{Id}_{\mathrm{Id}_A(x,z)}(\mathrm{tr}_{\alpha^{-1}}(\beta), \beta \circ \alpha)$ type is inhabited.

Proof. (1) and (2) follow from transport, Proposition 2.4.5.1. For (3), we do a further path induction: assuming $y \equiv x$ and $\alpha \equiv \mathrm{refl}_x$, the term $\Gamma, x : A \vdash \mathrm{lun}_{\mathrm{refl}_x} : \mathrm{Id}_{\mathrm{Id}_A(x,x)}(\mathrm{refl}_x \circ \mathrm{refl}_x, \mathrm{refl}_x)$ induces the claimed term. The associated computation rule provides (4). For (5), by iterating path induction we may assume $z \equiv y \equiv x$ and $\beta \equiv \alpha \equiv \mathrm{refl}_x$. Thus, it suffices to provide a homotopy between $\mathrm{tr}_{\mathrm{refl}_x^{-1}}(\mathrm{refl}_x)$ and $\mathrm{tr}_{\mathrm{refl}_x}(\mathrm{refl}_x)$. For this, apply Lemma 2.4.3.1 on the function $\mathrm{tr}_{(-)}(\mathrm{refl}_x)$ and the equality $\mathrm{refl}_x^{-1} = \mathrm{refl}_x$ from Proposition 2.4.4.1. The iterated J-rule provides the desired term. □

We call $\beta \circ \alpha$ the *(vertical) composition of* α *and* β. Therefore, we provided a composition of equalities function, which is unital with respect to the reflexive term, up to homotopy. Furthermore, in (4) we saw that the two ways of equating $\mathrm{refl}_x \circ \mathrm{refl}_x$ to refl_x by using unitality on the left or on the right coincide up to higher homotopy. Finally, in (5) we saw how transport inside an identity type acquires more symmetry, given by the fact that we could indifferently transport either equality along the other, up to inverting correctly, and the result will not change. In other words, we constructed the same composition assignment by transporting α along β or β along α^{-1}. By convention, we will always think of composition as a transport along the postcomposed equality as in (1).

Remark 2.4.6.2. The relation of being equal on the terms of a type is an equivalence relation, thanks to the reflexive term, the inversion and the vertical composition of equalities.

Note that, for the construction of the composition function, the strengthening of the J-rule had a crucial role. Furthermore, the reader might be surprised by the asymmetry in the construction of the equalities (2) and (3). This is a consequence of the fact that the two copies of the identity type in the context had two very different roles: one was used for path induction, whereas the other was in the inert context Ξ. More, in homotopy type theory, (2) even holds judgementally, asymmetrically compared to (3).

Remark 2.4.6.3. In homotopy type theory, we saw that equalities of terms of a type correspond to (natural) isomorphisms of objects (functors into the type). In particular, they compose via composition of equalities. It will be our responsibility to show that composition of equalities satisfies the desired coherences to deserve this interpretation.

Since we defined a composition and an inversion operation, the first thing will be to understand their relation. In particular, we show that the name inversion comes from the fact that it inverts equalities with respect to composition.

Proposition 2.4.6.4 (Inversion inverts equalities)**.** Let $\Gamma \vdash A$ type. We can construct the following terms.

(1) *Inversion is right inverse:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{rinv}_{\mathrm{Id}}(\alpha) : \mathrm{Id}_{\mathrm{Id}_A(x,x)}(\alpha \circ \alpha^{-1}, \mathrm{refl}_x)$.

(2) *Inversion is left inverse:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{linv}_{\mathrm{Id}}(\alpha)\, \mathrm{Id}_{\mathrm{Id}_A(x,x)}(\alpha^{-1} \circ \alpha, \mathrm{refl}_x)$.

(3) *Inversion is an involution:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{inv}_{(-)^{-1}}(\alpha) : \mathrm{Id}_{\mathrm{Id}_A(x,y)}((\alpha^{-1})^{-1}, \alpha)$.

Proof. Recall that we have the homotopy $\Gamma, x : A \vdash \mathrm{comp}_{\mathrm{Id}}(\mathrm{refl})(x) : \mathrm{Id}_{\mathrm{Id}_A(x,x)}(\mathrm{refl}_x^{-1}, \mathrm{refl}_x)$ by Proposition 2.4.4.1. Both (1) and (2) follow by path induction applied respectively to $\Gamma, x : A \vdash \mathrm{run}(\mathrm{refl}_x) : \mathrm{Id}_{\mathrm{Id}_A(x,x)}(\mathrm{refl}_x^{-1} \circ \mathrm{refl}_x, \mathrm{refl}_x^{-1})$ and $\Gamma, x : A \vdash \mathrm{lun}_{\mathrm{Id}}(x) : \mathrm{Id}_{\mathrm{Id}_A(x,x)}(\mathrm{refl}_x \circ \mathrm{refl}_x^{-1}, \mathrm{refl}_x^{-1})$ postcomposed with $\mathrm{comp}_{\mathrm{Id}}(\mathrm{refl})(x)$. (3) follows again by path induction, since $(\mathrm{refl}_x^{-1})^{-1}$ is homotopic to refl_x^{-1} and then to refl_x, by means of Proposition 2.4.4.1 on the inversion function. The J-rule gives the claim. □

Remark 2.4.6.5. In homotopy type theory, Proposition 2.4.6.4 says that equalities, as morphisms in a type, are always invertible with respect to composition, under the inversion construction. On the one hand, this makes us realise why homotopy types have a groupoidal behaviour. On the other side, though, it makes us hopeless about describing equalities as internal morphisms in our theory, since we want to describe categories with non-invertible morphisms.

2.4.7 Compatibility of tr and ap with composition and inversion

We started the discussion about the rules of the identity type by defining the operators ap and tr. In particular, they both take equalities as inputs. Now that we endowed equalities with a well-behaved composition and inversion operation, we question ourselves if the operators ap and tr are compatible with it. Indeed, we will get the expected relations. This will, in particular, lead us to prove associativity of composition of equalities. Let us start with the ap operator.

Lemma 2.4.7.1 (Compatibility of ap with inversion and composition)**.** Let $\Gamma \vdash A, B, C$ type and consider functions $\Gamma, x : A \vdash f(x) : B$ and $\Gamma, y : B \vdash g(y) : C$. The following types are inhabited.

(1) $\Gamma, x : A, y : A, z : A, \alpha : \mathrm{Id}_A(x,y), \beta : \mathrm{Id}_A(y,z) \vdash \mathrm{Id}_{\mathrm{Id}_B(f(x),f(z))}(\mathrm{ap}_f(\beta \circ \alpha), \mathrm{ap}_f(\beta) \circ \mathrm{ap}_f(\alpha), \mathrm{ap}_f(\beta \circ \alpha))$ type.

(2) $\Gamma, x : A, y : A \vdash \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{Id}_{\mathrm{Id}_B(f(y),f(x))}(\mathrm{ap}_f(\alpha^{-1}), \mathrm{ap}_f(\alpha)^{-1})$ type.

(3) $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{Id}_{\mathrm{Id}_C(g(f(x)),g(f(y)))}(\mathrm{ap}_g(\mathrm{ap}_f(\alpha)), \mathrm{ap}_{g\circ f}(\alpha))$ type.

(4) $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{Id}_{\mathrm{Id}_A(x,y)}(\mathrm{ap}_q(\alpha), \alpha)$ type.

Proof. For (1), by iterating path induction we may assume that $z \equiv y \equiv x$ and $\alpha \equiv \beta \equiv \mathrm{refl}_x$, so that it suffices to induce a homotopy between $\mathrm{ap}_f(\mathrm{refl}_x \circ \mathrm{refl}_x)$ and $\mathrm{ap}_f(\mathrm{refl}_x) \circ \mathrm{ap}_f(\mathrm{refl}_x)$. This can be done by equating both to $\mathrm{refl}_{f(x)}$, by composing the equalities $\mathrm{ap}_f(\mathrm{refl}_x \circ \mathrm{refl}_x) = \mathrm{ap}_f(\mathrm{refl}_x) = \mathrm{ap}_f(\mathrm{refl}_x) \circ \mathrm{refl}_{f(x)} = \mathrm{ap}_f(\mathrm{refl}_x) \circ \mathrm{ap}_f(\mathrm{refl}_{f(x)})$ following from Proposition 2.4.6.1 on ap_f and Lemma 2.4.3.1. The iterated J-rule concludes the argument. (2) similarly follows by path induction, by equating $\mathrm{ap}_f(\mathrm{refl}_x^{-1})$ and $\mathrm{ap}_f(\mathrm{refl}_x)^{-1}$ to $\mathrm{refl}_{f(x)}$ by Lemma 2.4.3.1 and Proposition 2.4.4.1. (3) follows by path induction by equating $\mathrm{ap}_g(\mathrm{ap}_f(\mathrm{refl}_x))$ and $\mathrm{ap}_{g\circ f}(\mathrm{refl}_x)$ to $\mathrm{refl}_{g(f(x))}$ by means of Lemma 2.4.3.1 applied iteratively. Finally, (4) is a simple path induction by means of Lemma 2.4.3.1. □

Remark 2.4.7.2. In homotopy type theory, where equalities are internal morphisms in ∞-groupoids, these statement have a familiar interpretation. For instance, (1) is a functoriality condition for a function with respect to composition of morphisms. Together with Lemma 2.4.3.1, this realises functions as a good notion of functor in homotopy type theory. Although it is too early to explain it, in our setting, this condition should be regarded as functoriality of functions with respect to invertible morphisms.

Now, with other path induction arguments, we will show that also the transport function is coherent with composition and inversion in the way we expect.

Proposition 2.4.7.3 (Compatibilities of tr)**.** Let $\Gamma \vdash A$ type and let $\Gamma.A \vdash B$ type. We have:

(1) *Compatibility with* $- \circ -$*:* $\Gamma, x : A, y : A, z : A, \alpha : \mathrm{Id}_A(x,y), \beta : \mathrm{Id}_A(y,z), w : B(x) \vdash \mathrm{coh}^{A;B}_{\mathrm{tr},-\circ-}(\alpha, \beta, w) : \mathrm{Id}_{B(z)}(\mathrm{tr}_\beta(\mathrm{tr}_\alpha(w)), \mathrm{tr}_{\beta\circ\alpha}(w))$.

(2) There exists a canonical inhabitant of the following type: $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), w : B(y) \vdash \mathrm{Id}_{\mathrm{Id}_B(y)(\mathrm{tr}_{\mathrm{refl}_y}(\mathrm{tr}_\alpha(w)), \mathrm{tr}_{\mathrm{refl}_y \circ \alpha}(w))}(\mathrm{coh}^{A;B}_{\mathrm{tr},-\circ-}(\alpha, \mathrm{refl}_y, w), (\mathrm{ap}_{\mathrm{tr}_{(-)}(w)}(\mathrm{lun}_\alpha))^{-1} \circ \mathrm{refltr}(y, \mathrm{tr}_\alpha w))$ type.

(3) *Compatibility with* $(-)^{-1}$*:* $\Gamma, x, y : A, \alpha : \mathrm{Id}_A(x,y), w : B(x) \vdash \mathrm{ap}_{\mathrm{tr}_{(-)}(z)}(\mathrm{linv}_{\mathrm{Id}}(\alpha)) \circ \mathrm{coh}^{A;B}_{\mathrm{tr},-\circ-}(\alpha, \alpha^{-1}, w) : \mathrm{Id}_{B(x)}(\mathrm{tr}_{\alpha^{-1}}(\mathrm{tr}_\alpha(w)), w)$ and analogously on the other side.

(4) *Compatibility of iterated transport with* $(-)^{-1}$*:* Given $\Gamma.A_1 \dots A_n \vdash C$ type, the following type is inhabited $\Gamma, x_1 : A_1, x_1' : A_1, \alpha_1 : \mathrm{Id}_{A_1}(x_1, x_1'), \dots, x_n : A_n(x_1, \dots, x_{n-1}), x_n' : A_n(x_1', \dots, x_{n-1}'), \alpha_n : \mathrm{Id}_{A_n(x_1', \dots, x_{n-1}')}(\mathrm{tr}_{\alpha_1,\dots,\alpha_{n-1}}(x_n', x_n), w : C(x_1, \dots, x_n) \vdash \mathrm{Id}_{C(x_1', \dots, x_n')}(\mathrm{tr}_{\alpha_1^{-1},\dots,\alpha_n^{-1}}(\mathrm{tr}_{\alpha_1,\dots,\alpha_n}(w), w)$.

(5) *Functions preserve transport:* For every $\Gamma.A \vdash C$ type and $\Gamma.A.B \vdash g : C[p]$ function, we have $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : B(x) \vdash g_{\mathrm{tr}}(z) : \mathrm{Id}_{C(y)}(g(\mathrm{tr}_\alpha(z)), \mathrm{tr}_\alpha(g(z)))$.

Proof. For (1), we apply path induction by assuming $z \equiv y$ and $\beta \equiv \mathrm{refl}_y$, so that it suffices to construct a term of $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), w : B(x) \vdash \mathrm{Id}_{B(y)}(\mathrm{tr}_{\mathrm{refl}_y}(\mathrm{tr}_\alpha(w)), \mathrm{tr}_{\mathrm{refl}_y \circ \alpha}(w))$ type. This is constructed by equating both terms to $\mathrm{tr}_\alpha(w)$ by combining Lemma 2.4.3.1 and Proposition 2.4.5.1. Indeed, we get the term $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), w : B(x) \vdash (\mathrm{ap}_{\mathrm{tr}_{(-)}(w)}(\mathrm{lun}_\alpha))^{-1} \circ \mathrm{refltr}(y, \mathrm{tr}_\alpha w) : \mathrm{Id}_{B(y)}(\mathrm{tr}_{\mathrm{refl}_y \circ \alpha}(w), \mathrm{tr}_{\mathrm{refl}_y}(\mathrm{tr}_\alpha(w)))$, and we conclude by path induction. (2) is the associated computation

rule. (3) is (1) with $\beta = \alpha^{-1}$ together with Proposition 2.4.6.4 and Lemma 2.4.3.1 applied on $\mathrm{tr}_{(-)}(z)$. For (4), by path induction we can assume all the equalities are the refltr terms from Proposition 2.4.5.1, and in such case $\mathrm{tr}_{\mathrm{refltr},\dots,\mathrm{refltr}}$ and $\mathrm{tr}_{\mathrm{refltr}^{-1},\dots,\mathrm{refltr}^{-1}}$ are both homotopic to the identical assignment. Finally, (5) follows once again by path induction for $y \equiv x$ and $\alpha \equiv \mathrm{refl}_x$, by means of $g(\mathrm{tr}_{\mathrm{refl}_x}(z)) = g(z) = \mathrm{tr}_{\mathrm{refl}_x}(g(z))$. □

Since composition of equalities is a particular case of transport, the above yields immediately:

Corollary 2.4.7.4. Composition of equalities is associative, as we can construct $\Gamma, x : A, y : A, z : A, w : A, \alpha : \mathrm{Id}_A(x,y), \beta : \mathrm{Id}_A(y,z), \gamma : \mathrm{Id}_A(z,w) \vdash \mathrm{ass}_{\alpha,\beta,\gamma} :\equiv \mathrm{coh}^{A;\mathrm{Id}_A(x,y)}_{\mathrm{tr},-\circ-}(\beta,\gamma,\alpha) : \mathrm{Id}_{\mathrm{Id}_A(x,w)}(\gamma\circ(\beta\circ\alpha),(\gamma\circ\beta)\circ\alpha)$.

We will translate the computation rules of Proposition 2.4.7.3 for the associativity once we discuss whiskering across equalities. Associativity of composition makes us deduce that it does not matter the order in which we compose a string of equalities, as we would strongly wish. Moreover, it allows us to write commutative diagrams of equalities with no ambiguity. In particular, these will have the extra feature that every equality is invertible, so that every side might be replaced with its inverse and maintain commutativity up to higher equality. Furthermore, in homotopy type theory, the above means that composition of morphisms in ∞-groupoids is associative as it should.

2.4.8 Horizontal composition across equalities

Another fundamental tool will be the ability to compose *horizontally* equalities between equalities, in the same way we do for natural transformation of functors or for 2-cells in any bicategory.

Remark 2.4.8.1. As it always happens for transport, as seen in the proof of Lemma 2.4.3.1, we may interpret composition of equalities as a function in either of its two arguments. Fix $\Gamma \vdash a, b, c : A$:

(1) For every $\Gamma \vdash \alpha : \mathrm{Id}_A(a,b)$ we have a *precomposition with* α function $\Gamma, z : A, \beta : \mathrm{Id}_A(b,c) \vdash (-\circ\alpha)(\beta) : \mathrm{Id}_A(x,z)$, assigning to every $\Gamma \vdash \beta : \mathrm{Id}_A(b,c)$ the equality $\Gamma \vdash \beta\circ\alpha : \mathrm{Id}_A(a,c)$.

(2) For every $\Gamma \vdash \beta : \mathrm{Id}_A(b,c)$ we have a *postcomposition with* β function $\Gamma, z : A, \alpha : \mathrm{Id}_A(a,b) \vdash (\beta\circ-)(\alpha) : \mathrm{Id}_A(x,z)$, assigning to every $\Gamma \vdash \alpha : \mathrm{Id}_A(b,c)$ the equality $\Gamma \vdash \beta\circ\alpha : \mathrm{Id}_A(a,c)$.

As a consequence, by Lemma 2.4.3.1, we inherit preservation of equalities in both variables in $\beta\circ\alpha$, by regarding composition as a function in α or in β. These are instances of the general phenomenon of horizontal composition of equalities, namely the whiskerings. Indeed, we saw that we can whisker equalities with functions of types, and here we claim that we can whisker higher order equalities with equalities of one level lower.

Proposition 2.4.8.2 (Horizontal composition of equalities)**.** Let $\Gamma \vdash A$ type. We can construct the following terms.

(1) *Right whiskering:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : A, \beta, \beta' : \mathrm{Id}_A(y,z), \tau : \mathrm{Id}_{\mathrm{Id}_A(y,z)}(\beta,\beta') \vdash \tau\alpha :\equiv \mathrm{ap}_{-\circ\alpha}(\tau) : \mathrm{Id}_{\mathrm{Id}_A(x,z)}(\beta\circ\alpha,\beta'\circ\alpha)$.

(2) *Left whiskering:* $\Gamma, x : A, y : A, \alpha, \alpha' : \mathrm{Id}_A(x,y), \sigma : \mathrm{Id}_{\mathrm{Id}_A(x,y)}(\alpha,\alpha'), z : A, \beta : \mathrm{Id}_A(y,z) \vdash \beta\sigma :\equiv \mathrm{ap}_{\beta\circ-}(\sigma) : \mathrm{Id}_{\mathrm{Id}_A(x,z)}(\beta\circ\alpha,\beta\circ\alpha')$.

(3) *Whiskering with* refl *on the right:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : A, \beta : \mathrm{Id}_A(y,z) \vdash \mathrm{comp}^{-\circ\alpha}(\beta) : \mathrm{Id}_{\mathrm{Id}_{\mathrm{Id}_A(x,z)}(\beta\circ\alpha,\beta\circ\alpha)}(\mathrm{refl}_\beta\,\alpha, \mathrm{refl}_{\beta\circ\alpha})$.

(4) *Whiskering with* refl *on the left:* $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : A, \beta : \mathrm{Id}_A(y,z) \vdash \mathrm{comp}^{\beta\circ-}(\alpha) : \mathrm{Id}_{\mathrm{Id}_{\mathrm{Id}_A(x,z)}(\beta\circ\alpha,\beta\circ\alpha)}(\beta\,\mathrm{refl}_\alpha, \mathrm{refl}_{\beta\circ\alpha})$.

(5) *Horizontal compositions:* $\Gamma, x : A, y : A, \alpha, \alpha' : \mathrm{Id}_A(x,y), \sigma : \mathrm{Id}_{\mathrm{Id}_A(x,y)}(\alpha,\alpha'), z : A, \beta, \beta' : \mathrm{Id}_A(y,z), \tau : \mathrm{Id}_{\mathrm{Id}_A(y,z)}(\beta,\beta') \vdash \mathrm{unihor}(\sigma,\tau) : \mathrm{Id}_{\mathrm{Id}_{\mathrm{Id}_A(x,z)}(\beta\circ\alpha,\beta'\circ\alpha')}(\beta'\sigma\circ\tau\alpha, \tau\alpha'\circ\beta\sigma)$.

Proof. (1), (2), (3) and (4) are induced as in the statement by Lemma 2.4.3.1, under the considerations in Remark 2.4.8.1. For (5), by path induction, we can assume $\beta \equiv \beta'$ and $\tau \equiv \mathrm{refl}_\beta$, so that it suffices to induce a term of $\Gamma, x : A, y : A, \alpha, \alpha' : \mathrm{Id}_A(x,y), \sigma : \mathrm{Id}_{\mathrm{Id}_A(x,y)}(\alpha,\alpha'), z : A, \beta : \mathrm{Id}_A(y,z) \vdash \mathrm{Id}_{\mathrm{Id}_{\mathrm{Id}_A(x,z)}(\beta\circ\alpha,\beta\circ\alpha')}(\beta\sigma\circ \mathrm{refl}_\beta\,\alpha, \mathrm{refl}_\beta\,\alpha'\circ\beta\sigma)$ type. The strategy is to show that they are both homotopic to $\beta\sigma$. Indeed, $\mathrm{refl}_\beta\,\alpha$ and

$\mathrm{refl}_\beta\ \alpha'$ are homotopic respectively to $\mathrm{refl}_{\beta\circ\alpha}$ and $\mathrm{refl}_{\beta\circ\alpha'}$ by (3) and (4). In particular, by whiskering as in (1) and (2), we get homotopies from $\beta\sigma\circ\mathrm{refl}_\beta\ \alpha$ and $\mathrm{refl}_\beta\ \alpha'\circ\beta\sigma$ to $\beta\sigma$ by unitality of composition in Proposition 2.4.6.1. Now, composing and inverting the appropriate equalities, we get an induced equality between $\beta\sigma\circ\mathrm{refl}_\beta\ \alpha$ and $\mathrm{refl}_\beta\ \alpha'\circ\beta\sigma$ as wished. By path induction, this induces the desired term. □

Proposition 2.4.8.2 allows us to define the *horizontal composition term* $\Gamma, x : A, y : A, \alpha, \alpha' : \mathrm{Id}_A(x,y), \sigma : \mathrm{Id}_{\mathrm{Id}_A(x,y)}(\alpha,\alpha'), z : A, \beta, \beta' : \mathrm{Id}_A(y,z), \tau : \mathrm{Id}_{\mathrm{Id}_A(y,z)}(\beta,\beta') \vdash \tau * \sigma :\equiv \beta'\sigma\circ\tau\alpha$, which expresses the full compatibility of composition with equality in both arguments. In particular, in (5), we showed that this term coincide with the other possible composition of whiskerings up to higher equality. Now we will make use of all the tools we have to prove various coherences satisfied by equalities. We can make use of horizontal composition of equalities to study another particular case of transport. Indeed, we saw that composition of equalities was defined as transport within an identity type. However, we are equally interested to studying the base changes of the identity types along a function of types. In particular, we can describe explicitly transport inside these base changes of the identity type in the most intuitive way we could expect.

Lemma 2.4.8.3 (Transport inside base change of the identity type)**.** Let $\Gamma \vdash A, B, C$ type with two functions of types $\Gamma.A \vdash f : B[p]$ and $\Gamma.C \vdash g : B[p]$. Then, the following types are inhabited:

(1) $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : C, \beta : \mathrm{Id}_B(f(x),g(z)) \vdash \mathrm{Id}_{\mathrm{Id}_B(f(y),g(z))}(\mathrm{tr}_\alpha(\beta), \beta\circ\mathrm{ap}_f(\alpha)^{-1})$ type.

(2) $\Gamma, x : A, z : C, w : C, \gamma : \mathrm{Id}_C(z,w), \beta : \mathrm{Id}_B(f(x),g(z)) \vdash \mathrm{Id}_{\mathrm{Id}_B(f(x),g(w))}(\mathrm{tr}_\gamma(\beta), \mathrm{ap}_g(\gamma)\circ\beta)$ type.

(3) $\Gamma, x, y : A, \alpha : \mathrm{Id}_A(x,y), z, w : C, \gamma : \mathrm{Id}_C(z,w), \beta : \mathrm{Id}_B(f(x),g(z)) \vdash \mathrm{Id}_{\mathrm{Id}_B(f(y),g(w))}(\mathrm{tr}_{\alpha,\gamma}(\beta), \mathrm{ap}_g(\gamma)\circ(\beta\circ\mathrm{ap}_f(\alpha)^{-1}))$ type.

Proof. Let us prove (1) via path induction, by assuming $y \equiv x$ and $\alpha \equiv \mathrm{refl}_x$. In that case, we can equate $\mathrm{tr}_{\mathrm{refl}_x}(\beta)$ to β, and $\mathrm{ap}_f(\mathrm{refl}_x)$ to $\mathrm{refl}_{f(x)}$. In particular, by Proposition 2.4.8.2 and Proposition 2.4.6.1, the latter induces a homotopy between $\beta\circ\mathrm{ap}_f(\mathrm{refl}_x)$ and β. By Proposition 2.4.4.1, this gives a homotopy in the opposite direction, and therefore, by Proposition 2.4.6.1, because they are both homotopic to β we induce an equality between $\mathrm{tr}_{\mathrm{refl}_x}(\beta)$ and $\beta\circ\mathrm{ap}_f(\mathrm{refl}_x)$. By path induction we conclude (1). The same argument, using the other unitality with respect to composition, yields (2). Finally, (3) follows from a double path induction and an analogous argument. □

2.4.9 Naturality of functions with respect to equalities

We want to extend the discussion of Remark 2.3.8.1, where we saw that functions of types are automatically functorial, and Lemma 2.4.3.1, where we saw that functions of types have a functorial action on equalities. In particular, we will now show that equalities of functions of types are automatically natural. More precisely, in Lemma 2.4.3.1 we saw that we can "apply" a function to an equality via the ap operator, and furthermore we can compose equalities, which makes us inherit a notion of commutative square of equalities. In this setting, we state and prove naturality in the most naive way as possible. In homotopy type theory, indeed, functions of types are regarded as functors of ∞-groupoids, and in order to interpret equalities of functors as natural transformation of functors, we need to show they are natural with respect to all equalities, that are, morphisms in ∞-groupoids. The same proof works in our setting as the theory is still agnostic about the types it is describing. However, it is true that later we will regard types as ∞-categories, and natural transformation of functors that will be *directed*. It will still be true, though, that equalities of terms will corresponds to *some* morphisms in a category, namely the isomorphisms, and that equalities of functions will correspond to *some* natural transformation of functors, namely the natural isomorphism. In particular, this result will inherit the meaning that natural isomorphisms of functors have a naturality square at every isomorphism.

Proposition 2.4.9.1 (Naturality of equalities)**.** Let $\Gamma \vdash A, B$ type together with two functions of types $\Gamma.A \vdash f, g : B[p]$ and a homotopy $\Gamma, a : A \vdash \sigma(a) : \mathrm{Id}_A(f(a),g(a))$ between them. Then we have a term $\Gamma, a : A, b : A, \alpha : \mathrm{Id}_A(a,b) \vdash \mathrm{nat}_{\sigma,\alpha}(a) : \mathrm{Id}_{\mathrm{Id}_B(f(a),g(b))}(\mathrm{ap}_g(\alpha)\circ\sigma(a), \sigma(b)\circ\mathrm{ap}_f(\alpha))$, which can display it via the

following commutative square of equalities

$$\begin{array}{ccc} f(a) & \xrightarrow{\sigma(a)} & g(a) \\ {\scriptstyle \mathrm{ap}_f(\alpha)}\downarrow & & \downarrow{\scriptstyle \mathrm{ap}_g(\alpha)} \\ f(b) & \xrightarrow[\sigma(b)]{} & g(b) \end{array}.$$

Proof. By path induction, we may assume that $b \equiv a$ and $\alpha \equiv \mathrm{refl}_a$. Thus, we need to provide a homotopy between $\mathrm{ap}_g(\mathrm{refl}_a) \circ \sigma(a)$ and $\sigma(a) \circ \mathrm{ap}_f(\mathrm{refl}_a)$. By Lemma 2.4.3.1 and Proposition 2.4.8.2, these are homotopic to $\mathrm{refl}_{g(a)} \circ \sigma(a)$ and $\sigma(a) \circ \mathrm{refl}_{f(a)}$ respectively, and thus they both equate to $\sigma(a)$ by the unitality terms from Proposition 2.4.6.1. Composing and inverting the equalities induces a homotopy in $\Gamma, a : A \vdash \mathrm{Id}_{\mathrm{Id}_A(f(a),g(a))}(\mathrm{ap}_g(\mathrm{refl}_a) \circ \sigma(a), \sigma(a) \circ \mathrm{ap}_f(\mathrm{refl}_a))$. The J-elimination rule gives the claim. □

Let us explore a consequence of this phenomenon. We saw that we promoted whiskering of equalities with lower equalities to a horizontal composition operation across equalities. Although they are of different nature, we would like to define a horizontal composition of equalities across functions out of the whiskerings of equalities with functions which we defined. More explicitly, whenever we have two equalities α and β as above, we can compose the two whiskerings to get an equality $\Gamma, x : A \vdash (\beta * \alpha)(x) : \mathrm{Id}_C((g \circ f)(x), (g' \circ f')(x))$ through a combination of whiskerings of equalities with functions. This tells us that composition of functions of types is compatible with homotopy. The fact that this construction is well-defined, in the sense that the combination of whiskerings we choose does not matter, is a consequence of Proposition 2.4.9.1.

Corollary 2.4.9.2 (Horizontal composition across functions)**.** Let $\Gamma \vdash A, B, C$ type, and consider functions $\Gamma.A \vdash f, f' : B[p]$ and $\Gamma.B \vdash g, g' : C[p]$. Assume we have pointwise homotopies $\Gamma, a : A \vdash \beta(a) : \mathrm{Id}_B(f(a), f'(a))$ and $\Gamma, b : B \vdash \gamma(b) : \mathrm{Id}_C(g(b), g'(b))$. Then, there is a term $\Gamma, a : A \vdash \mathrm{unihor}_{\mathrm{fun}}(\beta, \gamma)(a) :\equiv \mathrm{nat}_{\gamma,\beta(a)} : \mathrm{Id}_{\mathrm{Id}_C(g(f(a)),g'(f'(a)))}(\mathrm{ap}_{g'}(\beta(a)) \circ \gamma(f(a)), \gamma(f'(a)) \circ \mathrm{ap}_g(\beta(a)))$, which we can display via the following commutative square of equalities:

$$\begin{array}{ccc} g(f(a)) & \xrightarrow{\gamma(f(a))} & g'(f(a)) \\ {\scriptstyle \mathrm{ap}_g(\beta(a))}\downarrow & & \downarrow{\scriptstyle \mathrm{ap}_{g'}(\beta(a))} \\ g(f'(a)) & \xrightarrow[\gamma(f'(a))]{} & g'(f'(a)) \end{array}.$$

Proof. It is a base change of Proposition 2.4.9.1 as in the claim. □

Corollary 2.4.9.2 allows us to deduce that composition of functions is fully compatible with equality in a unique way. In other words, we can define the *horizontal composition of* β *and* γ (as in the statement) to be $\Gamma, x : A \vdash (\beta * \alpha)(x) :\equiv \mathrm{ap}_g(\beta(a)) \circ \beta(f(a)) : \mathrm{Id}_C((g \circ f)(x), (g' \circ f')(x))$, as we proved that the other possible combination of whiskering would coincide with it.

2.4.10 Pentagon non-axiom for equalities

To convince the reader that this theory possesses enough coherences, we will prove some desired ones, ending up by giving an insight on the pentagon for composition of equalities. Indeed, since, by Corollary 2.4.7.4, composition of equalities is non-strictly associative, we can wonder if the constructed associativity terms satisfy a pentagon coherence when we compose four equalities in a row. Before that, we need to prove some technical lemmas.

Lemma 2.4.10.1. Let $\Gamma \vdash A$ type. The following squares of equalities commute over $\Gamma, a : A, b : A, \alpha : \mathrm{Id}_A(a,b), \beta : \mathrm{Id}_A(a,b), \sigma : \mathrm{Id}_{\mathrm{Id}_A(a,b)}(\alpha,\beta)$

$$\begin{array}{ccc} \alpha \circ \mathrm{refl}_a & \xrightarrow{\mathrm{run}_\alpha} & \alpha \\ {\scriptstyle \sigma\,\mathrm{refl}_a}\downarrow & & \downarrow{\scriptstyle \sigma} \\ \beta \circ \mathrm{refl}_a & \xrightarrow[\mathrm{run}_\beta]{} & \beta \end{array} \qquad \begin{array}{ccc} \mathrm{refl}_b \circ \alpha & \xrightarrow{\mathrm{lun}_\alpha} & \alpha \\ {\scriptstyle \mathrm{refl}_b\,\sigma}\downarrow & & \downarrow{\scriptstyle \sigma} \\ \mathrm{refl}_b \circ \beta & \xrightarrow[\mathrm{lun}_\beta]{} & \beta \end{array}.$$

Proof. Let us show that the left square commutes. By path induction on the identity type of $\mathrm{Id}_A(a,b)$ we can assume $\beta \equiv \alpha$ and $\sigma \equiv \mathrm{refl}_\alpha$. In this case both composites are homotopic to run_α: by computation rule of the whiskering in Proposition 2.4.8.2 we get $\mathrm{comp}^{-\circ \mathrm{refl}_a}(\mathrm{refl}_\alpha) : \mathrm{Id}_{\mathrm{Id}_{\mathrm{Id}_A(a,b)}(\alpha\circ\mathrm{refl}_a, \alpha\circ\mathrm{refl}_a)}(\mathrm{refl}_\alpha\,\mathrm{refl}_a, \mathrm{refl}_{\alpha\circ\mathrm{refl}_a})$, and the claim follows by the left unitality of run_α and right unitality of run_β. The same argument gives the right square. □

In particular, when in the left square we consider α to be of the form $\mathrm{refl}_b \circ \alpha'$ and β of the form α' respectively, with σ being the unital term $\mathrm{lun}_{\alpha'} : \mathrm{Id}_{\mathrm{Id}_A(a,b)}(\mathrm{refl}_b \circ \alpha', \alpha')$, and dually for the right square, we conclude that the left and right unital equality terms "commute". Now we study in detail this case.

Lemma 2.4.10.2 (Computation of the associativity term)**.** Let $\Gamma \vdash A$ type. The associativity term fits into the following commutative triangle of equalities over $\Gamma, a : A, b : A, c : A, \alpha : \mathrm{Id}_A(a,b), \beta : \mathrm{Id}_A(b,c)$:

$$\begin{array}{ccc} \mathrm{refl}_b \circ (\beta \circ \alpha) & \xrightarrow{\mathrm{lun}_{\beta\circ\alpha}} & \beta \circ \alpha \\ & {\scriptstyle \mathrm{ass}_{\alpha,\beta,\mathrm{refl}_b}}\searrow & \downarrow{\scriptstyle \mathrm{lun}_\beta^{-1}\,\alpha} \\ & & (\mathrm{refl}_b \circ \beta) \circ \alpha \end{array}$$

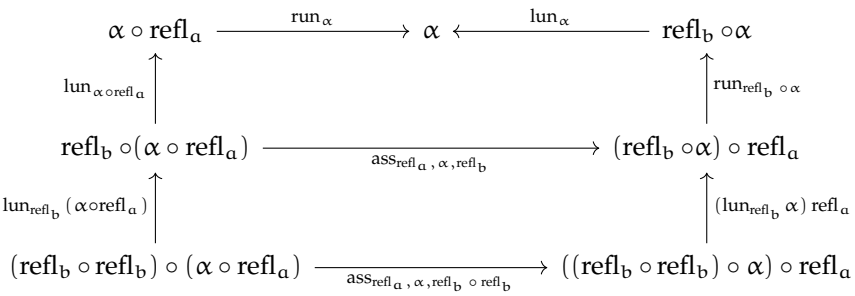
Furthermore, when both the external equalities are refl, we can decompose the associativity term into a sequence of unitalities through the following commutative diagram of equalities over $\Gamma, a : A, b : A, \alpha : \mathrm{Id}_A(a,b)$

$$\begin{array}{ccccc} \alpha \circ \mathrm{refl}_a & \xrightarrow{\mathrm{run}_\alpha} & \alpha & \xrightarrow{\mathrm{lun}_\alpha^{-1}} & \mathrm{refl}_b \circ \alpha \\ {\scriptstyle \mathrm{lun}_{\alpha\circ\mathrm{refl}_a}}\uparrow & \searrow^{\mathrm{lun}_\alpha^{-1}\,\mathrm{refl}_a} & & & \downarrow{\scriptstyle \mathrm{run}_{\mathrm{refl}_b\circ\alpha}} \\ \mathrm{refl}_b \circ (\alpha \circ \mathrm{refl}_a) & & \xrightarrow[\mathrm{ass}_{\mathrm{refl}_a,\alpha,\mathrm{refl}_b}]{} & & (\mathrm{refl}_b \circ \alpha) \circ \mathrm{refl}_a \end{array}.$$

Proof. The commutativity of the triangle is just the translation of (2) in Proposition 2.4.7.3 into the case of the associativity term, where transport is composition of equalities. Indeed, by definition of composition, unitality and whiskering, that term can be rewritten through $\mathrm{refltr}(z, \mathrm{tr}_\alpha) \equiv \mathrm{lun}_{\beta\circ\alpha}$ and $(\mathrm{ap}_{\mathrm{tr}_{(-)}(\alpha)(\beta)}(\mathrm{lun}_\beta))^{-1} \equiv (\mathrm{lun}_\beta\,\alpha)^{-1} = \mathrm{lun}_\beta^{-1}\,\alpha$. The last homotopy follows by Lemma 2.4.7.1, from which whiskering, being a function of types, is compatible with inversion. The last diagram is a pasting of the triangle above with the appropriate square in Lemma 2.4.10.1. □

Remark 2.4.10.3. Lemma 2.4.10.2 allows us to decompose the associativity term of some $\alpha : \mathrm{Id}_A(a,b)$, with refl_a and refl_b on the right and on the left, into a sequence of unitalities. However, we would like to compute the case where refl_b is replaced with another loop homotopic to it: in the pentagon axiom we will deal with associativity terms with respect to chains of reflexive terms $\mathrm{refl}_b \circ \mathrm{refl}_b$. This can be done in general, since by Lemma 2.4.5.2 we know that dependent functions, in this case $l : \mathrm{Id}_A(b,b) \vdash \mathrm{ass}_{\mathrm{refl}_a,\alpha,l} : \mathrm{Id}_{\mathrm{Id}_A(a,b)}(l \circ (\alpha \circ \mathrm{refl}_a), (l \circ \alpha) \circ \mathrm{refl}_a)$, are compatible with equalities. This means that, for every $\sigma : \mathrm{Id}_{\mathrm{Id}_A(b,b)}(\mathrm{refl}_b, l)$, the associativity term $\mathrm{ass}_{\mathrm{refl}_a,\alpha,l}$ is homotopic to the transport $\mathrm{tr}_\sigma(\mathrm{ass}_{\mathrm{refl}_a,\alpha,\mathrm{refl}_b})$. Now, by Lemma 2.4.8.3, we know that we can describe transport inside the identity type $\mathrm{Id}_{\mathrm{Id}_A(a,b)}(l \circ (\alpha \circ \mathrm{refl}_a), (l \circ \alpha) \circ \mathrm{refl}_a)$ as a composition. More precisely, it will be the postcomposition with $\mathrm{ap}_{(-\circ\alpha)\circ\mathrm{refl}_a}(\sigma)$, which is homotopic to $\mathrm{ap}_{-\circ\mathrm{refl}_a}(\mathrm{ap}_{(-\circ\alpha)}(\sigma))$, and the precomposition with $\mathrm{ap}_{-\circ(\alpha\circ\mathrm{refl}_a)}(\sigma)$. These are respectively the iterated whiskering $(\sigma\alpha)\,\mathrm{refl}_a$ and the whiskering $\sigma(\alpha \circ \alpha)$ by definition. We conclude that we can decompose

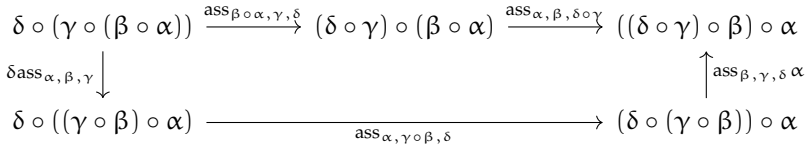
$\mathrm{ass}_{\mathrm{refl}_a,\alpha,l}$ as in Lemma 2.4.10.2 up to composing appropriately with the whiskerings of σ. For instance, if $l \equiv \mathrm{refl}_b \circ \mathrm{refl}_b$ and $\sigma \equiv \mathrm{lun}_{\mathrm{refl}_b}$, we can decompose $\mathrm{ass}_{\mathrm{refl}_a,\alpha,\mathrm{refl}_b \circ \mathrm{refl}_b}$ as follows



where the upper square is provided by Lemma 2.4.10.2 and the lower square by the discussion above. An analogous procedure works for $\mathrm{ass}_{\mathrm{refl}_a \circ \mathrm{refl}_a,\alpha,\mathrm{refl}_b}$.

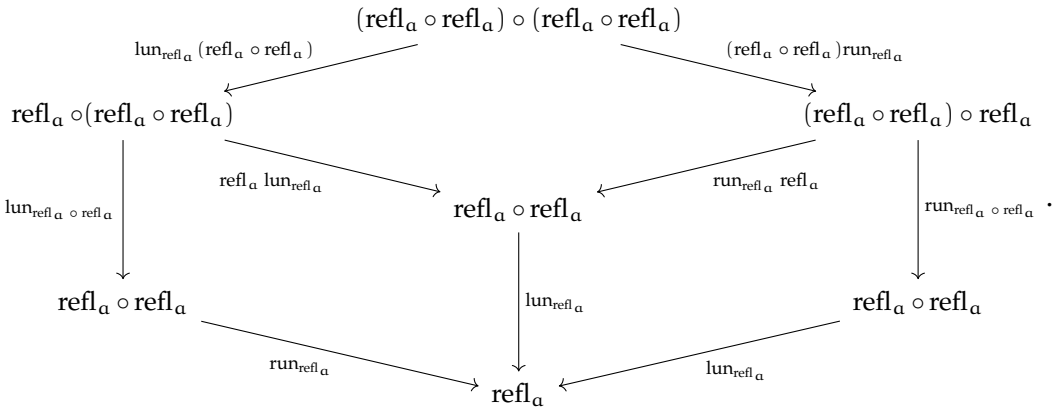
Proposition 2.4.10.4 (Pentagon non-axiom for equalities)**.** Composition of equalities satisfies the pentagon coherence.

Proof. Consider $\Gamma \vdash A$ type and $a, b, c, d, e : A$ with four homotopies $\alpha : \mathrm{Id}_A(a, b)$, $\beta : \mathrm{Id}_A(b, c)$, $\gamma : \mathrm{Id}_A(c, d)$, $\delta : \mathrm{Id}_A(d, e)$. To show the pentagon axiom, we need to show that the pentagon



commutes. By iterated path induction, we may assume that $d \equiv c \equiv b \equiv a$ and $\delta \equiv \gamma \equiv \beta \equiv \alpha \equiv \mathrm{refl}_a$, so that every vertex of the pentagon is a composite of reflexive terms, and all arrows are (whiskerings of) associativity terms $\mathrm{ass}_{\rho,\sigma,\tau}$ with ρ and τ being refl_a or $\mathrm{refl}_a \circ \mathrm{refl}_a$. The strategy is to decompose all the associativity terms on the sides of the pentagon into sequences of unitality terms and their whiskerings by Lemma 2.4.10.2 and Remark 2.4.10.3. The first will be enough for associativity terms of the form $\mathrm{ass}_{\mathrm{refl}_a,\sigma,\mathrm{refl}_a}$, the latter for all the other ones when a composite $\mathrm{refl}_a \circ \mathrm{refl}_a$ appears instead of refl_a. In particular, in Remark 2.4.10.3, we saw an explicit decomposition of the term $\mathrm{ass}_{\mathrm{refl}_a,\mathrm{refl}_a,\mathrm{refl}_a \circ \mathrm{refl}_a}$, and this can be done for all sides of the pentagon. However, the diagrams we construct over each side do not automatically provide a filling of the whole pentagon, as even their vertices can differ a priori, some being of the form refl_a and others $\mathrm{refl}_a \circ \mathrm{refl}_a$. In other words, the constructed diagrams do not paste with eachother, and therefore we need to fill the interior with further commutative diagrams. For this purpose, we equate all the vertices to refl_a, so that we split the interior into five closed diagrams, one over each vertex. Proving their commutativity is computationally heavy, but we will give an idea at least for the upper-middle vertex $(\mathrm{refl}_a \circ \mathrm{refl}_a) \circ (\mathrm{refl}_a \circ \mathrm{refl}_a)$. The two chains of equalities from this to refl_a come from the decompositions of $\mathrm{ass}_{\mathrm{refl}_a,\mathrm{refl}_a,\mathrm{refl}_\circ \mathrm{refl}_a}$, which we made explicit in Remark 2.4.10.3, and of $\mathrm{ass}_{\mathrm{refl}_a \circ \mathrm{refl}_a,\mathrm{refl}_a,\mathrm{refl}_a}$, which has an

analogous decomposition. We obtain the following outer diagram, which we subdivide as in the picture:



The upper square commutes by uniqueness of the horizontal composite from Proposition 2.4.8.2: we are equating $(\mathrm{refl}_a \circ \mathrm{refl}_a) \circ (\mathrm{refl}_a \circ \mathrm{refl}_a)$ to $\mathrm{refl}_a \circ \mathrm{refl}_a$ by using unitalities on both brackets, and we showed that concatenating two whiskerings does not depend on the order. The left square commutes by commutativity of left and right unitality seen in Lemma 2.4.10.1 in the case σ is a unitality term. The same applies to the right square, together with the fact that we have an equality between $\mathrm{lun}_{\mathrm{refl}_a}$ and $\mathrm{run}_{\mathrm{refl}_a}$ by Proposition 2.4.6.1. Analogous arguments apply for each vertex: for whiskerings of associativity terms, it will be useful to note that whiskering is compatible with composition of equality being a function of types by Lemma 2.4.7.1, being a function of types. In the end, we obtain a filling for the whole pentagon. □

The reader could have fun in proving the further coherences he can think of with the tools we provided. Now, we focus our attention on two special classes of function of types: equivalences and embeddings.

2.4.11 Equivalences of types

The intensional identity type introduces the notion of homotopy into our setting, which we can exploit to define an equivalence of two types over some context. When we will regard types as categories, we will make sure that equalities between (generalised) terms will represent natural isomorphisms, and then the notion of equivalence of types will represent that of equivalence of categories. First of all, we will split the condition of being an equivalence in the conjunction of that of having a section and a retraction.

Definition 2.4.11.1 (Section and retraction of types)**.** Let $\Gamma \vdash A, B$ type, and consider a function of types $\Gamma.A \vdash f : B[p]$.

(1) A *section of* f is a function of types $\Gamma.A \vdash h : B[p]$ together with an equality $\Gamma.B \vdash \mathrm{Id}_{B[p]}(f \circ h, q)$.

(2) A *retraction of* f is a function of types $\Gamma.A \vdash g : B[p]$ together with an equality $\Gamma.A \vdash \mathrm{Id}_{A[p]}(g \circ f, q)$.

Definition 2.4.11.2 (Equivalence of types)**.** Let $\vdash \Gamma\,\mathrm{ctx}$ and let $\Gamma \vdash A, B$ type. An *equivalence from* A *to* B is a function of types $\Gamma.A \vdash f : B[p]$ together with a section (right inverse) and a retraction (left inverse).

A first good think to check is that these concepts are compatible with base change.

Lemma 2.4.11.3. Let $\Gamma \vdash A, B$ type and let $\Gamma.A \vdash f : B[p]$ be a function. Consider a context substitution $\vdash \gamma : \Delta \to \Gamma\,\mathrm{ctx}$.

(1) If $\Gamma.B \vdash g : A[p]$ is a retraction of f, then $\Delta.B[\gamma] \vdash g[\gamma] : A[\gamma][p]$ is a retraction of $\Gamma.A[\gamma] \vdash f[\gamma] : B[\gamma][p]$.

(2) If $\Gamma.B \vdash h : A[p]$ is a section of f, then $\Delta.B[\gamma] \vdash h[\gamma] : A[\gamma][p]$ is a section of $\Gamma.A[\gamma] \vdash f[\gamma] : B[\gamma][p]$.

(3) If f is an equivalence over Γ, then $\Delta.A[\gamma] \vdash f[\gamma] : B[\gamma]$ is an equivalence over Δ, with left and right inverses given by base change of the left and right inverses of f.

Proof. (3) is a consequence of (1) and (2). For (1), we have a homotopy $\Gamma.A \vdash \alpha : \mathrm{Id}_{A[p]}(g \circ f, q)$, and base change yields $\Delta.A[\gamma] \vdash \alpha[\gamma.A] : \mathrm{Id}_{A[\gamma][p]}(g[\gamma] \circ f[\gamma], q)$, thus the claim. (2) is analogous. □

Equivalences of types clearly enjoy various familiar properties inherited by equivalences and isomorphisms in ordinary category theory. Without giving explicit argument, which are just the translation of the classical ones, let us mention some of the key properties.

Remark 2.4.11.4. With the standard argument, one can prove that if $f : A \to B$ over Γ is an equivalence, for every $g : B \to A$ over Γ we have that $g \circ f$ is homotopic to 1_A if and only if $f \circ g$ is homotopic to 1_B. We call g an *inverse* of f. In particular, every retraction or section of f is homotopic to g.

Lemma 2.4.11.5. Equivalences of types enjoy the 2-out-of-3 and 2-out-of-6 properties.

Now it is time to provide the first example of equivalence which we have already encountered. Indeed, we will now prove that transport over equal terms is an equivalence between the fibres:

Proposition 2.4.11.6. Let $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. The function of types $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), z : B(x) \vdash \mathrm{tr}_\alpha(z) : B(y)$ is an equivalence of types from $B(x)$ to $B(y)$ over $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y)$.

In particular, $\Gamma, x : A, y : A, \sigma : \mathrm{Id}_A(x,y) \vdash \alpha'(x)^{-1}(\sigma \circ \alpha(x))\,\mathrm{Id}_A(h(x), h'(y))$ is an equivalence of types.

Proof. By Proposition 2.4.7.3, $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y), w : B(y) \vdash \mathrm{tr}_{\alpha^{-1}}(w) : B(x)$ is an inverse. □

A crucial point in our theory is that we want all the type constructors to be compatible with equivalences, in the same way categorical constructions are invariant under equivalences. First of all, let us check this property holds for the intensional identity type.

Proposition 2.4.11.7 (Id is compatible with equivalences)**.** Let $\Gamma \vdash A, B$ type, and let $\Gamma.A \vdash f : B[p]$ be an equivalence of types. Then, the function $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{ap}_f(\alpha) : \mathrm{Id}_B(f(x), f(y))$ of Lemma 2.4.3.1 is an equivalence of types from Id_A to $f^*\,\mathrm{Id}_B$ over $\Gamma.A.A[p]$.

Proof. Let $\Gamma.B \vdash g : A[p]$ be a retraction of f under a homotopy $\Gamma, a : A \vdash r(a) : \mathrm{Id}_A(g(f(a)), a)$. This induces a function $\Gamma, x : A, y : A, \beta : \mathrm{Id}_B(f(x), f(y)) \vdash \mathrm{ap}_g(\beta) : \mathrm{Id}_A(g(f(x)), g(f(y)))$. Using that g is a retraction of f, by Proposition 2.4.11.6 we have an equivalence $\mathrm{Id}_A(g(f(x)), g(f(y))) \simeq \mathrm{Id}_A(x,y)$, that gives rise to $\Gamma, x : A, y : A, \beta : \mathrm{Id}_B(f(x), f(y)) \vdash r(y) \circ (\mathrm{ap}_g(\beta) \circ r(x)^{-1}) : \mathrm{Id}_A(x,y)$. We need to check that this is a retraction of ap_f, i.e. that the composite function $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash r(y) \circ (\mathrm{ap}_g(\mathrm{ap}_f(\alpha)) \circ r(x)^{-1}) : \mathrm{Id}_A(x,y)$ is homotopic to the identical assignment. For this purpose, the J-rule allows us to assume $y \equiv x$ and $\alpha \equiv \mathrm{refl}_x$. In particular, by Lemma 2.4.3.1 and Lemma 2.4.7.1, we can equate $\mathrm{ap}_g(\mathrm{ap}_f(\mathrm{refl}_x))$ to $\mathrm{refl}_{g(f(x))}$ up to homotopy, and the claim follows by $r(x) \circ (\mathrm{refl}_{g(f(x))} \circ r(x)^{-1})$ being homotopic to refl_x in $\mathrm{Id}_A(x,x)$. An analogous argument works for a section of f, and thus we conclude the claim. □

This result means that if two types are equivalent, then so are their (higher) identity types. This means that two equivalent type also share the same notion of homotopy under the identification function. Furthermore, we could interpret this by saying that equivalences of types reflect equalities.

Remark 2.4.11.8 (Logical equivalence and equivalence as types)**.** In homotopy type theory, under the interpretation of types as statements, we defined what it means for two types to be logically equivalent. However, now that we have a concept of equivalence of types, we are interested in comparing the two notions. Recall that, given $\Gamma \vdash A, B$ type, a logical equivalence between them is the datum of two functions $A \to B$ and $B \to A$, as giving a proof of A determines one of B and conversely. However, the type-interpretation of statements is deeper, since being equivalent as types always asks for the two functions of a logical equivalence to be inverse to eachother. This is surely a stronger conditions (for instance, two constant functions could provide a logical equivalence but not an equivalence between types). In other words, logical equivalence between two types represents the equivalence between two meta-theoretical definitions of the types being inhabited, whereas an equivalence as types is its fully internal translation. In particular, the latter is the kind of equivalence we look for.

2.4.12 Meta Yoneda Lemma

This paragraph will be devoted to understanding what we will call *meta Yoneda Lemma*. The idea is to unlock the ability to provide equivalences between types by knowing that they have the same terms. This is something that is intrinsic in the type-theoretical formalism, and it can surprisingly be interpreted as a meta-theoretical formulation of the Yoneda Lemma in type theory. In homotopy type theory, this result is constantly used implicitly, but we will see how we will need to be more careful in our setting, for functoriality reasons linked to Remark 2.3.8.3. Arguably, for a type theorist, the most interesting aspect of this paragraph will be to notice how this result will *not* work without some assumptions, ending into a change in the philosophy from homotopy type theory. The starting idea is that, to provide an equivalence of types from A to B over Γ, we need a function $\Gamma, x : A \vdash f(x) : B$, which has to be functorial a priori in the sense of Remark 2.3.8.1. In other words, it must be instrinsically an assignment on generalised terms. However, we will often have to do with type constructors which are compatible with base change, therefore we will be able to turn an assignment $\Gamma \vdash a : A \mapsto \Gamma \vdash f(a) : B$ into a functorial assignment syntactically, and a mere bijection of terms up to homotopy will provide an equivalence.

Theorem 2.4.12.1 (Meta Yoneda Lemma)**.** Let $\vdash \Gamma$ ctx, and $\Gamma \vdash X, Y$ type. Assume that we have rules

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash x : X[p]}{\Gamma.A \vdash f_A(x) : Y[p]} \qquad \frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash y : Y[p]}{\Gamma.A \vdash g_A(y) : X[p]}$$

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash x : X[p]}{\Gamma.A \vdash l_A : \mathrm{Id}_{X[p]}(g_A(f_A(x)), x)} \qquad \frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash y : Y[p]}{\Gamma.A \vdash r_A : \mathrm{Id}_{Y[p]}(f_A(g_A(y)), y)}$$

with f_A, g_A, l_A and r_A compatible with base change under functions of types over Γ. Then, $f_X(q)$ and $g_X(q)$ provide an equivalence of types between X and Y.

Proof. We can apply the rules for the contexts $\Gamma.X$ and $\Gamma.Y$ and the universal terms $\Gamma.X \vdash q : X[p]$ and $\Gamma.Y \vdash q : Y[p]$. These yield functions $\Gamma.X \vdash f_X(q) : Y[p]$ and $\Gamma.X \vdash g_Y(q) : X[p]$. Note that $g_Y(q) \circ f_X(q) \equiv g_X(q)[p.f_X(q)] \equiv g_X(f_X(q))$ which is equal to the the identity term q via $\Gamma.X \vdash l_X : \mathrm{Id}_{X[p]}(g_X(f_X(q)), q)$. The same argument holds for the other composite. We conclude that $f_X(q)$ and $g_Y(q)$ provide an equivalence of types with equalities l_X and r_Y. □

This argument says that we can prove equivalences of type constructors in a very naive way by providing a *homotopy bijection on terms*, i.e. a bijection up to homotopy, which is compatible with base change. However, the price we pay is that we need to require these rules to hold for any arbitrary context extension of Γ. In particular, when we have type constructors that are defined over all context extensions of some Γ, a naive proof of bijection of terms translates syntactically into a proof in every context extension, as in the statement, and thus provides an equivalence by Meta Yoneda Lemma. This is not surprising: we saw that the difference between having a variable $x : A$ in the context and having a term $a : A$ is the implicit functoriality of the first, meaning that x stands for all terms of the possible weakenings of A. This is overcome by asking the rules to hold in all the context extensions, so that a naive termwise proof reasonably yields a functorial proof.

Remark 2.4.12.2. The careful reader will have noticed that Theorem 2.4.12.1 resembles the Yoneda Lemma. Indeed, it states that if two types $\Gamma \vdash A, B$ type are such that their generalised terms are in bijection up to homotopy, compatibly with base change, then they are equivalent. This means that if the functions of types $S \to A$ are in bijection with the functions of types $S \to B$ compatibly with composition, then A and B are equivalent types. Therefore, it obviously mimics the Yoneda Lemma for the homotopy classes of maps.

The ability of showing equivalences with almost set-theoretical proofs, consisting of bijections of terms, will be a very important feature in our setting, as much as Yoneda Lemma is in category theory. On the other hand, later we will deal with type constructors which are not defined over all contexts, e.g. only over animae. In particular, their rules will not tell us all their generalised terms, and Theorem 2.4.12.1 will not apply. This is in strong contrast with homotopy type theory as observed in Remark 2.3.8.3, since this is a theory about statements where everything is automatically functorial. Indeed, there, we never have any difference between an equivalence of types and a homotopy bijection of terms, as the latter is functorial

by default because the rules are the same in all contexts. In some sense, in homotopy type theory, every type looks like a set, a "collection", among which it is very easy to prove bijections. Instead, the fact that a homotopy bijection of terms will not automatically lift to an equivalence of types is peculiar of our theory. Furthermore, as we will observe in detail, this will emphasise the importance in our theory of animae and our interest into them. In particular, the invariance of rules over animae will allow us to interpret animae as "collections" in the above sense.

An interesting aspect of the proof is that we did not really make use of the rules over *any arbitrary* context extension of Γ with a type. The same argument would in fact work when the rules hold for the two context extensions $\Gamma.X$ and $\Gamma.Y$. Therefore, in this assumption, we would keep the ability of showing equivalences with ease. This will be discussed in detail in the case of animae.

2.4.13 The homotopy category of types

We introduced an equivalence relation on terms of types, and thus on functions of types, which is compatible with the composition operation. Thus, we can take the quotient of functions over this relation. What we get is an ordinary category, as all the homotopy data are forced to become boolean truth values.

Definition 2.4.13.1. Given $\vdash \Gamma\,\mathrm{ctx}$, the *homotopy category in context* Γ, $\mathrm{Ho}(\mathrm{Type}_{/\Gamma})$, is the ordinary category whose objects are types in context Γ and whose arrows are homotopy classes of functions of types over Γ. We denote with $\mathrm{Ho}(\mathrm{Type})$ the homotopy category of types over the empty context.

Remark 2.4.13.2. Note that isomorphisms in $\mathrm{Ho}(\mathrm{Type})$ are equivalences of types. In particular, Meta Yoneda Lemma is the Yoneda Lemma in the homotopy ordinary category $\mathrm{Ho}(\mathrm{Type})$. Indeed, the reader should not be surprised by the importance this result will have in the theory.

We could assume not to know category theory in this setting. However, we have an example of an ordinary category in our hands, and all the theorems of category theory will have a proof in this specific example with the usual argument. This means that it will not be restrictive to assume them.

2.4.14 Equivalences of contexts

We introduced an identification on types over a context Γ: equivalences of types. Since we can extend Γ with types, we can wonder how the obtained contexts are related. This leads us to think that there is a reasonable way to identify contexts homotopically. Instead of mimicking the usual definition of equivalence with context substitutions, however, we will characterise them with an induction principle. Equivalent contexts should be contexts with "the same types" over them.

Definition 2.4.14.1 (Equivalence of contexts)**.** Let $\vdash \Gamma\,\mathrm{ctx}$ and let $\vdash \Gamma.A :\equiv \Gamma.A_1 \dots A_m, \Gamma.B :\equiv \Gamma.B_1 \dots, B_n\,\mathrm{ctx}$ be two chains of dependent types over it. An *equivalence of contexts over* Γ from $\Gamma.A_1 \dots A_m$ to $\Gamma.B_1 \dots B_n$, denoted with $\Gamma.A \simeq \Gamma.B$, is a context substitution $\gamma : \Gamma.A \to \Gamma.B$ such that $p^n \circ \gamma = p^m$ (i.e. *over* Γ) equipped with two rules, called respectively *induction principle* and *computation rule*,

$$\frac{\Gamma.B_1 \dots B_n \vdash Q\,\mathrm{type} \quad \Gamma.A_1 \dots A_m \vdash t : Q[\gamma]}{\Gamma.B_1 \dots B_n \vdash \mathrm{ind}(t) : Q} \qquad \frac{\Gamma.B_1 \dots B_n \vdash Q\,\mathrm{type} \quad \Gamma.A_1 \dots A_m \vdash t : Q[\gamma]}{\Gamma.A_1 \dots A_m \vdash \mathrm{comp}(t) : \mathrm{Id}_{Q[\gamma]}(\mathrm{ind}(t)[\gamma], t)},$$

together with further rules expressing the compatibility with base change along substitutions $\Delta \to \Gamma$.

Remark 2.4.14.2. Let $\Gamma \vdash A, B\,\mathrm{type}$, and let $\Gamma.A \vdash f : B[p]$ be a function of types from A to B. Then, f is an equivalence of types over Γ if and only if $p.f : \Gamma.A \to \Gamma.B$ is an equivalence of contexts.

The reader might expect an equivalence of contexts to appear as the datum of two substitutions in opposite directions that satisfy a symmetric property. Instead, the given rules are asymmetric, and seem to define a directed notion: it might look like the characterisation of a one-sided inverse. It turns out that the two rules play dual roles to make the definition equivalent to a more familiar, symmetric, definition. First of all, let us define the concept of homotopic substitutions, which will be its central idea.

Remark 2.4.14.3. Given two types $\Gamma \vdash A, B$ type, context substitutions $\Gamma.A \to \Gamma.B$ that commute with p are the same as terms $A \vdash f : B[p]$, and this comes equipped with a notion of homotopy. In the same way, given $\vdash \Gamma$ ctx and $\vdash \Gamma.A_1 \dots A_m, \Gamma.B_1 \dots B_n$ ctx, we can consider the types $\Gamma.A_1 \dots A_m.B_1[p^m] \dots B_{i-1}[p^m] \vdash B_i[p^m]$ type for every $i \in \{1, \dots, n\}$. Then, a context substitution $\gamma : \Gamma.A_1 \dots A_m \to \Gamma.B_1 \dots B_n$ with $p^n \circ \gamma = p^m$ is the same as an n-tuple of terms $\Gamma.A_1 \dots A_m \vdash \gamma_i : B_i[p^m.\gamma_1 \dots \gamma_{i-1}]$ by iterating the argument with types, with $\gamma_i :\equiv q[p^{n-i+1}][\gamma]$, i.e. the action of γ on the i-th variable. For another context substitution $\delta : \Gamma.C_1 \dots C_l \to \Gamma.A_1 \dots A_m$ with $p^m \circ \delta = p^l$, the composite substitution $\gamma \circ \delta$ corresponds of the n-tuple $\Gamma.C_1 \dots C_l \vdash \gamma_i[\delta] : B_i[p^l.\gamma_1[\delta] \dots \gamma_{i-1}[\delta]]$.

Comparing two context substitutions $\Gamma.A \to \Gamma.B$ over Γ is particularly simple, as we can directly compare them as term of $A \vdash B[p]$ type. For more general substitutions we can iterate this process and define two substitutions $\gamma, \gamma' : \Gamma.A_1 \dots A_m \to \Gamma.B_1 \dots B_n$ over Γ to be *homotopic*, written $\gamma \simeq \gamma'$, if, by considering their representations as n-tuples of terms, we get a chain of equality terms

$$\begin{aligned}
&\Gamma.A_1 \dots A_m \vdash \alpha_1 : \mathrm{Id}_{B_1[p^m]}(\gamma_1, \gamma_1'),\\
&\Gamma.A_1 \dots A_m \vdash \alpha_2 : \mathrm{Id}_{B_2[p^m.\gamma_1']}(\mathrm{tr}_{\alpha_1}(\gamma_2), \gamma_2'),\\
&\Gamma.A_1 \dots A_m \vdash \alpha_3 : \mathrm{Id}_{B_3[p^m.\gamma_1'.\gamma_2']}(\mathrm{tr}_{\alpha_1,\alpha_2}(\gamma_3), \gamma_3'),\\
&\vdots\\
&\Gamma.A_1 \dots A_m \vdash \alpha_n : \mathrm{Id}_{B_n[p^m.\gamma_1' \dots \gamma_{n-1}']}(\mathrm{tr}_{\alpha_1,\dots,\alpha_{n-1}}(\gamma_n), \gamma_n')
\end{aligned}$$

Now that we have a notion of homotopy on context substitutions, we want to study how this reflects onto the actions on types. In a well-behaved theory, the base change operation along homotopic substitutions should give equivalent types, and this is exactly the case:

Proposition 2.4.14.4. Let $\vdash \Gamma$ ctx and consider two context substitutions $\gamma, \gamma' : \Gamma.A_1 \dots A_m \to \Gamma.B_1 \dots B_m$ over Γ. If γ and γ' are homotopic, then for every $\Gamma.B_1 \dots B_n \vdash Q$ type, there is an equivalence of types $Q[\gamma] \simeq Q[\gamma']$ over $\Gamma.A_1 \dots A_m$.

Proof. We are given a sequence of equalities as in Remark 2.4.14.3 expressing a homotopy from γ to γ'. Then we have the transport function $\Gamma, x_1 : A_1, \dots x_m : A_m(x_1, \dots, x_{m-1}), z : Q(\gamma(x_1), \dots, \gamma(x_m)) \vdash \mathrm{tr}_{\alpha_1,\dots,\alpha_n}(z) : Q(\gamma'(x_1), \dots, \gamma(x_m))$ which has inverse $\mathrm{tr}_{\alpha_1^{-1},\dots,\alpha_n^{-1}}$ by Proposition 2.4.7.3. This provides an equivalence of types $Q[\gamma] \simeq Q[\gamma']$ over $\Gamma.A_1 \dots A_m$. □

We will be able to prove the converse implication to this by using the language of dependent sums. Now, we put in relation the notion of equivalence of context with that of homotopy of substitution. In particular, we see how the induction principle and the computation rule act dually to provide a proper two-sided inverse to an equivalence of contexts. Explicitly, by means of the induction principle we will construct an inverse substitution, which will prove to be a left inverse using the associated computation rule, and by a further induction principle we will prove it is, in fact, a right inverse as well.

Theorem 2.4.14.5. Let $\vdash \Gamma$ ctx, and let $\gamma = \mathrm{id}.\gamma_1 \dots \gamma_n : \Gamma.A_1 \dots A_m \to \Gamma.B_1 \dots B_n$ be a context substitution over Γ. The following are equivalent:

(1) γ is an equivalence of contexts over Γ.

(2) There exists a context substitution $\delta : \Gamma.B_1 \dots B_n \to \Gamma.A_1 \dots A_m$ over Γ such that $\delta \circ \gamma$ is homotopic to 1_Δ and $\gamma \circ \delta$ is homotopic to 1_Γ in the sense of Remark 2.4.14.3.

(3) For every $\Gamma.B_1 \dots B_n \vdash Q$ type, the base change along γ realises a homotopy bijection of generalised terms between Q and $Q[\gamma]$.

Proof. One can easily show that (2) implies (1), let us show the converse implication. We need to construct an inverse substitution $\delta : \Gamma.B_1 \dots B_n \to \Gamma.A_1 \dots A_m$. We can define it as an m-tuple $(\delta_1, \dots, \delta_m)$ where $\Gamma.B_1 \dots B_n \vdash \delta_i : A_i[p^n.\delta_1 \dots \delta_{i-1}]$. We construct all these by the definition of γ being an equivalence: each of them can be induced from a term over $\Gamma.A_1 \dots A_m$. The first term $\Gamma.B_1 \dots B_n \vdash \delta_1 : A_1[p^n]$ can be constructed out of the first variable term $\Gamma.A_1 \dots A_m \vdash q[p^{m-1}] : A_1[p^n][\gamma]$, as the term $\Gamma.B_1 \dots B_n \vdash \delta_1 :\equiv \mathrm{ind}(q[p^{m-1}]) : A_1[p^n]$. By computation rule, this comes equipped with an equality $\Gamma.A_1 \dots A_m \vdash \alpha_1 :\equiv$

$\mathrm{comp}(q[p^{n-1}]) : \mathrm{Id}_{\Delta_1[p^m]}(\delta_1[\gamma], q[p^{m-1}])$. The second term, of the form $\Gamma.B_1 \dots B_n \vdash \delta_2 : A_2[p^n.\delta_1]$, can be induced out of $\Gamma.A_1 \dots A_m \vdash A_2[p^m.\delta_1[\gamma]]$ type. By transport, we can instead produce a term in the fibre at $q[p^{m-1}]$, which is homotopic to $\delta_1[\gamma]$ via α_1. Here we have the second variable term $\Gamma.A_1 \dots A_m \vdash q[p^{m-2}] : A_2[p^m.q[p^{m-1}]] \equiv A[p^{m-1}]$, which induces $\Gamma.A_1 \dots A_m \vdash \mathrm{tr}_{\mathrm{comp}(q[p^{m-1}])}(q[p^{m-2}]) : A_2[p^m.\delta_1[\gamma]]$. Now, γ being an equivalence produces $\Gamma.B_1 \dots B_n \vdash \delta_2 :\equiv \mathrm{ind}(\mathrm{tr}_{\mathrm{comp}(q[p^{m-1}])}(q[p^{m-2}])) : A_2[p^n.\delta_1]$. Inductively, we can construct all δ_i's, each equipped with $\Gamma.A_1 \dots A_m \vdash \alpha_i : \mathrm{Id}_{A_i[p^{m-i+1}]}(\mathrm{tr}_{\alpha_1,\dots,\alpha_{i-1}}(\delta_i[\gamma]), q[p^{m-i}])$. The tuple $(\delta_1, \dots, \delta_n)$ determines by Remark 2.4.14.3 a context substitution $\delta : \Delta \to \Gamma$, which comes equipped with $\delta \circ \gamma \simeq \mathrm{id}_{\Gamma.A_1 \dots A_m}$ by means of the equality terms we provided via the computation rule. Now we need to prove that the converse composition is homotopic to the identity substitution as well. Let us write γ as sequence of terms $(\gamma_1, \dots, \gamma_n)$. We need to show that for every $i \in \{1, \dots, n\}$ the term $\Gamma.B_1 \dots B_n \vdash \gamma_i[\delta] : B_i[p^n.\gamma_1 \dots \gamma_{i-1}]$ comes equipped with a homotopy $\Gamma.B_1 \dots B_n \vdash \beta_i : \mathrm{Id}_{B_i[p^{n-i+1}]}(\mathrm{tr}_{\beta_1,\dots,\beta_{i-1}}(\gamma_i[\delta]), q[p^{n-i}])$. For this purpose, by γ being an equivalence, it suffices to provide a term of the fibres at γ. Inductively, β_1 can be constructed by providing an inhabitant of $\Gamma.A_1 \dots A_m \vdash \mathrm{Id}_{B_1[p^m]}(\gamma_1[\delta \circ \gamma], q[p^{m-1}][\gamma]) \equiv \mathrm{Id}_{A_1[p^m]}(\gamma_1[\delta \circ \gamma], \gamma_1)$ type, which is canonically inhabited by means of $\delta \circ \gamma \simeq 1_{\Gamma.A_1 \dots A_m}$. An analogous argument using this homotopy combined with transports holds for all the other β_i's. In particular we conclude that $\gamma \circ \delta \simeq 1_{\Gamma.B_1 \dots B_n}$, proving that (1) and (2) are equivalent.

Finally, (3) is a stronger version of (1), a priori, which provides a section of the base change along γ assignment from Q to $Q[\gamma]$. We need to show that this is a retraction as well. In other words, we need to show that for every term $\Gamma.B_1 \dots B_n \vdash t : Q$, the identity type $\Gamma.B_1 \dots B_n \vdash \mathrm{Id}_Q(\mathrm{ind}(t[\gamma]), t)$ is inhabited. By γ being an equivalence, the inhabitant of the base changed type $\Gamma.A_1 \dots A_m \vdash \mathrm{comp}(t[\gamma]) : \mathrm{Id}_{Q[\gamma]}(\mathrm{ind}(t[\gamma])[\gamma], t[\gamma])$ suffices to get $\Gamma.B_1 \dots B_n \vdash \mathrm{ind}(\mathrm{comp}(t[\gamma])) : \mathrm{Id}_Q(\mathrm{ind}(t[\gamma]), t)$. □

We conclude that an equivalence of contexts $\Delta \simeq \Gamma$ reasonably means that working over Δ is equivalent to working over Γ. In the assumptions one of the two contexts is well-understood, the equivalence allows to determines the types over the other with ease, under the induction principle and computation rule. This is a key idea in type theory, as we often want to characterise the so-called inductive types by specifying what it means to look at types over them. We will see further instances of this idea. Now, as a consequence of the result above, let us show that equivalences of contexts enjoys some stability properties.

Corollary 2.4.14.6 (Stability under extension)**.** Let $\vdash \Gamma$ ctx with $\vdash \Gamma.A, \Gamma.B$ ctx, with an equivalence of contexts $\gamma : \Gamma.A \to \Gamma.B$ over Γ. Then, for every $\Gamma.B \vdash Q$ type, the substitution $\gamma.Q : \Gamma.A.Q[\gamma] \to \Gamma.B.Q$ is an equivalence of contexts.

Proof. By Theorem 2.4.14.5, we know that there is an inverse substitution $\gamma^{-1} : \Gamma.B \to \Gamma.A$ which is itself an equivalence. In particular, we can construct a substitution $\gamma^{-1}.Q[\gamma] : \Gamma.B.Q[\gamma][\gamma^{-1}] \to \Gamma.A.Q[\gamma]$. Now, again by Theorem 2.4.14.5, we know that $Q[\gamma][\gamma^{-1}]$ is equivalent to Q over $\Gamma.B$, inducing an equivalence of contexts $\Gamma.B.Q \simeq \Gamma.B.Q[\gamma][\gamma^{-1}] \to \Gamma.A.Q[\gamma]$ by Remark 2.4.14.2. By construction, this is an inverse of $\gamma.Q$, hence is an equivalence of contexts again by Theorem 2.4.14.5. □

Lemma 2.4.14.7 (Stability under base change)**.** Let $\vdash \Gamma$ ctx and let $\vdash \Gamma.A, \Gamma.B$ ctx.

(1) Let $\gamma, \gamma' : \Gamma.A \to \Gamma.B$ be two homotopic context substitutions over Γ, given by sequences $\gamma = (\gamma_1, \dots, \gamma_n)$ and $\gamma' = (\gamma'_1, \dots, \gamma'_n)$. For any substitution $\vdash \delta : \Delta \to \Gamma$ the induced $\gamma[\delta], \gamma'[\delta] : \Delta.A[\delta] \to \Delta.B[\delta]$, defined by the base changes of the components γ_i and γ'_i, are homotopic.

(2) If $\gamma : \Gamma.A \to \Gamma.B$ is an equivalence of contexts over Γ, for ever substitution $\vdash \gamma : \Delta \to \Gamma$ the induced $\gamma[\delta] : \Delta.A[\delta] \to \Delta.B[\delta]$ is an equivalence.

Proof. Let us prove (1). We have homotopic substitutions $\gamma, \gamma' : \Gamma.A \to \Gamma.B$ over Γ. We want to show that homotopy of substitutions is compatible with base change, i.e. we want to induce a homotopy between the corresponding $\gamma[\delta], \gamma'[\delta] : \Delta.A[\gamma] \to \Delta.B[\gamma]$. First of all, the term $\Gamma.A_1 \dots A_m \vdash \alpha_1 : \mathrm{Id}_{B_1[p^m]}(\gamma_1, \gamma'_1)$ induces $\Gamma.A_1[\delta] \dots A_m[\delta.A_1 \dots A_{m-1}] \vdash \alpha_1[\gamma.A] : \mathrm{Id}_{B_1[\delta][p^m]}(\gamma_1[\delta.A], \gamma'_1[\delta.A])$ and so on for any homotopy of the sequence, as transport is compatible with base change. (2) is a straightforward consequence of (1) by Theorem 2.4.14.5. □

A further stability property, which we expect from the one for equivalences of types seen in Lemma 2.4.11.5, is the 2-out-of-3 property. In particular they are stable under composition:

Lemma 2.4.14.8. Equivalences of contexts have the 2-out-of-3 property.

Proof. Consider $\vdash \Gamma$ ctx with $\vdash \Gamma.A, \Gamma.B, \Gamma.C$ ctx, and with substitutions $\delta : \Gamma.A \to \Gamma.B$ and $\gamma : \Gamma.B \to \Gamma.C$ over Γ. We split the proof in the three cases.

(1) Assume that δ and γ are equivalences of contexts. We want to provide an induction principle and a computation rule for $\gamma \circ \delta$. For this purpose, consider $\Gamma.C \vdash Q$ type, and assume we have a term $\Gamma.A \vdash t : Q[\gamma][\delta]$. Then, by induction principle of δ we induce $\Gamma.B \vdash \mathrm{ind}(t) : Q[\gamma]$, and by induction principle of γ we induce $\Gamma.A \vdash \mathrm{ind}(\mathrm{ind}(t)) : Q$. This has a computation rule, because $\mathrm{ind}(\mathrm{ind}(t))[\delta][\gamma]$ is homotopic to $\mathrm{ind}(t)[\gamma]$, which is homotopic to t.

(2) Assume that δ and $\gamma \circ \delta$ are equivalences. For any $\Gamma \vdash Q$ type, assume we have a term $\Gamma.B \vdash s : Q[\gamma]$. Then, we may consider $\Gamma.C \vdash \mathrm{ind}(s[\delta]) : Q$ by relying on the induction principle of $\gamma \circ \delta$. Now, we need to show that the type $\Gamma.C \vdash \mathrm{Id}(\mathrm{ind}(s[\delta])[\gamma], s)$ type is inhabited. For this purpose, we use the induction principle of δ, by which we can construct a term of $\Gamma.C \vdash \mathrm{Id}(\mathrm{ind}(s[\delta])[\gamma][\delta], s[\delta])$ type. This is inhabited by the computation rule for $\gamma \circ \delta$.

(3) Assume that γ and $\gamma \circ \delta$ are equivalences. Consider a type $\Gamma.B \vdash S$ type. By Theorem 2.4.14.5, we can construct an inverse equivalence γ^{-1}, and in particular the terms of $\Gamma.B \vdash S$ type are in homotopy bijection with those of $\Gamma.C \vdash S[\gamma^{-1}]$ type. Then, the induction principle and computation rule for $\gamma \circ \delta$ yield the desired induction principle and computation rule for δ. □

Before focussing on a particular case, let us show a strictification result for equivalences of contexts, which expresses the fact that fibrewise homotopy equivalences are the same as homotopy equivalences.

Proposition 2.4.14.9 (Strictification of equivalences of contexts)**.** Let $\Gamma \vdash A$ type and let $\Gamma.A \vdash B, C$ type. Assume that we have an equivalence of contexts $p.f : \Gamma.A.B \to \Gamma.A.C$ over Γ. Then, it is automatically an equivalence of contexts over $\Gamma.A$.

Proof. By Theorem 2.4.14.5 we know that there exists an inverse $p^2.g_1.g_2 : \Gamma.A.C \to \Gamma.A.B$ that lives over Γ only, a priori. Out of this, we provide a right inverse over $\Gamma.A$ of $p.f$. Indeed, the substitution comes equipped with a homotopy $\Gamma, a : A, c : C(a) \vdash h_1(a, c) : \mathrm{Id}_A(g_1(a, c), a)$, and a homotopy $\Gamma, a : A, c : C(a) \vdash h_2(a, c) : \mathrm{Id}_{C(a)}(\mathrm{tr}_{h_1(a,c)}(f(g_2(a, c))), c)$. Thus, we can construct the substitution $p.\mathrm{tr}_{h_1}\, g_2 : \Gamma.A.C \to \Gamma.A.B$. This lives over $\Gamma.A$, and thus is a function of types $\Gamma, a : A, c : C(a) \vdash \mathrm{tr}_{h_1}(g_2(a, c)) : B(a)$. We claim that this is a right inverse of f as functions of types. Indeed, since by Proposition 2.4.7.3 transport commutes with functions, the equality h_2 provides an equality $\Gamma, a : A, c : C(a) \vdash h_2'(a, c) : \mathrm{Id}_{C(a)}(f(\mathrm{tr}_{h_1(a,c)}(g_2(a, c))), c)$ as desired. For the left inverse, we use the induction principle of the equivalence $p.f$. The universal term $\Gamma.A.B \vdash q : B[p]$, in fact, induces a term $\Gamma.A.C \vdash \mathrm{ind}(q) : B[p]$. This is a function of types over $\Gamma.A$, and comes equipped with a term $\Gamma.A.B \vdash \mathrm{comp}(q) : \mathrm{Id}_{B[p]}(\mathrm{ind}(q) \circ f, q)$, realising it as a left inverse of f. □

Now we move our attention towards some specific equivalences of contexts: contractible extensions.

Definition 2.4.14.10 (Contractible context extension)**.** Let $\vdash \Gamma$ ctx, and let Ξ be a sequence of dependent types over Γ. We say that $\Gamma.\Xi$ is a *contractible context extension of* Γ if there is an equivalence of contexts $\varphi : \Gamma \to \Gamma.\Xi$ such that $\Gamma \xrightarrow{\varphi} \Gamma.\Xi \xrightarrow{p^{\Xi}} \Gamma$ is the identity substitution.

Essentially, contractible context extensions are extensions of a base context that have "the same" types of the base contexts, in the sense seen in Theorem 2.4.14.5. The reader might have noticed an example of this phenomenon which we used frequently:

Example 2.4.14.11. The rules for the identity type tell us that for every $\Gamma \vdash A$ type, the context $\Gamma.A.A[p].\mathrm{Id}_A$ is a contractible context extension of $\Gamma.A$ under the equivalence $\mathrm{id}.q.\mathrm{refl} : \Gamma.A \to \Gamma.A.A[p].\mathrm{Id}_A$. The theory over an identity type is thus the same as the theory over the fibre at the reflexive term.

In light of this example, we define the following class of contractible context extensions.

Definition 2.4.14.12 (Elementary contractible context extension)**.** Let $\Gamma \vdash A$ type, and assume we have $\Gamma.A \vdash f : B[p]$. Then $\Gamma.A.B[p].\mathrm{Id}_B(f[p], q)$ is a contractible context extension of $\Gamma.A$, which we name an *atomic contractible context extension of* $\Gamma.A$ (along f). In general, we define an *elementary contractible context extension of* $\Gamma.A$ to be an iteration of atomic contractible context extensions of a context $\Gamma.A$

Note that atomic, and thus elementary, contractible context extensions are indeed contractible context extensions. This follows from Lemma 2.4.14.7, because $\Gamma.A.B[p].\mathrm{Id}_B(f[p],q) \to \Gamma.A.B[p]$ is the base change of the contractible context extension $\Gamma.B.B[p].\mathrm{Id}_B \to \Gamma.B.B[p]$ along the substitution $\Gamma.A.B[p] \to \Gamma.B.B[p]$ induced by f. The idea is that all the data in the contexts are essentially determined by $\Gamma, x : A$, as with $y : B, f(y) : \mathrm{Id}_B(f(x), y)$ we are adding data equal to something we already had.

Example 2.4.14.13. For $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type, the context $\Gamma, x : A, x' : A, \alpha : \mathrm{Id}_A(x,x'), y : B(x), y' : B(x'), \beta : \mathrm{Id}_{B(x')}(\mathrm{tr}_\alpha(y), y')$ is an elementary contractible context extension of $\Gamma, x : A, y : B(x)$.

Remark 2.4.14.14. In the semantics of a tribe, homotopy equivalences are the translation of equivalences of types, if we regard objects as types, or of equivalences of contexts, if we regard objects as contexts.

2.4.15 Embeddings of types

An embedding of types is a function of types that shall be thought as an inclusion. We will give various characterisations of these throughout this work. For the moment, it will be introduced as a meta-theoretical concept just as equivalences. However, the introduction of types of functions will allow us to construct a type of proofs of a function f being an embedding. This first definition of embedding will strictly recall the usual definition of injective map of sets: equality of the images is the same as equality in the domain.

Definition 2.4.15.1. Let $\Gamma \vdash A, B$ type, and let $\Gamma.A \vdash f : B[p]$ be a function of types. We say that f is an *embedding* if the function $\Gamma, a : A, a' : A, \alpha : \mathrm{Id}_A(a,b) \vdash \mathrm{ap}_f(\alpha) : \mathrm{Id}_B(f(a), f(a'))$ of preservation of equalities under f, coming from Lemma 2.4.3.1, is an equivalence of types.

This definition means that embeddings are those functions $f : A \to B$ for which we can detect equality of terms in B inside the source A. Unlike the set-theoretical case of injections, being an embedding is not equivalent to having a left homotopy inverse, i.e. a retraction. For this reason, we had to introduce the property of being an equivalence before that of being an embedding.

Example 2.4.15.2. By Proposition 2.4.11.7, we get that equivalences are embeddings.

Remark 2.4.15.3 (A remark on functoriality)**.** In homotopy type theory, e.g. in [BGL+17], following Remark 2.3.8.3, one can define embeddings in a simpler way, which is not explicitly functorial in the sense of Remark 2.3.8.1. Indeed, given $\Gamma \vdash A, B$ type with f function from A to B, it suffices to ask that for each $\Gamma \vdash a, a' : A$ the function $\Gamma, \alpha : \mathrm{Id}_A(a,a') \vdash \mathrm{ap}_f(\alpha) : \mathrm{Id}_B(f(a), f(a'))$ is an equivalence of types. This is because in homotopy type theory the rules are the same in all contexts, therefore a proof in context Γ provides a proof in context $\Gamma, x : A, y : A$ as well. However, in the present framework, this principle will be heavily despised: in general, if Γ is a nice context (i.e. an anima), a syntactic proof in context Γ would automatically give a syntactic proof only over *some* context extensions of Γ (i.e. those that extend Γ to an anima), and this will *not* be equivalent to our definition of embedding which requires the property to hold functorially for all generalised terms of A. In other words, this is an instance of Remark 2.3.8.3, and means that, with this formulation, being an embedding is a functorial condition rather than an objectwise condition.

Being embedding the generalisation of mono of ordinary category theory, we recover:

Lemma 2.4.15.4. Let $\Gamma \vdash A, B$ type and let $\Gamma.A \vdash f : B[p]$ be an embedding. Assume that f has a right homotopy inverse $\Gamma.B \vdash g : A[p]$. Then, f and g exhibit an equivalence of types between A and B.

Proof. By assumption we have a homotopy $\Gamma, b : B \vdash \beta(b) : \mathrm{Id}_B(f(g(b)), b)$. It suffices to show that g is also a left homotopy inverse of g. Note that we have a homotopy $\Gamma, a : A \vdash \beta(f(a)) : \mathrm{Id}_B(f(g(f(a))), f(a))$, and by f being an embedding we have an equivalence $\mathrm{Id}_A(g(f(a)), a) \simeq \mathrm{Id}_B(f(g(f(a))), f(a))$ over $\Gamma, a : A$. In particular, the former type is inhabited as well, inducing a homotopy from $g \circ f$ to q and thus the claim. □

2.5 Terminal type

In every type theory there is an empty context, which is introduced as a terminal object of the category of contexts and context substitutions. However, the basic syntax introduced until now does not include

the existence of a terminal type, over a fixed context, with only one term. Whenever we suppose that we can describe the semantics of a type theory in a tribe, this assumption is implicit in the fact that identities are fibrations, and from the fact that the isomorphism $\mathcal{E} \cong \mathcal{E}(*)$ maps the terminal object from the tribe of contexts to a terminal object in the tribe of types in absolute contexts. This is a consequence of the fact that in tribes we cannot really distinguish contexts and types, whereas in the syntax we need to be careful and make this correspondence precise. Thus, we impose rules introducing a terminal type. The *formation rule* is

$$\frac{\vdash \Gamma\,\mathrm{ctx}}{\Gamma \vdash \Delta^0_\Gamma\,\mathrm{type}}.$$

In particular, being it defined on every context, its compatibility with base change says that the terminal type in context Γ coincides with the weakening of the terminal type over the empty context or any other subcontext of Γ. We will see other types with this property. The *introduction rule* states the existence of a canonical term

$$\frac{\vdash \Gamma\,\mathrm{ctx}}{\Gamma \vdash *_\Gamma : \Delta^0_\Gamma}.$$

Terminality of Δ^0_Γ would be determined by adding the uniqueness of this inhabitant. However, instead of making it explicit in a rule, we prefer to characterise the terminal type as an inductive type, by means of an induction principle. Once again, this is a rule about how we can induce terms of dependent types over it. Our goal is in fact to make $\Gamma.\Delta^0_\Gamma$ a contractible context extension of Γ. The *elimination rule* states

$$\frac{\Gamma.\Delta^0_\Gamma \vdash C\,\mathrm{type} \quad \Gamma \vdash c : C[\mathrm{id}\,.*_\Gamma]}{\Gamma.\Delta^0_\Gamma \vdash \mathrm{ind}_{\Delta^0}(c) : C}.$$

This says that, in order to induce a term of the fibre of C at some term $a : \Delta^0_\Gamma$, it suffices to induce it at $*_\Gamma : \Delta^0_\Gamma$. The elimination rule comes equipped with a *computation rule* putting in relation the starting term c and the induced one. Morally, we want to say that the fibre of the induced term at $*_\Gamma$ is c up to homotopy:

$$\frac{\Gamma.\Delta^0_\Gamma \vdash C\,\mathrm{type} \quad \Gamma \vdash c : C[\mathrm{id}\,.*_\Gamma]}{\Gamma \vdash \mathrm{comp}_{\Delta^0}(c) : \mathrm{Id}_{C[\mathrm{id}\,.*_\Gamma]}(\mathrm{ind}_{\Delta^0}(c)[\mathrm{id}\,.*_\Gamma], c)}.$$

Once again, because the computation rule is unstrictified, being the production of a homotopy term, it will be itself compatible with base change.

Remark 2.5.0.1. As we realised an equivalence of contexts $\Gamma.\Delta^0_\Gamma \simeq \Gamma$, we know that over $\Gamma.*_\Gamma$ there are "the same types" as Γ by Theorem 2.4.14.5. Thus, the theory is invariant under extension with the terminal type.

In particular, this means that terms of types are the same as functions from the terminal type. Let us understand this in full detail.

Proposition 2.5.0.2 (Terms as maps from $*_\Gamma$)**.** Let $\vdash \Gamma\,\mathrm{ctx}$ and $\Gamma \vdash A\,\mathrm{type}$. We have a bijection, up to homotopy, between terms $\Gamma \vdash a : A$ and functions of types $\overline{a} : \Delta^0_\Gamma \to A$ compatible with base change realised by the following assignments:

(1) Given $\Gamma \vdash a : A$, we associate $\Gamma.\Delta^0_\Gamma \vdash \overline{a} :\equiv \mathrm{ind}_{\Delta^0}(a) : A[p]$.

(2) Given $\Gamma.\Delta^0_\Gamma \vdash \overline{a} : A[p]$, we associate $\Gamma \vdash \overline{a}(*_\Gamma) : A$.

Furthermore, this is compatible with application of functions: for any $\Gamma \vdash B\,\mathrm{type}$ and function $\Gamma.A \vdash f : B[p]$, we have $\Gamma, a : A, x : \Delta^0_\Gamma \vdash \mathrm{Id}_B(\overline{f(a)}(x), f(\overline{a}(x)))$. Thus, the functions $\overline{f}(a), f \circ \overline{a} : \Delta^0_\Gamma \to B$ are homotopic.

Proof. The provided correspondence is a homotopy bijection by the induction principle, or as seen more generally in Theorem 2.4.14.5. The last part of the claim follows since, by the elimination rule, it suffices to check $\overline{f(a)}(*_\Gamma)$ is homotopic to $f(\overline{a}(*_\Gamma))$, and by computation rule they are both homotopic to $f(a) : B$. □

Remark 2.5.0.3 (Δ^0 has only one term)**.** Let $\vdash \Gamma\,\mathrm{ctx}$. The type $\Gamma \vdash \Delta^0_\Gamma\,\mathrm{type}$ has only the term $*_\Gamma$ up to homotopy. Indeed, the term $\Gamma \vdash \mathrm{refl}_{*_\Gamma} : \mathrm{Id}_{\Delta^0_\Gamma}(*_\Gamma, *_\Gamma)$ induces, by the elimination rule, a term $\Gamma, x : \Delta^0_\Gamma \vdash \mathrm{ind}_{\Delta^0}(\mathrm{refl}_{*_\Gamma}) : \mathrm{Id}_{\Delta^0_\Gamma}(x, *_\Gamma)$.

Corollary 2.5.0.4 (Terminality of $*_\Gamma$)**.** Let $\Gamma \vdash A$ type. Then, there exists a canonical function $!_A : A \to \Delta^0_\Gamma$ given by $\Gamma.A \vdash *_{\Gamma.A} : \Delta^0_{\Gamma.A}$, and it is unique up to homotopy.

Semantically, $\Gamma \vdash \Delta^0_\Gamma$ type corresponds to the identity fibration $\Gamma \xrightarrow{=} \Gamma$, which is in fact the weakening of 1_* at the terminal map $\Gamma \twoheadrightarrow *$. We might refer to the type Δ^0_Γ as the 0-*simplex*, especially in the next chapter which will be about synthetic category theory explicitly.

2.5.1 Identity in Δ^0

Unsurprisingly, the identity type of the terminal type will be the terminal type itself.

Remark 2.5.1.1. Let $\vdash \Gamma$ ctx. We have an equivalence of types $\mathrm{Id}_{\Delta^0_\Gamma} \simeq \Delta^0_{\Gamma.\Delta^0_\Gamma.\Delta^0_\Gamma[p]}$ over $\Gamma.\Delta^0_\Gamma.\Delta^0_\Gamma[p]$. This is because we have an equivalence of contexts $\Gamma.\Delta^0_\Gamma.\Delta^0_\Gamma[p].\mathrm{Id}_{\Delta^0_\Gamma} \simeq \Gamma.\Delta^0_\Gamma \simeq \Gamma.\Delta^0_\Gamma.\Delta^0_\Gamma[p].\Delta^0_\Gamma[p^2]$.

This means that not only all the terms $x, y : \Delta^0$ are equal, but they are equal via a unique equality, and so on for all higher homotopies.

2.5.2 Contractible types

$\Gamma \vdash \Delta^0_\Gamma$ is the prototype of a type with only one term up to homotopy. Therefore, we introduce the class of types with this same property, as we will not care about a type being strictly Δ^0_Γ, but instead about being equivalent to it. However, the condition of being equivalent to the terminal type can be simplified in more than one way through the following:

Lemma 2.5.2.1. Let $\Gamma \vdash A$ type. The following are equivalent:

(1) The function $!_A : A \to \Delta^0_\Gamma$ is an equivalence of types.

(2) The function $!_A : A \to \Delta^0_\Gamma$ has a retraction.

(3) A is inhabited and the function $!_A : A \to \Delta^0_\Gamma$ is an embedding of types.

(4) A has a *centre of contraction*, i.e. a term $\Gamma \vdash c : A$ together with an inhabitant of $\Gamma, a : A \vdash \mathrm{Id}_A(a, c)$ type.

Proof. Clearly (1) implies (2) and (3). Conversely, both in (2) and (3) the type A is inhabited. Every term $\Gamma \vdash a : A$ gives rise to a map $\Gamma.\Delta^0_\Gamma \vdash \overline{a} : A[p]$ by Proposition 2.5.0.2, and this is always a section of the terminal function from A to Δ^0_Γ. Thus, both (2) and (3) imply (1), where for the latter we rely on Lemma 2.4.15.4. Finally, (4) is clearly implied by the other conditions. Conversely, it realises $\overline{c} : \Delta^0_\Gamma \to A$ as a retraction of $!_A$. Indeed, $\overline{c}(!_A(a))$ is homotopic to c, and therefore $\mathrm{Id}_A(a, c) \simeq \mathrm{Id}_A(a, \overline{c}(!_A(a)))$ by Proposition 2.4.11.6. □

As a consequence, we can characterise contractibility more simply as follows:

Definition 2.5.2.2 (Contractible type)**.** We say that $\Gamma \vdash A$ type is *contractible* if the terminal function $A \to \Delta^0_\Gamma$ has a retraction. By Lemma 2.5.2.1, this is equivalent to $!_A$ being an equivalence, or an embedding whenever A is inhabited, or again to A having a centre of contraction.

In other words, all the terms of A are equal to the same term functorially, by the fact that contractibility means having a centre of contraction.

Lemma 2.5.2.3. The property of being contractible is stable under base change and equivalence.

Proof. The first follows from Δ^0 being compatible with base change and Lemma 2.4.11.3. The second is a consequence of Lemma 2.5.2.1 and Lemma 2.4.11.5. □

The notion of a contractible type over Γ essentially means that the theory over them is the same as the theory over Γ. Indeed, it is strongly linked to the concept of contractible context extension:

Lemma 2.5.2.4. Let $\Gamma \vdash A$ type. A is contractible if and only if $\Gamma.A$ is a contractible context extension of Γ.

Proof. Follows by the equivalence $\Gamma.\Delta^0_\Gamma \simeq \Gamma$ together with Remark 2.4.14.2 and Lemma 2.4.14.8. □

Lemma 2.5.2.5. Let $\Gamma \vdash A, B$ type with B contractible. Then all the functions $\Gamma.A \vdash f : B[p]$ are homotopic.

Proof. As B is contractible over Γ, then $B[p]$ is contractible over $\Gamma.A$, thus has one term up to homotopy. □

A very important property is that a retract of a contractible type is contractible.

Definition 2.5.2.6 (Retract)**.** Let $\Gamma \vdash A, B$ type. We say that A is a *retract of* B if there exists a function $\Gamma.A \vdash s : B[p]$ together with a retraction $\Gamma.B \vdash r : A[p]$.

In other words, we have $\Gamma, a : A \vdash \alpha(a) : \mathrm{Id}_A(r(s(a)), a)$ that realises A as a retract of B.

Lemma 2.5.2.7 (Retract of a contractible type)**.** Le $\Gamma \vdash A, B$ type. If A is a retract of B via $\Gamma.A \vdash s : B[p]$, and B is contractible, then A is contractible.

Proof. Follows by the fact that retractions compose: Both r and $B \to \Delta^0_\Gamma$ have a retraction, so that their composition also have a retraction □

2.6 Dependent sum

We now move on to the next type constructor, the dependent sum. This takes a dependent type B over an extended context $\Gamma.A$ and returns a type $\Sigma_A B$ over Γ. Intuitively, its terms should consist of pairs of a term a of A and a term of the fibre $B(a)$. Therefore, we will introduce the type $\Sigma_A B$ with an equivalence of contexts between $\Gamma, x : A, y : B(x)$ and $\Gamma, s : \Sigma_A B$.

As the name suggests, since a term of this dependent sum will be a choice of $a : A$ and a term inside the fibre $B(a)$, the type $\Sigma_A B$ can be thought as the sum, or disjoint union, of the fibres of B indexed by the terms of A. It carries the name of *dependent sum of* B *over* A as we picture a dependent type as the family of its fibres. To emphasise this perspective, we will denote the dependent sum via $\Gamma \vdash \Sigma_{x:A} B(x)$ type.

Semantically, this operation is modelled via composition of fibrations, by unwinding the meaning of providing a section of a composition. Thus, the existence of dependent sums is implicit in the semantics of tribes, as fibrations are closed under composition. This is what allows us to regard contexts as types in that setting, since we can always sum a chain of dependent types to be a type over the point. Another aspect to remark is that, semantically, postcomposition along a fibration $f : B \twoheadrightarrow A$ is the left adjoint to pullback along f. The rules will indeed describe the summing operation as a left adjoint of the weakening.

2.6.1 Rules of the dependent sum

First of all, we have the *formation rule*, which allows us to form the type

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Gamma \vdash \Sigma_A B \text{ type}}.$$

The dependent sum $\Sigma_A B$ comes equipped with a canonical way of inducing terms as pairs of a term of A and a term of the fibre of B, by means of the *introduction rule*

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash a : A \quad \Gamma \vdash b : B[\mathrm{id}.a]}{\Gamma \vdash \mathrm{pair}(a, b) : \Sigma_A B}.$$

Remark 2.6.1.1. Note that, for every $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type, we have $\Gamma.A.B \vdash \mathrm{pair} :\equiv \mathrm{pair}(q[p], q) : (\Sigma_A B)[p^2]$ which restricts to a function of types from B to $(\Sigma_A B)[p]$ over $\Gamma.A$. In particular, $\Gamma, a : A, b : B(x) \vdash \mathrm{pair}(a, b) : \Sigma_{x:A} B(x)$ is immediately compatible with equality on the second entry by Lemma 2.4.3.1. We will soon realise the importance of this function, being it the unit of the adjunction between dependent sum and weakening.

Now there are usually two equivalent ways to complete the definition of the dependent sum. On the one hand, we could say that all its terms are of the above form: this means that there are two projection assignments from the terms of $\Sigma_A B$ to the terms of A and of the fibres of B, which provide a bijection with the pair. By Theorem 2.4.14.5, this condition is the same as giving an equivalence of context $\Gamma.\Sigma_A B$ and $\Gamma.A.B$ over Γ, which can be better described by means of induction and computation rules. This will be the path we follow to complete the rules of dependent sums. First of all, we consider the substitution $p^2.\mathrm{pair} : \Gamma.A.B \to \Gamma.\Sigma_A B$, which takes a dependent type $\Gamma, p : \Sigma_A B \vdash E(p)$ type into the fibre $\Gamma, a : A, b : B \vdash E(\mathrm{pair}(a,b))$. The *induction principle* is

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type} \quad \Gamma.\Sigma_A B \vdash E \text{ type} \quad \Gamma.A.B \vdash e : E[p^2.\mathrm{pair}]}{\Gamma.\Sigma_A B \vdash \mathrm{ind}_\Sigma(e) : E}.$$

This tells us that when we want to induce a term of a dependent type over $\Sigma_A B$, it suffices to provide a term of the dependent type parametrised by those terms of the form $\mathrm{pair}(a,b)$ for some $a : A$ and $b : B(a)$. In other words, the induction principle says that given $\Gamma, s : \Sigma_{x:A} B(x) \vdash E(s)$ type for which we can construct a term $\Gamma, a : A, b : B(a) \vdash e(a,b) : E(\mathrm{pair}(a,b))$ we can induce a term $\Gamma, s : \Sigma_{x:A} B(x) \vdash \mathrm{ind}_\Sigma(e)(s) : E(s)$ type.

Similarly to every induction principle, this comes equipped with a computational tool, whose goal is to relate $\mathrm{ind}_\Sigma(e)$ with the term e itself. More explicitly, the *computation rule* says that, after we form $\Gamma.\Sigma_A B \vdash \mathrm{ind}_\Sigma(e) : E$, if we pull back along $p^2.\mathrm{pair} : \Gamma.A.B \to \Gamma.\Sigma_A B$, we get back the term e, up to homotopy:

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type} \quad \Gamma.\Sigma_A B \vdash E \text{ type} \quad \Gamma.A.B \vdash e : E[p^2.\mathrm{pair}]}{\Gamma.A.B \vdash \mathrm{comp}_\Sigma(e) : \mathrm{Id}_{E[p^2.\mathrm{pair}]}(\mathrm{ind}_\Sigma(e)[p^2.\mathrm{pair}], e)}.$$

Once again, this is made non-strict unlike the classical rule, as it holds only up to homotopy, closely to the philosophy of [vdBdB21] although no dependent sum is introduced in that setting. Because it provides a term, there will also be a rule making it compatible with base change.

Convention. In this section we will write explicitly the pair constructors. However, later on we may omit writing $\mathrm{pair}(a,b)$ and write instead (a,b), even for iterated dependent sums. Indeed, the reader should really think these terms of dependent sums as sequences of terms of the correct fibres.

Remark 2.6.1.2. The induction principle is compatible with equality, More precisely, consider any $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type with a dependent type $\Gamma.\Sigma_A B \vdash E$ type, and terms $\Gamma, a : A, b : B(a) \vdash e(a,b), e'(a,b) : E(\mathrm{pair}(a,b))$. The base change of $\Gamma, s : \Sigma_{x:A} B(x) \vdash \mathrm{Id}_{E(s)}(\mathrm{ind}_\Sigma(e)(s), \mathrm{ind}_\Sigma(e')(s))$ type along pair is equivalent to $\Gamma, a : A, b : B(a) : \mathrm{Id}_{E(\mathrm{pair}(a,b))}(e, e')$ type under the computation via Proposition 2.4.11.6. In particular, any equality between e and e' induces an equality between the $\mathrm{ind}_\Sigma(e)$ and $\mathrm{ind}_\Sigma(e')$ through the induction principle.

2.6.2 Projection maps

Because we have an equivalence of contexts $p^2.\mathrm{pair} : \Gamma.A.B \to \Gamma.\Sigma_A B$ over Γ, we can use the results we proved in general in that discussion. In particular, we know that we get an inverse substitution $p.\mathrm{pr}_1.\mathrm{pr}_2 : \Gamma.\Sigma_A B \to \Gamma.A.B$ determining an inverse up to homotopy. As a consequence, we inherit projection assignments that, on a term of $\Sigma_A B$ of the form $\mathrm{pair}(a,b)$, assign the two components. This allows to identify every term of $\Sigma_A B$ with something of the form $\mathrm{pair}(a,b)$ up to equality.

Proposition 2.6.2.1 (Projection maps)**.** Let $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. We have the following terms.

(1) *First projection:* $\Gamma, s : \Sigma_A B \vdash \mathrm{pr}_1(s) :\equiv \mathrm{ind}_\Sigma(q[p])(s) : A$, defining a function from $\Sigma_A B$ to A over Γ.

(2) pr_1 *on pairs:* $\Gamma, a : A, b : B(a) \vdash \mathrm{pr}_1^{\mathrm{eq}}(a,b) :\equiv \mathrm{comp}_\Sigma(q[p])(a,b) : \mathrm{Id}_{A[p]}(\mathrm{pr}_1(\mathrm{pair}(a,b)), a)$.

(3) *Second projection:* $\Gamma, s : \Sigma_A B \vdash \mathrm{pr}_2(s) :\equiv \mathrm{ind}_\Sigma(\mathrm{tr}_{(\mathrm{pr}_1^{\mathrm{eq}})^{-1}}(q))(s) : B(\mathrm{pr}_1(\mathrm{pair}(a,b)))$, defining a dependent function from $\Sigma_A B$ to $B[p.\mathrm{pr}_1]$ over Γ.

(4) pr_2 *on pairs:* $\Gamma, a : A, b : B(a) \vdash \mathrm{pr}_2^{\mathrm{eq}}(a,b) : \mathrm{Id}_{B(\mathrm{pr}_1(\mathrm{pair}(a,b)))}(\mathrm{pr}_2(\mathrm{pair}(a,b)), \mathrm{tr}_{(\mathrm{pr}_1^{\mathrm{eq}})^{-1}}(b))$.

(5) *Terms of* $\Sigma_A B$*:* $\Gamma, s : \Sigma_{x:A} B(x) \vdash \mathrm{termaspair}(s) : \mathrm{Id}_{\Sigma_A B}(\mathrm{pair}(\mathrm{pr}_1(s), \mathrm{pr}_2(s)), s)$.

Proof. Follows by the characterisations of equivalences of contexts in Theorem 2.4.14.5. □

We constructed projection assignments pr_1 and pr_2 which we made explicit on pairs. These provide the substitution $p.\mathrm{pr}_1.\mathrm{pr}_2 : \Gamma.\Sigma_A B \to \Gamma.A.B$ as an inverse to $p^2.\mathrm{pair} : \Gamma.A.B \to \Gamma.\Sigma_A B$ in the sense of Theorem 2.4.14.5. In particular, (5) determines all the terms of $\Sigma_A B$, by saying that they are all of the form $\mathrm{pair}(a, b)$ up to homotopy, with components given by the projections.

Remark 2.6.2.2. Note that, in the classical situation, where it is judgementally true that $\mathrm{pr}_1(a, b) \equiv a$, it is significantly easier to induce pr_2. Indeed, the universal term is itself a term $\Gamma, a : A, b : B(x) \vdash b : B(a) \equiv B(\mathrm{pr}_1(\mathrm{pair}(a, b)))$. Instead, the fact of having a non-strict equality between $\mathrm{pair}(a, b)$ and a forces us to use transport, in order to translate the above universal term into a term of $B(\mathrm{pr}_1(\mathrm{pair}(a, b)))$. Furthermore, unlike the classical case where $\mathrm{pr}_2(\mathrm{pair}(a, b)) \equiv b$ judgementally, here we only have that the transport $\mathrm{tr}_{\mathrm{pr}_1^{\mathrm{eq}}}(\mathrm{pr}_2(\mathrm{pair}(a, b)))$, function from $B(a)$ to $B(a)$ over $\Gamma, a : A, b : B(a)$, is homotopic to the identity.

Lemma 2.6.2.3. Let $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. The projections are compatible with equality:

(1) pr_1 *and* Id*:* $\Gamma, s : \Sigma_{x:A} B(x), s' : \Sigma_{x:A} B(x), \sigma : \mathrm{Id}_{\Sigma_A B}(s, s') \vdash \mathrm{ap}_{\mathrm{pr}_1}(s) : \mathrm{Id}_A(\mathrm{pr}_1(s), \mathrm{pr}_1(s'))$.

(2) pr_2 *and* Id*:* $\Gamma, s : \Sigma_{x:A} B(x), s' : \Sigma_{x:A} B(x), \sigma : \mathrm{Id}_{\Sigma_A B}(s, s') \vdash \mathrm{ap}_{\mathrm{pr}_2}(s) : \mathrm{Id}_A(\mathrm{tr}_{\mathrm{ap}_{\mathrm{pr}_1}(s)}(\mathrm{pr}_2(s)), \mathrm{pr}_2(s'))$.

Proof. The claim follows from Lemma 2.4.3.1 and Lemma 2.4.5.2. □

2.6.3 Functoriality of dependent sum

The construction of the dependent sum is functorial. More explicitly, we can define an action of Σ on functions and check it is compatible with composition of functions and the identical function.

Proposition 2.6.3.1. Let $\Gamma \vdash A$ type and $\Gamma.A \vdash B, C$ type, and assume we have a function of types $\Gamma.A.B \vdash f : C[p]$ over $\Gamma.A$. We can construct the following terms.

(1) *Functoriality of* Σ_A*:* $\Gamma, s : \Sigma_{x:A} B(x) \vdash \Sigma_A f(s) : \Sigma_{x:A} C(x)$.

(2) *Computation of functoriality of* Σ_A*:* $\Gamma, x : A, y : B(x) \vdash \mathrm{funsum}(x, y) : \mathrm{Id}_{\Sigma_A C}(\Sigma_A f(\mathrm{pair}(x, y)), \mathrm{pair}(x, f(y)))$.

(3) *Compatibility with composition:* For any $\Gamma.A \vdash D$ type and function of types $\Gamma.A.C \vdash g : D[p]$ type, we have a homotopy $\Gamma, s : \Sigma_{x:A} B(x) \vdash \mathrm{compsum}(s) : \mathrm{Id}_{\Sigma_{x:A} D(x)}(\Sigma_A g(\Sigma_A f(s)), \Sigma_A(g \circ f)(s))$.

(4) *Compatibility with universal term:* $\Gamma, s : \Sigma_{x:A} B(x) \vdash \mathrm{Id}_{\Sigma_A B}(\Sigma_A(q)(s), s)$ type is inhabited.

(5) *Compatibility with equality of maps:* For any other function $\Gamma.A.B \vdash f' : C[p]$ and homotopy $\Gamma.A.B \vdash \alpha : \mathrm{Id}_{C[p]}(f, f')$, we have $\Gamma, s : \Sigma_{x:A} B(x) \vdash \mathrm{ind}_\Sigma(\mathrm{ap}_{\mathrm{pair}(q[p], q)}(\alpha))(s) : \mathrm{Id}_{\Sigma_A C}(\Sigma_A f(s), \Sigma_A f'(s))$ type.

Proof. For (1) and (2), apply induction and computation rule to the term $\Gamma, x : A, y : B(x) \vdash \mathrm{pair}(x, f(y)) : \Sigma_{x:A} C(x)$. For (3), the induction principle together with the computation term of (2) provide an inhabitant of the fibre over pairs. (4) follows by the computation rule of the terms $\Sigma_A(q)$ which is induced by $\Gamma, x : A, y : B(x) \vdash \mathrm{pair}(x, y) : \Sigma_{x:A} C(x)$. For (5), we use Remark 2.6.1.2 on the homotopy $\Gamma, x : A, y : B(x) \vdash \mathrm{ap}_{\mathrm{pair}(q[p], q)}(\alpha(y)) : \mathrm{Id}_{\Sigma_A C}\,\mathrm{pair}(x, f(y), \mathrm{pair}(x, f'(y)))$ to get a homotopy between $\Sigma_A f$ and $\Sigma_A f'$. □

Therefore, out of $\Gamma.A.B \vdash f : C[p]$, we can induce $\Gamma.\Sigma_A B \vdash \Sigma_A f : (\Sigma_A C)[p]$, which, by computation, acts on pairs as the original map f up to homotopy. This is functorial being compatible with composition and identity up to homotopy. Now, from functoriality of dependent sum and its compatibility of equality, we get the following fundamental property about the dependent sum as for every other type constructor we will see:

Proposition 2.6.3.2 ($\Sigma_A B$ is compatible with equivalence in B)**.** Let $\Gamma \vdash A$ type and $\Gamma.A \vdash B, C$ type, and consider $\Gamma.A.B \vdash f : C[p]$. If f is an equivalence from B to C over $\Gamma.A$, then $\Sigma_A f$ is an equivalence from $\Sigma_A B$ to $\Sigma_A C$ over Γ. In other words, dependent sum preserves equivalences.

Proof. A left inverse g for f provides a left inverse $\Sigma_A g$ for $\Sigma_A f$ by Proposition 2.6.3.1: (1) provides a homotopy between $\Sigma_A(g) \circ \Sigma_A(f)$ and $\Sigma_A(g \circ f)$, and (5) yields a homotopy between $\Sigma_A(g \circ f)$ and the function $\Sigma_A(q)$ from $\Sigma_A B$ to $\Sigma_A B$ induced by the universal term, which is equal to the identity by (4). □

More in general, we wish this property to hold for equivalences both in A and B, as we will prove later. Furthermore, we can iterate dependent sums in the order we prefer.

Proposition 2.6.3.3. Let $\Gamma \vdash A$ type, $\Gamma.A \vdash B$ type and $\Gamma.A.B \vdash C$ type. Then, there is an equivalence $\Sigma_{\Sigma_A B} C[p^2.\mathrm{pair}] \simeq \Sigma_A \Sigma_B C$ that fits into a commutative triangle

$$\begin{array}{ccc} \Sigma_{\Sigma_A B} C[p^2.\mathrm{pair}] & \xrightarrow{\sim} & \Sigma_A \Sigma_B C \\ & \searrow^{\mathrm{pr}_1} & \downarrow{\scriptstyle \Sigma_A \mathrm{pr}_1} \\ & & \Sigma_A B \end{array}.$$

Proof. By Corollary 2.4.14.6, we have that $\Gamma.\Sigma_A.\Sigma_B.C \simeq \Gamma.A.B.C \simeq \Gamma.\Sigma_A B.C[p^2.\mathrm{pair}] \simeq \Gamma.\Sigma_{\Sigma_A B}[p^2.\mathrm{pair}]$, which determines the wished equivalence. By induction principle and computation rules, one can check that the provided triangle commutes. □

In other words, Proposition 2.6.3.3 tells us that we can reduce the length of a context in the order we prefer and the result will not change.

2.6.4 The homotopy category of contexts

We now want to characterise the identity type of the dependent sum, in order to determine the notion of equality on its terms. For this purpose, let us define an identity type of pairs. This will first of all lead us to the definition of a homotopy category of contexts and deduce properties of equivalences of contexts

Construction 2.6.4.1. Let $\Gamma \vdash A$ type and $\Gamma, x : A \vdash B(x)$ type. We construct the type $\Gamma, x : A, x' : A, y : B(x), y' : B(x') \vdash \mathrm{Id}^{\mathrm{pair}}_{A.B}((x,y),(x',y')) :\equiv \Sigma_{\alpha:\mathrm{Id}_A(x,x')} \mathrm{Id}_{B(x')}(\mathrm{tr}_\alpha(y), y')$. The terms of this type are, up to homotopy, pairs of $\alpha : \mathrm{Id}_A(x,x')$ and $\beta : \mathrm{Id}_{B(x')}(\mathrm{tr}_\alpha(y), y')$. In particular, by the rules of the dependent sum, the context $\Gamma, x : A, y : B(x), x' : A, y' : B(x'), z : \mathrm{Id}^{\mathrm{pair}}_{A.B}((x,y),(x',y'))$ is equivalent to the context $\Gamma, x : A, y : B(x), x' : A, y' : B(x'), \alpha : \mathrm{Id}_A(x,x'), \zeta : \mathrm{Id}_{B(x')}(\mathrm{tr}_\alpha(y), y')$, and therefore, by Example 2.4.14.13, it is a contractible context extension of $\Gamma, x : A, y : B(x)$ by iteration of the rules of Id. By a straightforward generalisation, whenever we have a chain of dependent types $\vdash \Gamma.A_1 \dots A_n$ ctx, we can construct a type given by $\Gamma, x_1 : A_1, x_1' : A_1, \dots, x_n : A_n(x_1,\dots,x_n), x_n' : A_n(x_1',\dots,x_n') \vdash \mathrm{Id}^{n-\mathrm{tuples}}_{A_1\dots A_n}((x_1,\dots,x_n),(x_1',\dots,x_n')) :\equiv \Sigma_{\alpha_1:\mathrm{Id}_{A_1}(x_1,x_1')} \cdots \Sigma_{\alpha_{n-1}:\mathrm{Id}_{A_{n-1}(x_1',\dots,x_{n-2}')}(\mathrm{tr}_{\alpha_1,\dots,\alpha_{n-2}}(x_{n-1}),x_{n-1}')} \mathrm{Id}_{A_n(x_1',\dots,x_{n-1}')}(\mathrm{tr}_{\alpha_1,\dots,\alpha_{n-1}}(x_n), x_n')$. Its terms are the provision of a n-tuple of equalities with a transport term at each step.

Consider $\Gamma.A_1 \dots A_m$ and $\Gamma.B_1 \dots B_n$ context extensions of Γ. Given two substitutions $p^m.\gamma_1 \dots \gamma_n$ and $p^m.\gamma_1' \dots \gamma_n'$, a homotopy as in Remark 2.4.14.3 is a term of $\mathrm{Id}^{n-\mathrm{tuples}}_{B_1\dots B_n}((\gamma_1,\dots,\gamma_n),(\gamma_1',\dots,\gamma_n'))$. We can finally prove the converse implication for Proposition 2.4.14.4

Proposition 2.6.4.2. Let $\vdash \Gamma$ ctx and consider two context substitutions $\gamma, \gamma' : \Gamma.A_1 \dots A_m \to \Gamma.B_1 \dots B_m$ over Γ. Then, γ and γ' are homotopic, if and only if for every $\Gamma.B_1 \dots B_n \vdash Q$ type, there is an equivalence of types $Q[\gamma] \simeq Q[\gamma']$ over $\Gamma.A_1 \dots A_m$.

Proof. The direct implication was proven in Proposition 2.4.14.4. Conversely, we consider $\Gamma, y_1 : B_1, \dots, y_n : B_n(y_1,\dots,y_{n-1}), x_1' : A_1, \dots, x_n' : A_n(x_1',\dots,x_n') \vdash \mathrm{Id}^{n-\mathrm{tuples}}_{B_1\dots B_n}((\gamma_1(x_1'),\dots,\gamma_n(x_n')),(y_1,\dots,y_n))$ and we apply base change with respect to γ and γ'. This yields, by assumption, equivalent types $\Gamma, x_1 : A_1, \dots, x_n : A_n(x_1,\dots,x_{n-1}) \vdash \mathrm{Id}^{n-\mathrm{tuples}}_{B_1\dots B_n}((\gamma_1(x_1),\dots,\gamma_n(x_1,\dots x_{n-1})),(\gamma_1(x_1),\dots,\gamma_n(x_1,\dots x_{n-1})))$ type and $\Gamma, x_1 : A_1, \dots, x_n : A_n(x_1,\dots,x_{n-1}) \vdash \mathrm{Id}^{n-\mathrm{tuples}}_{B_1\dots B_n}((\gamma_1(x_1),\dots,\gamma_n(x_1,\dots x_{n-1})),(\gamma_1'(x_1),\dots,\gamma_n'(x_1,\dots x_{n-1})))$. As the first is canonically inhabited by the n-tuple of reflexive terms, then so is the second under the equivalence of types. □

An immediate consequence is that homotopy of context substitutions is compatible with composition. Therefore, we can consider the ordinary category $\mathcal{E}$ of contexts and take the equivalence classes on arrows under this relation. This allows the following definition.

Definition 2.6.4.3. The *homotopy category of contexts* is the ordinary category $\mathrm{Ho}(\mathrm{Ctx})$ whose objects are contexts and arrows are homotopy classes of context substitutions. For every $\vdash \Gamma\,\mathrm{ctx}$, we can consider the full subcategory $\mathrm{Ctx}(\Gamma)$ of Ctx spanned by iterations of weakening substitutions into Γ. We define the *homotopy category of contexts over* Γ as the ordinary category $\mathrm{Ho}(\mathrm{Ctx}(\Gamma))$ whose objects are (iterated) context extensions of Γ and arrows are homotopy classes of substitutions over Γ.

Note that isomorphisms in $\mathrm{Ho}(\mathrm{Ctx}(\Gamma))$ are equivalences of contexts over Γ by Theorem 2.4.14.5. Therefore, we could reprove Lemma 2.4.14.8, and furthermore show that equivalences of contexts enjoy the 2-out-of-6 property as well, by the basic properties of isomorphisms in an ordinary category.

Remark 2.6.4.4. Dependent sums allow us to conclude that, given $\vdash \Gamma\,\mathrm{ctx}$, we have an equivalence of ordinary categories $\mathrm{Ho}(\mathrm{Ctx}(\Gamma)) \simeq \mathrm{Ho}(\mathrm{Type}_{/\Gamma})$. Indeed, we can assign to each context extension $\Gamma.A_1 \dots A_n$ the type $\Gamma \vdash \Sigma_{A_1} \cdots \Sigma_{A_{n-1}} A_n$ type. This assignment can be extended to arrows by Proposition 2.6.3.1 compatibly with homotopy. Conversely, to each dependent type $\Gamma \vdash A$ type we assign the context extension $\Gamma.A$ and with the obvious action on arrows. In particular, in empty context, we have $\mathrm{Ho}(\mathrm{Type}) \simeq \mathrm{Ho}(\mathrm{Ctx})$.

In some essence, this is the reason why, in a tribe, we can switch between the point of view of objects as types and as contexts. Further, we now observe why taking the local tribes over types, instead of arbitrary chains of dependent types, describes all the local theories in the semantics.

Remark 2.6.4.5. By definition, for $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type, we have an equivalence of contexts $\Gamma.\Sigma_A B \simeq \Gamma.A.B$ over Γ. Therefore, by Lemma 2.4.14.8, for every finite chain of dependent types $\Gamma.A_1 \dots A_n$ this context is equivalent to the context $\Gamma.\Sigma_{A_1} \dots \Sigma_{A_{n-1}} A_n$ of reduced length. By iterating this process, since an absolute type $\vdash B$ type is the same as a type over the terminal $\Delta_0 \vdash B[!_{\Delta^0}]$ type, we can translate any finite chain of dependent types into an absolute type by means of an equivalence of contexts. This justifies the idea that, in a tribe, we regard a local theory as the local tribe over an object. This object can independently come equipped with a sequence of fibrations (chain of dependent types) or with the terminal map (the iterated dependent sum of the chain), and we know that the theory over it is independent on it (up to equivalence, in our setting). More formally, in any tribe $\mathcal{E}$, for every $f : B \twoheadrightarrow A$, we intrinsically have an isomorphism of tribes $\mathcal{E}(A)(B, f) \cong \mathcal{E}(B)$, which expresses the equivalence of the theory over $A.B$ and of the theory over $\Sigma_A B$, although they are different contexts in principle. This can be iterated to any finite chain of types,

2.6.5 The Fundamental Theorem of the identity type

A second step towards the characterisation of the identity type of the dependent sum as the identity type of pairs is the so-called Fundamental Theorem of the identity type. Assume we have a type $\Gamma \vdash T$ type, and consider a $\Gamma, x : Y, y : T \vdash B(x, y)$ type. An interesting question is to determine whether there are any checkable properties characterising the identity types among all these B. Dependent sums allow us to determine a property to characterise B as Id_T. This property will be the fact that the dependent sum $\Sigma_{y:T} B(x, y)$ is a contractible type. The first step is to show that the identity type does indeed satisfy it.

Lemma 2.6.5.1. Let $\Gamma \vdash T$ type. Then, $\Gamma.T \vdash \Sigma_{T[p]} \mathrm{Id}_T$ type is a contractible type.

Proof. By Lemma 2.5.2.4, it suffices to check that the context $\Gamma.T.\Sigma_{T[p]} \mathrm{Id}_T$ is a contractible extension of $\Gamma.T$. By the rules of the dependent sum, we have an equivalence of contexts $p.\mathrm{pair} : \Gamma.T.T[p].\mathrm{Id}_T \to \Gamma.T.\Sigma_{T[p]} \mathrm{Id}_T$, and by the rules of the identity type we have an equivalence of contexts $\mathrm{id}.q.\mathrm{refl} : \Gamma.T \to \Gamma.T.T[p].\mathrm{Id}_T$. Since equivalences of contexts compose by Lemma 2.4.14.8, we get a composite equivalence of contexts $\mathrm{id}.\mathrm{pair}(q, \mathrm{refl}) : \Gamma.T \to \Gamma.T.\Sigma_{T[p]} \mathrm{Id}_T$ over $\Gamma.T$ as claimed. □

Now, as a second tool, we show that equivalences of dependent types can be checked on their dependent sums, i.e. dependent sum reflects equivalences. Semantically, this is the actual translation of the fact that fibrewise homotopy equivalences can be checked on the underlying arrows.

Lemma 2.6.5.2 (Σ_A reflects equivalences)**.** Let $\Gamma \vdash S$ type, and let $\Gamma.S \vdash A, B$ type. Consider a function $\Gamma.S.A \vdash f : B[p]$. Then, f is an equivalence over $\Gamma.S$ if and only if $\Gamma.\Sigma_S A \vdash \Sigma_S f : \Sigma_S B$ is an equivalence over Γ.

Proof. We have the following commutative square of context substitutions up to homotopy

$$\begin{array}{ccc} \Gamma.S.A & \xrightarrow{p.f} & \Gamma.S.B \\ \downarrow \wr & & \downarrow \wr \\ \Gamma.\Sigma_S A & \xrightarrow[p.\Sigma_A f]{} & \Gamma.\Sigma_S B \end{array} .$$

Being it a commutative square in $\mathrm{Ho}(\mathrm{Ctx}(\Gamma))$ with vertical isomorphisms, we have that the upper substitution is an equivalence over Γ if and only if the lower one is. In particular, by Proposition 2.4.14.9, we have that the upper substitution is an equivalence of contexts over Γ if and only if it is over $\Gamma.S$. By Remark 2.4.14.2, we can translate the equivalence of context into an equivalence of types, thus the claim. □

Theorem 2.6.5.3 (Fundamental Theorem of the identity type)**.** Let $\Gamma \vdash T$ type and let $\Gamma.T.T[p] \vdash B$ type. Consider a function $\Gamma.T.T[p].\mathrm{Id}_T \vdash f : B[p]$. If $\Gamma.T \vdash \Sigma_{T[p]} B$ type is contractible, then f is an equivalence.

Proof. By Lemma 2.6.5.1 and by the assumptions, we have equivalences of types $\Sigma_{T[p]} \mathrm{Id}_T \simeq \Delta^0_{\Gamma.T} \simeq \Sigma_{T[p]} B$ over $\Gamma.T$. Being $\Sigma_{T[p]} B$ contractible, by Lemma 2.5.2.5 this equivalence is homotopic to the function $\Gamma.T.\Sigma_{T[p]} \mathrm{Id}_P\, \Sigma_A f : (\Sigma_{T[p]} B)[p]$, which is thus an equivalence. By Lemma 2.6.5.2, we conclude that f is an equivalence of types $\mathrm{Id}_T \simeq B$ over the extended context $\Gamma.T.T[p]$. □

2.6.6 Identity in dependent sum

Our goal is now to characterise the identity type of the dependent sum by means of the identity type of pairs we introduced in Construction 2.6.4.1. The base changed type $\Gamma, s : \Sigma_{x:A} B(x), t : \Sigma_{y:A} B(y) \vdash \mathrm{Id}^{\mathrm{pair}}_{A.B}(\mathrm{pr}_1, \mathrm{pr}_2)(s,t) :\equiv \mathrm{Id}^{\mathrm{pair}}_{A.B}((\mathrm{pr}_1(s), \mathrm{pr}_2(s)), (\mathrm{pr}_1(t), \mathrm{pr}_2(t)))$ type is inhabited by pairs of an equality between $\mathrm{pr}_1(s)$ and $\mathrm{pr}_1(t)$ and an equality between the transport of $\mathrm{pr}_2(s)$ and $\mathrm{pr}_2(t)$. We want to provide an equivalence between $\mathrm{Id}^{\mathrm{pair}}_{A.B}(\mathrm{pr}_1, \mathrm{pr}_2)$ and $\mathrm{Id}_{\Sigma_A B}$.

Proposition 2.6.6.1. We have an equivalence of types $\mathrm{Id}_{\Sigma_A B} \simeq \mathrm{Id}^{\mathrm{pair}}_{A.B}(\mathrm{pr}_1, \mathrm{pr}_2)$ over $\Gamma.\Sigma_A B.(\Sigma_A B)[p]$.

Proof. First of all, let us show that $\Gamma.A.B \vdash \Sigma_{A[p]} \Sigma_{B[p]} \mathrm{Id}^{\mathrm{pair}}_{A.B}$ type is contractible. Indeed, the associated context $\Gamma.A.B.\Sigma_{A[p]} \Sigma_{B[p]} \mathrm{Id}^{\mathrm{pair}}_{A.B}$ type is equivalent to $\Gamma.A.B.A[p].B[p].\mathrm{Id}^{\mathrm{pair}}_{A.B}$, which is a contractible context extension of $\Gamma.A.B$ by construction. Thus, the base change $\Gamma.\Sigma_A B \vdash \mathrm{Id}^{\mathrm{pair}}_{A.B}(\mathrm{pr}_1, \mathrm{pr}_2)$ is contractible as well by Lemma 2.5.2.3. By Theorem 2.6.5.3 we conclude that $\mathrm{Id}^{\mathrm{pair}}_{A.B}(\mathrm{pr}_1, \mathrm{pr}_2) \simeq \mathrm{Id}_{\Sigma_A B}$ over $\Gamma.\Sigma_A B$. □

Proposition 2.6.6.1 finally determines equality in a dependent sum, as it is exactly the provision of an equality of the first projection and an equality between the second projections up to transport. Furthermore, this could be seen as a commutativity property of the dependent sum and the identity type, as we are saying that there is an equivalence of types $\Sigma_{\alpha : \mathrm{Id}_A(\mathrm{pr}_1(s), \mathrm{pr}_1(s'))} \mathrm{Id}_{B(\mathrm{pr}_1(s'))}(\mathrm{tr}_\alpha(\mathrm{pr}_2(s)), \mathrm{pr}_2(s')) \simeq \mathrm{Id}_{\Sigma_{x:A} B(x)}(s, s')$.

As a corollary, we now see that the canonical way of transporting terms between fibres of a dependent type over equalities maps every term into an "equal" terms. Let us be more precise: given $\Gamma.A \vdash B$ type and $\Gamma \vdash a, a'$ with a homotopy $\alpha : \mathrm{Id}_A(a, a')$, then the transport term of $b : B(a)$ lives in $B(a')$, and thus cannot be compared with b; however, if we sum along all the fibres $\Sigma_{x:A} B(x)$, we obtain two terms of the same type that can be compared. Unsurprisingly, transporting along an equality makes b change fibre, but maps it into a homotopic term when regarded inside the dependent sum.

Corollary 2.6.6.2 (Transport is homotopic to identity)**.** Let $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. We have a homotopy $\Gamma, a : A, a' : A, \alpha : \mathrm{Id}_A(a, a'), b : B(a) \vdash \mathrm{Id}_{\Sigma_A B}(\mathrm{pair}(a, b), \mathrm{pair}(a', \mathrm{tr}_\alpha(b))$.

Proof. By induction, we may assume $a' \equiv a$ and $\alpha \equiv \mathrm{refl}_a$. In such a case, we get a homotopy $\Gamma, a : A, b : B(a) \vdash \mathrm{pair}(\mathrm{refl}_a, \mathrm{refltr}(a, b)) : \mathrm{Id}^{\mathrm{pair}}((a, b), (a, \mathrm{tr}_{\mathrm{refl}_a}(b)))$, that by Proposition 2.6.6.1 determines an inhabitant of $\Gamma, a : A, b : B(a) \vdash \mathrm{Id}_{\Sigma_A B}(\mathrm{pair}(a, b), \mathrm{pair}(a, \mathrm{tr}_{\mathrm{refl}_a})(b))$ type. □

2.6.7 Full compatibility of dependent sum with equivalences

Now, we wish to generalise the compatibility of Σ with equivalences seen in Proposition 2.6.3.2. There we only showed that the functorial assignment Σ_A preserves equivalences. Instead, we now want to show that $\Sigma_A B$ is invariant under equivalence both in A and in B, in some sense. What we know a priori is that given an equivalence of contexts $\Gamma.A.B \simeq \Gamma.A'.B'$ we can conclude $\Gamma.\Sigma_A B \simeq \Sigma_{A'} B'$. However, we want a more explicit characterisation starting from assumptions we can check explicitly.

Remark 2.6.7.1. We can slightly modify (1) and (3) in Proposition 2.6.3.1 to provides a more general result, where instead of making only B vary, we allow A to vary as well. Let $\Gamma \vdash A, A'$ type with $\Gamma.A \vdash B$ type and $\Gamma.A' \vdash B'$ type. Consider functions $f : A \to A'$ over Γ and $g : B \to f^* B'$ over $\Gamma.A$. $\Gamma, s : \Sigma_{x:A} B(x) \vdash \Sigma_f g(s) :\equiv \mathrm{ind}_\Sigma(\mathrm{pair}(f(a), g(a, b))) : \Sigma_{x':A'} B'(x)$ from $\Sigma_A B$ to $\Sigma_{A'} B'$. This is compatible with composition and homotopy:

(a) For any other $\Gamma \vdash A''$ type with $\Gamma.A'' \vdash B''$ type, and a pair of functions $f' : A' \to A''$ and $g' : B' \to f'^* B''$, we may consider the function $f^*(g') : f^* B' \to f^* f'^* B''$ defined by base change of g' along f, i.e. $\Gamma, a : A, b' : B'(f(a)) \vdash f^*(g)(b) :\equiv g(f(b')) : B''(f'(f(a)))$. We have a homotopy between $\Sigma_{f' \circ f}(f^*(g') \circ g)$ and $\Sigma_{f'} g' \circ \Sigma_f g$ as functions $\Sigma_A B \to \Sigma_{A''} B''$. This follows by an analogous argument to the case with $A \equiv A' \equiv A''$ seen in Proposition 2.6.3.1.

(b) For any other functions $\Gamma.A \vdash \overline{f} : A'[p]$ and $\Gamma.A \vdash \overline{g} : (\overline{f}^* B)[p]$ equipped with homotopies $\Gamma.A \vdash \alpha : \mathrm{Id}_{A'[p]}(f, \overline{f})$, and $\Gamma.A \vdash \beta : \mathrm{Id}_{(\overline{f}^* B)[p]}(\mathrm{tr}_\alpha g, \overline{g})$, there is a homotopy between $\Sigma_f g$ and $\Sigma_{f'} g'$ as functions $\Sigma_A B \to \Sigma_{A'} B'$. Indeed, by induction principle it suffices to check over pairs that $\Sigma_f(g)(\mathrm{pair}(a, b))$ is homotopic to $\Sigma_{f'}(g')(\mathrm{pair}(a, b))$. By computation terms, these are homotopic to $\mathrm{pair}(f(a), g(b))$ and to $\mathrm{pair}(\overline{f}(a), \overline{g}(b))$ respectively. These are homotopic in $\Sigma_{A'} B'$ by Proposition 2.6.6.1, as the assumptions provide an equality of pairs.

Now that we have functoriality of dependent sum in both arguments and its compatibility with composition and equality, we can mimic the proof of Proposition 2.6.3.2 in the more general case of both A and B varying:

Proposition 2.6.7.2 (Σ is fully compatible with equivalences)**.** Let $\Gamma \vdash A, A'$ type and consider $\Gamma.A \vdash B$ type and $\Gamma.A' \vdash B'$ type. Assume there is an equivalence of types $\Gamma.A \vdash f : A'[p]$ over Γ and a further equivalence of types $\Gamma.A.B \vdash g : (f^* B')[p]$ over $\Gamma.A$. Then, the function $\Gamma.\Sigma_A B \vdash \Sigma_f(g) : (\Sigma_{A'} B')[p]$ is an equivalence of types.

Proof. Consider a left inverse f^{-1} for f and a left inverse g^{-1} for g. First of all, we can base change g^{-1} along f^{-1} to get $\Gamma, a' : A, b' : B'(f(f^{-1}(a')) \vdash g^{-1}(a', b') : B(f^{-1}(a'))$. This defines $\Gamma, a' : A, b' : B'(a') \vdash \overline{g^{-1}}(a') : B(f^{-1}(a'))$. In particular, we have a well-defined $\Sigma_{f^{-1}} \overline{g^{-1}} : \Sigma_{A'} B' \to \Sigma_A B$. We claim that this is a left inverse of $\Sigma_f g$. Indeed, by (1) in Remark 2.6.7.1, $\Sigma_{f^{-1}} g^{-1} \circ \Sigma_f g$ is homotopic to $\Sigma_{f^{-1} \circ f}(f^*(\overline{g}^{-1}) \circ g)$, and by construction $f^*(\overline{g}^{-1})$ is homotopic to g^{-1}. The claim then follows by (2) in Remark 2.6.7.1. □

We would like every type constructor we introduce to enjoy a property of this kind.

2.6.8 The adjunction $f_! \dashv f^*$

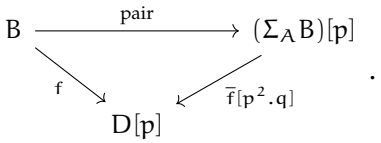
We want to exhibit the dependent sum as an adjoint. Semantically, the dependent sum via a fibration $f : B \twoheadrightarrow A$ is modelled via postcomposition with f, and this is the left adjoint of the pullback functor f^* between the local tribes. Syntactically, we have a distinction between fibrations and the underlying maps in nature. However, the adjunction is preserved, and translated into a "syntactic adjunction" between dependent sum and weakening.

Remark 2.6.8.1. A particular case of the induction principle of the dependent sum is when $E \equiv D[p]$ for some $\Gamma \vdash D$ type. The induction principle says that a function of types $\Gamma.A.B \vdash f : D[p^2]$ from B to $D[p]$ over $\Gamma.A$ induces a function of types $\Gamma.\Sigma_A B \vdash \mathrm{ind}_\Sigma(f) : D[p]$ from $\Sigma_A B$ to D over Γ. The computation rule says that this function, evaluated at some $\mathrm{pair}(a, b)$, is $f(b)$.

Proposition 2.6.8.2 ($f_! \dashv f^*$)**.** Let $\Gamma \vdash A, C$ type and let $\Gamma.A \vdash B$ type.

(1) We have a homotopy bijection between functions $\Sigma_A B \to C$ over Γ and functions $B \to C[p]$ over $\Gamma.A$.

(2) For any $\Gamma \vdash D$ type, for every function $\Gamma.B \vdash f : D[p^2]$ there exists a unique, up to equality, function $\Gamma.\Sigma_A B \vdash \bar{f} : D[p]$ such that the composite $\Gamma.B \vdash \bar{f}[p^2.q] \circ \mathrm{pair} : D[p]$ is equal to f, i.e. fitting into the following commutative triangle of functions of types, up to homotopy, over $\Gamma.A$:



In particular, we have an ordinary adjunction

$$\Sigma_A : \mathrm{Ho}(\mathrm{Type}_{/\Gamma.A}) \rightleftarrows \mathrm{Ho}(\mathrm{Type}_{/\Gamma}) : (-)[p] \equiv: w_A$$

between dependent sum and weakening at the level of the homotopy categories of types, with unit pair.

Proof. With the same argument in which one shows that the universal property of the unit determines an adjunction, (1) follows from (2). For the latter, Remark 2.6.8.1 determines the map $\mathrm{ind}_\Sigma(f) \equiv: \bar{f}$, and the computation rule, which determines its behaviour on pairs, exactly says that precomposing with the map pair gives f up to equality term. Uniqueness follows by the fact that every function out of a dependent sum is determined by its value on pairs by the induction principle. □

2.6.9 Fibre type

Dependent types $\Gamma, x : A \vdash B(x)$ type, up to now, could be seen as families of types, i.e. as functions of types from A to a universe type over Γ, although this is not representable in the theory. We now want to shift perspective, and regard dependent types as functions in a different way. Indeed, we will show that dependent types provide proper functions of types in the same way fibrations have an underlying arrow in a tribe. In the same way, every function of types may be promoted to a dependent type over the target up to equivalence, corresponding to the idea that, in a tribe, we can always factor an arrow as an equivalence followed by a fibration. This brings us closer to picturing the local theory in context Γ as the theory of (a full subcategory of) the "slice" over Γ. First of all, given a dependent type, let us define what the underlying *isofibration* map is.

Definition 2.6.9.1. Let $\vdash \Gamma$ ctx. Given $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type, the underlying *isofibration* is the function of types $\mathrm{pr}_1 : \Sigma_A B \to A$.

In other words, isofibrations are functions of types arising from a type dependence. A natural question is to understand when a map is an isofibration. The answer is always, up to equivalence, as we now see a canonical way of promoting a function to a type dependence whose underlying isofibration will coincide with the starting function up to equivalence.

Construction 2.6.9.2 (The fibre type)**.** Let $\Gamma \vdash A, B$ type and consider a function of types $\Gamma.B \vdash f : A[p]$. We define the *fibre type of* f as the dependent type $\Gamma, a : A \vdash \mathrm{fib}_f(a) :\equiv \Sigma_{b:B} \mathrm{Id}_A(f(b), a)$ type.

Indeed, the isofibration associated to the type of fibres of f is f itself, and conversely, the fibre type of the isofibration associated to a dependent type is equivalent to the dependent type itself. Let us state and prove this result, which hides more important consequence than it appears.

Proposition 2.6.9.3 (Isofibration and fibre type are inverse operations)**.** Let $\vdash \Gamma$ ctx.

(1) Let $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type, and consider the isofibration $\Gamma.\Sigma_A B \vdash \mathrm{pr}_1 : A[p]$. Then, we have an equivalence of types $B \simeq \mathrm{fib}_{\mathrm{pr}_1}$ over $\Gamma.A$.

(2) Let $\Gamma \vdash A, B$ type and consider a function $\Gamma.A \vdash f : B[p]$ type. Then, we have an equivalence $\varphi : \Sigma_A \mathrm{fib}_f \simeq A$ of types over Γ and a homotopy between $\mathrm{pr}_1 \circ \varphi$ and f.

Proof.

(1) Let us construct the two assignments. In one direction we have the function $\Gamma, a : A, b(a) : B(x) \vdash \mathrm{pair}(\mathrm{pair}(a, b(a)), \mathrm{pr}_1^{eq}) : \Sigma_{s(x):\Sigma_A B} \mathrm{Id}_A(\mathrm{pr}_1(s(a)), a)$. In the opposite direction, we induce one by the induction principle of the dependent sum bout of the assignment $\Gamma, a : A, s(a) : \Sigma_A B, \sigma(a, s(a)) : \mathrm{Id}_A(\mathrm{pr}_1(s(a)), a) \vdash \mathrm{tr}_{\sigma(a,s(a))}(\mathrm{pr}_2(s(a))) : B(a)$. The reader can check they provide an equivalence.

(2) Let us construct the map φ. By the iterated induction principle of dependent sums, it suffices to induce it on pairs, as the function $\Gamma, a : A, b : B, \sigma : \mathrm{Id}_A(f(b), a) \vdash b : B$. This is an equivalence of types as it has an inverse $\Gamma, b : B \vdash (f(b), b, \mathrm{refl}_{f(b)})$. In particular it lives over A as $\mathrm{fib}_{\mathrm{pr}_1}$ maps $(a : A, b : B, \sigma : \mathrm{Id}_A(f(b), a))$ into a. □

Remark 2.6.9.4. Let us explore the meaning of Proposition 2.6.9.3. (1) means that B is equivalent to the type of section of its associated isofibration $\Gamma, a : A \vdash \sigma(a) : \Sigma_{s(a):\Sigma_A B} \mathrm{Id}_A(\mathrm{pr}_1(s(a)), a)$. In particular, dependent terms of $\Gamma, a : A \vdash B(a)$ type are in correspondence with the sections of the isofibration $\mathrm{pr}_1 : \Sigma_A B \to A$, as intuition and semantics suggest. (2) means that every function of types is, up to equivalence, an isofibration, answering our question. For any map $f : A \to B$ in context Γ, we will refer to the factorisation $A \simeq \Sigma_A \mathrm{fib}_f \to B$ up to homotopy, seen in Proposition 2.6.9.3, as its *equivalence-isofibration* factorisation. We will use the same name whenever we have another factorisation equivalent to it via $X \simeq \mathrm{fib}_f$ over A for some $\Gamma.A \vdash X$ type. In particular, we can promote any function of types to a dependent type in a canonical way. Because dependent types have very well-behaved operations such as base change, it might be convenient sometimes. We will put in practice this construction various times.

Example 2.6.9.5. Let $\Gamma \vdash A$ type. The universal term $\Gamma, a : A \vdash a : A$ has fibre $\Gamma, a : A \vdash \Sigma_{a':A} \mathrm{Id}_A(a', a)$, that is contractible by Lemma 2.6.5.1. In particular, the identical function of A represent the dependent type $\Gamma.A \vdash \Delta^0_{\Gamma.A}$ type. This encodes the fact that identities are fibrations in a tribe.

Example 2.6.9.6. Let $\Gamma \vdash A$ type. then, the dependent type associated to the terminal function $!_A : A \to \Delta^0_\Gamma$ is $\Gamma.\Delta^0_\Gamma \vdash A[p]$. Indeed, $\Gamma, x : \Delta^0 \vdash \Sigma_{a:A} \mathrm{Id}_A(!_A(a), x)$ is the weakening of dependent sum over A of a contractible type by Remark 2.5.1.1, therefore it is equivalent to $A[p]$ by Example 2.6.9.5.

Taking Proposition 2.6.9.3 further, we now see how we can promote commutative triangles to functions between the fibres. This resembles the fact that, semantically, morphisms in the local tribe are commutative triangles.

Proposition 2.6.9.7 (Local functions and commutative triangles)**.** Let $\Gamma \vdash A, B, C$ type, and consider functions $f : A \to B$, $h : B \to C$, $g : A \to C$. If there is a homotopy $\Gamma, a : A \vdash \alpha(a) : \mathrm{Id}_C(h(f(a)), g(a))$ then there exists an induced function $f_{\mathrm{fib},g,h} : \mathrm{fib}_g \to \mathrm{fib}_h$ in context $\Gamma.C$, equipped with an inhabitant of $\Gamma, a : A \vdash \mathrm{Id}_C(\Sigma_C(f_{\mathrm{fib},g,h})(a), f(a))$ type.

Proof. We construct $f_{\mathrm{fib},g,h} : \mathrm{fib}_g \to \mathrm{fib}_h$ by induction principle as $\Gamma, c : C, a : A, \sigma : \mathrm{Id}_B(g(a), c) \vdash \mathrm{pair}(f(a), \sigma \circ \alpha(a)) \Sigma_{b:B} \mathrm{Id}_C(h(b), c)$. To show the desired homotopy, we notice that by Proposition 2.6.3.1 we may describe $\Sigma_C f_{\mathrm{fib},g,h} : \Sigma_C A \to \Sigma_C B$ as the assignment $\Gamma, c : C, a : A, \sigma : \mathrm{Id}_C(g(a), c) \vdash \mathrm{pair}(c, \mathrm{pair}(a, \sigma \circ \alpha(a))) : \Sigma_C \mathrm{fib}_h$. In particular, composed with $\Sigma_C \mathrm{fib}_h \to B$ gives $\Gamma, c : C, a : A, \sigma : \mathrm{Id}_C(g(a), c) \vdash f(a) : B$. □

The idea of isofibrations opens up for a new way of proving the contractibility of a type.

Proposition 2.6.9.8. Let $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. Then, B is contractible if and only if it is inhabited and the isofibration $\Sigma_A B \to A$ has a retraction.

Proof. B being inhabited provides a section of $\Sigma_A B \to A$ by Remark 2.6.9.4, so $\Sigma_A B \to A$ is an equivalence. By Proposition 2.6.10.1 this holds if and only if its fibre type, equivalent to B by Proposition 2.6.9.3, is contractible. □

Remark 2.6.9.9. This is interesting, because requiring $B \to \Delta^0_{\Gamma.A}$ to have a retraction is equivalent to B being contractible. The reason was that B must be inhabited to satisfy this condition. However, requiring that the image under Σ_A, that is $\Sigma_A B \to A$, has a retraction, is a weaker condition, as for instance it does not imply that B is inhabited. Thus, in order for it to imply contractibility, it must be paired with B being inhabited.

As for any type we construct, we are interested in spelling out the identity type of a fibre type.

Proposition 2.6.9.10 (Identity in the fibre type)**.** Let $\Gamma \vdash A, B$ type and $\Gamma.A \vdash f : B[p]$. Then, $\Gamma, b : B, b, a : A, a' : A, \sigma : \mathrm{Id}_B(f(a), b), \sigma' : \mathrm{Id}_A(f(a'), b) \vdash \mathrm{Id}_{\mathrm{fib}_f}(\mathrm{pair}(a, \sigma), \mathrm{pair}(a', \sigma'))$ type is equivalent to $\Gamma, b : B, b, a : A, a' : A, \sigma : \mathrm{Id}_B(f(a), b), \sigma' : \mathrm{Id}_A(f(a'), b) \vdash \Sigma_{\tau:\mathrm{Id}_A(a,a')} \mathrm{Id}_{\mathrm{Id}_B(f(a'),b)}(\sigma \circ \mathrm{ap}_f(\tau), \sigma')$ type.

Proof. By Proposition 2.6.6.1, we write $\mathrm{Id}_{\mathrm{fib}_f}(\mathrm{pair}(a, \sigma), \mathrm{pair}(a', \sigma'))$ as $\Sigma_{\tau:\mathrm{Id}_A(a,a')} \mathrm{Id}_{\mathrm{Id}_B(f(a),b)}(\mathrm{tr}_\tau(\sigma), \sigma')$, by the identifications in Proposition 2.6.2.1. Now, by Lemma 2.4.8.3 we can describe the transport term and rewrite this as $\Sigma_{\tau:\mathrm{Id}_A(a,a')} \mathrm{Id}_{\mathrm{Id}_B(f(a'),b)}(\sigma \circ \mathrm{ap}_f(\tau), \sigma')$. □

In particular, by Proposition 2.6.2.1, a homotopy in the fibre type of $f : A \to B$ between (a, σ) and (a', σ') consists of a homotopy $\tau : \mathrm{Id}_A(a, a')$ in A and a homotopy between $\sigma \circ \mathrm{ap}_f(\tau)$ and σ' as equalities from $f(a')$ to b.

Convention. To avoid cluttering of notation, we may write a context of the form $\Gamma, (a, b) : \Sigma_{x:A} B(x)$ in the following sense:

(1) When the judgement is of the form $\Gamma, (a, b) : \Sigma_{x:A} B(x) \vdash E(a, b)$ type, this will stand for $\Gamma, a : A, b : B(a) \vdash f(a, b) : E(\mathrm{pair}(a, b))$ type, to remind ourselves that any term constructed this way induces a term of $\Gamma, s : \Sigma_{x:A} B(x) \vdash E(s)$ type. For instance, given $\Gamma \vdash C, D$ type with a function $\Gamma.C \vdash h : D[p]$, writing $\Gamma, d : D, (c, \sigma) : \mathrm{fib}_h(c) \vdash E(d, c, \sigma)$ will stand for $\Gamma, d : D, c : C, \sigma : \mathrm{Id}_D(f(c), d) \vdash E(d, c, \sigma)$ type.

(2) When the judgement is of the form $\Gamma, (a, b) : \Sigma_{x:A} B(x) \vdash t(a, b) : E(a, b)$ type, this notation will stand for the induced term $\Gamma, s : \Sigma_{x:A} B(x) \vdash \mathrm{ind}_\Sigma(t) : E(s)$ type. For instance, given $\Gamma \vdash C, D$ type with a function $\Gamma.C \vdash h : D[p]$, a judgement $\Gamma, d : D, (c, \sigma) : \mathrm{fib}_h(c) \vdash t(d, c, \sigma) : E(d, c, \sigma)$ will stand for the induced term $\Gamma, d : D, x : \mathrm{fib}_h(d) \vdash E(d, \mathrm{pr}_1(x), \mathrm{pr}_2(x))$ over $\Gamma, d : D, c : C, \sigma : \mathrm{Id}_D(f(c), d)$.

This requires some attention: by constructing a term over $\Gamma, (a, b) : \Sigma_{x:A} B(x)$ we induce an actual term over the dependent sum that coincides with the term at $\mathrm{pair}(a, b)$ only up to homotopy. Despite the fact that we will avoid this convention for this section, later on it will be convenient to use it.

2.6.10 Equivalences as dependent types

We saw that we can factorise any function into an equivalence followed by an isofibration, that is, a map encoding a type dependence. We can now wonder how properties of functions, such as being an equivalence, get translated by regarding them as dependent types. Because equivalences resemble identities, and the identity of A can be promoted to a dependent type over A by considering the terminal type over A by Example 2.6.9.5, we expect to be able to characterise equivalences by saying the corresponding fibre type is contractible. Indeed, the fibre at each term shall be imagined as the preimage, and being an equivalence should intuitively mean that every fibre has a unique inhabitant. This intuition is correct, but the proof in unexpectedly trickier than one might expect.

Proposition 2.6.10.1. Let $\Gamma \vdash A, B$ type and consider a function of types $\Gamma.A \vdash f : B[p]$. Then, f is an equivalence of types if and only if $\Gamma.B \vdash \mathrm{fib}_f$ type is contractible.

Proof.

($\Rightarrow$) The first step is to notice that f is an equivalence if and only if there is a function $\Gamma.B \vdash g : A[p]$ equipped with homotopies $\Gamma, a : A \vdash \eta(a) : \mathrm{Id}_A(g(f(a)), a)$ and $\Gamma, b : B \vdash \varepsilon(b) : \mathrm{Id}_B(f(g(b)), b)$ by Remark 2.4.11.4. Now, the argument in [BGL+17] consists of showing that we can replace these data of having an inverse function, to the data of being a right adjoint equivalence. This means that the componentwise inversion of η and a replacement of ε satisfy one of the triangle identities of an adjunction. Because, later on, this proof will be lifted to an equivalence of types, but a type of "being an half adjoint equivalence" will not exist in our setting, we instead provide a more straightforward argument inspired to [Esc19]. We will show that the data of having an inverse g with homotopies η and ε gives rise to a proof that $\Gamma.B \vdash \mathrm{fib}_f$ type is contractible, and the key idea will be to rely on Lemma 2.5.2.7. Consider an inverse g equipped with $\Gamma, a : A \vdash \eta(a) : \mathrm{Id}_A(gf(a), a)$ and $\Gamma, b : B \vdash \varepsilon(b) : \mathrm{Id}_B(f(g(b)), b)$.

(a) We can construct $\Gamma, b_0 : B, a : A, \alpha : \mathrm{Id}(f(a), b_0) \vdash (f(a), \alpha \circ \mathrm{ap}_f(\eta(a))) : \Sigma_{b:B} \mathrm{Id}(f(g(b)), b_0)$, as we can postcompose $\mathrm{ap}_f(\eta(a))$, which is a homotoy from $f(g(f(a)))$ to $f(a)$, with α, which is a homotopy from $f(a)$ to b_0. By induction principle, this yields $\Gamma, b_0 : B, (a, \alpha) : \Sigma_{a:A} \mathrm{Id}(f(a), b_0) \vdash (f(a), \alpha \circ \mathrm{ap}_f(\eta(a))) : \Sigma_{b:B} \mathrm{Id}(f(g(b)), b_0)$. We claim that this exhibits $\Sigma_{a:A} \mathrm{Id}(f(a), b_0)$ as a retract of $\Sigma_{b:B} \mathrm{Id}(f(g(b)), b_0)$, i.e. that it admits a retraction. Indeed, we can easily construct the map $\Gamma, b_0 : B, b : B, \beta : \mathrm{Id}(f(g(b)), b_0) \vdash (g(b), \beta) : \Sigma_{a:A} \mathrm{Id}(f(a), b_0)$, which by induction principle induces a function in the opposite direction, and we can prove it is a retraction. Indeed, by Remark 2.6.7.1, the composite is the function $\Gamma, b_0 : B, (a, \alpha) : \Sigma_{a:A} \mathrm{Id}(f(a), b_0) \vdash (g(f(a)), \alpha \circ \mathrm{ap}_f(\eta(a))) : \Sigma_{a:A} \mathrm{Id}(f(a), b_0)$. We show it is homotopic to the identical assignment by Proposition 2.6.6.1. The equality $\eta(a)^{-1} : \mathrm{Id}_A(a, g(f(a)))$ equates the first components. By Lemma 2.4.8.3, we know that transport along $\eta(a)^{-1}$ in the base change of the identity type along f is $\mathrm{tr}_{\eta(a)^{-1}}(\alpha) \equiv \alpha \circ \mathrm{ap}_f(\eta(a))$. Therefore, this induces a homotopy of pairs between (a, α) and $(g(f(a)), \alpha \circ \mathrm{ap}_f(\eta(a)))$. Hence, by Proposition 2.6.6.1, a homotopy in $\Sigma_{a:A} \mathrm{Id}(f(a), b_0)$. In conclusion, we exhibited $\mathrm{Id}(f(a), b_0)$ as a retract of $\Sigma_{b:B} \mathrm{Id}(f(g(b)), b_0)$.

(b) Now, ε induces an equivalence of types $\Gamma, b_0 : B, b : B, \beta : \mathrm{Id}_B(f(g(b)), b_0) \vdash \beta \circ \varepsilon(b) : \mathrm{Id}_B(b, b_0)$, with inverse given by precomposing with $\varepsilon(b)^{-1}$ by Proposition 2.4.7.3. In particular, by Proposition 2.6.3.1 and Proposition 2.6.3.2, we get an equivalence of types from $\Sigma_{b:B} \mathrm{Id}_B(f(g(b)), b_0)$ to $\Sigma_{b:B} \mathrm{Id}_B(b, b_0)$ over $\Gamma, b_0 : B$. In particular, this exhibits the first as retract of the second.

Since retracts trivially compose, we can put together (a) and (b) to get that, over $\Gamma, b_0 : B$, the type $\Sigma_{a:A} \mathrm{Id}(f(a), b_0)$ is a retract of $\Sigma_{b:B} \mathrm{Id}_B(b, b_0)$. By Lemma 2.6.5.1, we know that the latter is contractible. Therefore, by Lemma 2.5.2.7, we conclude that $\Sigma_{a:A} \mathrm{Id}(f(a), b_0) \equiv \mathrm{fib}_f(b_0)$ is contractible.

($\Leftarrow$) We assume that $\Gamma, b : B \vdash \mathrm{fib}_f(b)$ type is contractible. In particular, we construct an inverse using the fact that it is inhabited by some term $\Gamma, b : B \vdash \mathrm{contr}_f(b) : \mathrm{fib}_f(b)$ type: we have $\Gamma, b : B \vdash g(b) :\equiv \mathrm{pr}_1(\mathrm{contr}_f(b)) : A$. This comes equipped with $\Gamma, b : B \vdash g(b) :\equiv \mathrm{pr}_2(\mathrm{contr}_f(b)) : \mathrm{Id}_B(f(g(b)), b)$, that exhibits g as a section of f. We now use contractibility to show it is a retraction as well. Indeed, by contractibility there is a witness of $\Gamma, b : B, (x, \alpha) : \mathrm{fib}_f(b) \vdash \mathrm{Id}_{\mathrm{fib}_f(b)}((x, \alpha), \mathrm{contr}_f(b))$. By Proposition 2.6.9.10, the first projection provides a term of $\Gamma, b : B, (x, \alpha) : \mathrm{fib}_f(b) \vdash \mathrm{Id}^{\mathrm{pair}}(x, g(b))$ type. Applying the base change along $a : A \mapsto (f(a), a, \mathrm{refl}_{f(a)})$, this yields an inhabitant of $\Gamma, a : A \vdash \mathrm{Id}_A(a, g(f(a)))$ type, which exhibits g as a retraction of f. □

2.6.11 Dependent sum for the construction of statements

Let us switch again to the point of view of homotopy type theory, where all types are in fact statements, with their terms corresponding to their proofs. The introduction of the dependent sum opens the way to translate a predicate onto a type into a statement itself, which is the idea of quantifiers. Hence we will need to interpret a type not only as a statement, but as a domain to quantify over. The dependent sum $\Gamma \vdash \Sigma_{a:A} B(a)$ corresponds to the statement whose proofs are the exhibition of a term $a : A$ and a term $b : B(a)$, that is, a proof that a satisfies $B(a)$. In other words, $\Sigma_{a:A} B(a)$ looks naively like the statement "there exists a such that $B(a)$", with the difference that a witness requires the exhibition of some a. The actual $\exists$ quantifier in type theory can be constructed out of the dependent sum and truncation, as one can find in [BGL+17], but for the matters of this project we will leave this discussion aside. Instead, we think of the dependent sum to have a role similar to the $\exists$ quantifier for the construction of statements. Let us give an explicit example with the few types we already have. Given $\Gamma \vdash A$ type and $\Gamma \vdash c : A$, the type $\Gamma \vdash \Sigma_{a:A} \mathrm{Id}_A(a, c)$ type is close to the statement "there exists $a : A$ such that a is equal to c", with the difference that a witness consists of the *exhibition* of an a such that a and $\alpha : \mathrm{Id}_A(a, c)$.

2.7 Product of types

We now focus our attention onto a particular case of the dependent sum, namely the product of two types. Semantically, it is no surprise, as the weakening $\Gamma \vdash B$ type at $\Gamma \vdash A$ type is exactly a (fibre) product equipped with the projection into the context $\Gamma.A$. The only way product differs from weakening is that it

regards the two arguments symmetrically and lives over the common context Γ. Its terms are then going to be a pair of a term of $\Gamma \vdash A$ type and a term of $\Gamma \vdash B$ type.

Construction 2.7.0.1. Let $\Gamma \vdash A, B$ type. We define the *product of* A *and* B to be $\Gamma \vdash A \times B :\equiv \Sigma_A B[p]$ type. This is compatible with base change, and comes equipped with:

(1) A function $\Gamma.A.B \vdash \mathrm{pair} : (A \times B)[p]$ that takes $\Gamma \vdash a : A$ and $\Gamma \vdash b : B$ into a term $\Gamma \vdash \mathrm{pair}(a, b) : A \times B$.

(2) The *projection maps*, given by functions of types $\Gamma.A \times B \vdash \mathrm{pr}_1 : A[p]$ and $\Gamma.A \times B \vdash \mathrm{pr}_2 : B[p]$.

These are once again compatible with substitution.

We might denote $\mathrm{pair}(a, b)$ as (a, b) resembling the categorical notation for the product. Note that the projection map $\mathrm{pr}_1 : A \times B \to A$ is, by definition, the isofibration associated to the weakening $\Gamma.B \vdash A[p]$ type. This construction inherits the properties of the dependent sums: in particular we can characterise every term of the product as a pair of its projections. Furthermore, we know that we can equate the projections of $\mathrm{pair}(a, b)$ to a and b respectively, as well as $\mathrm{pair}(\mathrm{pr}_1(s), \mathrm{pr}_2(s))$ to s.

Remark 2.7.0.2. Given $\Gamma \vdash A, B$ type, we have an equivalence of type $A \times B \simeq B \times A$ over Γ. Furthermore, from the general discussion about the dependent sum, we inherit an equivalence of contexts $\Gamma.A \times B \simeq \Gamma.A.B[p] \simeq \Gamma.B.A[p]$ with "the same theory" over them. In particular, from now on we can picture the identity type of A as $\Gamma.A \times A \vdash \mathrm{Id}_A$ up to equivalence.

Note that, despite our notation of the product as $\times$, this lives over a fixed context Γ, so it should be regarded as a fibre product over Γ instead. Semantically, this is the product in the local tribe over Γ. The reason why we denote it as a product rather than a pullback over Γ, is that we will also introduce the notion of homotopy pullback in context Γ, which is the homotopy pullback in the local tribe over Γ, between two functions of types in context Γ. This will be denoted with the usual notation of the pullback $\times_C$, to remark that C is itself a type in context Γ different from the point.

Proposition 2.7.0.3 (Identity in products)**.** Let $\Gamma \vdash A, B$ type. Then, we have an equivalence of types $\mathrm{Id}_{A\times B} \simeq \mathrm{Id}_A(\mathrm{pr}_1(q[p]), \mathrm{pr}_1(q)) \times \mathrm{Id}_B(\mathrm{pr}_2(q[p]), \mathrm{pr}_2(q))$ over $\Gamma.A \times B.(A \times B)[p]$.

Proof. Follows by Proposition 2.6.6.1, as by definition we have $\mathrm{Id}^{\mathrm{pair}}_{A.B[p]}((\mathrm{pr}_1(s), \mathrm{pr}_1(s')), (\mathrm{pr}_2(s), \mathrm{pr}_2(s'))) \equiv \Sigma_{\alpha:\mathrm{Id}_A(\mathrm{pr}_1(s),\mathrm{pr}_1(s'))} \mathrm{Id}_B(\mathrm{pr}_2(s), \mathrm{pr}_2(s')) = \mathrm{Id}_A(\mathrm{pr}_1(s), \mathrm{pr}_1(s')) \times \mathrm{Id}_B(\mathrm{pr}_2(s), \mathrm{pr}_2(s'))$ since transport in a weakening is trivial. □

In other words, inside a product, equalities are given componentwise. Hence, given $\alpha : \mathrm{Id}_A(a, a')$ and $\beta : \mathrm{Id}_B(b, b')$, we say that $(\alpha, \beta) : \mathrm{Id}_{A\times B}((a, b), (a', b'))$ is the *equality induced on the pair* by α and β. Conversely, an equality $\gamma : \mathrm{Id}_{A\times B}((a, b), (a', b'))$ consists of equalities $\gamma_1 : \mathrm{Id}_A(a, a')$ and $\gamma_2 : \mathrm{Id}_B(b, b')$, and we might refer to these as the "equalities induced on the projections" by γ.

Remark 2.7.0.4 (Universal property of the product)**.** Let $\Gamma \vdash A, B, C$ type. Then, for every pair of functions $\Gamma.C \vdash f : A[p]$ and $\Gamma.C \vdash g : B[p]$ there is an induced function $\Gamma.C \vdash (f, g) : (A \times B)[p]$ such that $\mathrm{pr}_1 \circ (f, g)$ is homotopic to f and $\mathrm{pr}_2 \circ h$ is homotopic to g. For any other function of types $\Gamma.C \vdash h : (A \times B)[p]$ such that $\mathrm{pr}_1 \circ h$ is homotopic to f and $\mathrm{pr}_2 \circ h$ is homotopic to g, there is a homotopy between (f, g) and h that is induced by the equalities on the projections.

This is a desired property, as we want products to become product of synthetic categories later on, and thus must come equipped with the expected universal property.

Lemma 2.7.0.5. Let $\Gamma \vdash A$ type. The projection $\mathrm{pr}_1 : A \times \Delta^0_\Gamma \to A$ is an equivalence.

Proof. Consider the assignment $\Gamma, a : A \vdash \mathrm{pair}(a, *_\Gamma) : A \times \Delta^0_\Gamma$. This is clearly a section by Proposition 2.6.2.1, and is a retraction by the induction principle of the product and elimination rule of Δ^0_Γ, since $\mathrm{pair}(\mathrm{pr}_1(\mathrm{pair}(a, *_\Gamma)), *_\Gamma)$ is homotopic to $(a, *_\Gamma)$. □

Lastly, we introduce some constructions with the product that will be particularly useful later on.

Definition 2.7.0.6 (Product of functions)**.** Let $\Gamma \vdash A, B, C, D$ type and $\vdash f : A \to C$ and $\vdash g : B \to D$. We define the map $f \times g :\equiv (f \circ \mathrm{pr}_1, g \circ \mathrm{pr}_2) : A \times B \to C \times D$.

Definition 2.7.0.7 (The diagonal map)**.** Let $\Gamma \vdash A$ type. The *diagonal map of* A is the function of types $\Delta_A :\equiv (1_A, 1_A) : A \to A \times A$ over Γ.

Remark 2.7.0.8 (Fibre type of the diagonal)**.** Let $\Gamma \vdash A$ type. We can regard Id_A as a dependent type $\Gamma.A \times A \vdash \mathrm{Id}_A$ type being $\Gamma.A.A[p] \simeq \Gamma.A \times A$. This is interesting because the fibre type $\Gamma.A \times A \vdash \mathrm{fib}_{\Delta_A}$ type of the diagonal map $\Delta : A \to A \times A$ is equivalent to this version of the identity type.

Remark 2.7.0.9. Let $\Gamma \vdash A, B$ type. If we think of A and B as statements in homotopy type theory, the product $\Gamma \vdash A \times B$ type is the statement corresponding to "A and B". Indeed, a proof of $A \times B$ corresponds to a pair of a proof of A and a proof of B, meaning that the two hold simultaneously.

2.8 Homotopy pullbacks

Finally, we want to introduce one of the most important constructions in category theory: homotopy pullbacks. The reader might notice that we already encoded the pullback construction in the base change. However, even in the semantics, homotopy pullbacks have a different role in their nature. On the one hand, base change is an operation of change of context along some substitution, and is in fact the pullback of a fibration. The input data are a dependent type $A \twoheadrightarrow \Gamma$ and a substitution $\gamma : \Delta \to \Gamma$, and the outcome is a dependent type $A[\gamma] \twoheadrightarrow \Delta$, of which we can study the terms, and we do not really care about having a commutative pullback square. Homotopy pullbacks, instead, are introduced very differently as squares inside a fixed context. More precisely, the input data will be two functions of types in a fixed context Γ and with the same target. Out of these, we construct another type over Γ providing a commutative square over the starting cospan. Since everything is introduced in the same context, we can manipulate the outcome and study its behaviour with respect to other maps in context Γ. This is, in some sense, a generalisation of the difference in nature between the product and the weakening. However, we will also prove that base change and homotopy pullback are both manifestations of a pullback phenomenon, that allows us to carry the properties of one point of view to the other.

2.8.1 Construction of homotopy pullbacks

Construction 2.8.1.1. Let $\Gamma \vdash A, B, C$ type and consider two maps $\Gamma.A \vdash f : C[p]$ and $\Gamma.B \vdash g : C[p]$. We consider the context substitution $p^2.f[p].g[p^2.q] : \Gamma.A.B[p] \to \Gamma.C.C[p]$, and consider the base change of the identity type $\Gamma.A.B[p] \vdash \mathrm{Id}_C[p^2.f[p].g[p^2.q]]$. This is just the reparametrisation of the identity type $\Gamma, x : A, y : B \vdash \mathrm{Id}_C(f(x), g(y))$. Now we can take the dependent sum twice, and define

$$\Gamma \vdash A \times_C B \equiv \Sigma_B \Sigma_{B[p]} \mathrm{Id}_C[p^2.f[p].g[p^2.q]]$$

called the *homotopy pullback of* f *and* g. As we notice, it is a type in the same context as A, B and C. More explicitly, it is the type $\Gamma \vdash \Sigma_{x:A}\Sigma_{y:B} \mathrm{Id}_C(f(x), g(y))$, which we can write as $\Gamma \vdash A \times_B C \equiv \{x : A, y : B, \sigma : \mathrm{Id}_C(f(x), g(y))\}$ as a collection of its terms. The universal term of $A \times_C B$ provides canonical functions $\Gamma.A \times_C B \vdash \mathrm{pr}_1 : A[p]$ type and $\Gamma.A \times_C B \vdash B[p]$ over Γ, together with an equality term between $f \circ \mathrm{pr}_1$ and $g \circ \mathrm{pr}_2$. These send a term $(a : A, b : B, \sigma : \mathrm{Id}_C(f(a), g(b)))$ to $a : A$ and $b : B$ respectively, and go under the name of *projections* in analogy with the previous denominations. The given homotopy means that these functions fit into the following commutative square up to homotopy in context Γ:

$$\begin{array}{ccc} A \times_C B & \xrightarrow{\mathrm{pr}_1} & A \\ {\scriptstyle \mathrm{pr}_2}\downarrow & & \downarrow{\scriptstyle f} \\ B & \xrightarrow[g]{} & C \end{array}.$$

Note that, by means of the automorphism of the context $\Gamma.C.C[p]$ swapping the two copies of C, we get an equivalence of types $A \times_C B \simeq B \times_C A$ over Γ.

Remark 2.8.1.2 (Identity in homotopy pullbacks)**.** Given two terms $(a : A, b : B, \sigma : \mathrm{Id}_C(f(a), g(b)))$ and $(a' : A, b' : B, \sigma' : \mathrm{Id}_C(f(a'), g(b')))$ of $A \times_C B$, an equality between them consists of the following data:

(1) an equality $\alpha : \mathrm{Id}_A(a, a')$, inducing $\mathrm{ap}_f(\alpha) : \mathrm{Id}_C(f(a), f(a'))$

(2) an equality $\beta : \mathrm{Id}_B(b, b')$, inducing $\mathrm{ap}_g(\beta) : \mathrm{Id}_C(g(b), g(b'))$,

(3) an equality $\varphi : \mathrm{Id}_{\mathrm{Id}_C(f(a'),g(b'))}(\mathrm{ap}_g(\beta) \circ (\sigma \circ \mathrm{ap}_f(\alpha)^{-1}), \sigma')$ by Lemma 2.4.8.3.

Given its name, we expect to be able to equip the homotopy pullback with its universal property, as this will be the pullback construction of categories. This is an immediate consequence of the definition:

Remark 2.8.1.3 (Universal property of the homotopy pullback)**.** Let $\Gamma \vdash A, B, C$ type. Given two functions $\alpha : V \to A$ and $\beta : V \to B$ over Γ with a homotopy $\Gamma.V \vdash \alpha : \mathrm{Id}_{C[p]}(f \circ \alpha, g \circ \beta)$, we get an induced function of types $\Gamma.V \vdash (\alpha, \beta) : (A \times_C B)[p]$ given by the triple $(\alpha : A[p], \beta : B[p], \alpha : \mathrm{Id}_{C[p]}(f(\alpha), g(\beta))$. Conversely, every other function $\Gamma.V \vdash h : (A \times_C B)[p]$ that determines (by composition with the projections) α and β up to homotopy is homotopic to (α, β).

Remark 2.8.1.4. Homotopy pullback is compatible with homotopy. More precisely, given $\Gamma \vdash A, B, C$ type with functions $\Gamma.A \vdash f, f' : C[p]$ and $\Gamma.B \vdash g : C[p]$, and a homotopy $\Gamma.A \vdash \alpha : \mathrm{Id}_{C[p]}(f, f')$, there is an induced equivalence of types between the homotopy pullbacks of g with f and f'.

Furthermore, homotopy pullback is compatible with equivalences. More precisely, let $\Gamma \vdash A, A', B, C$ type and consider functions $\Gamma.A \vdash f : C[p]$, $\Gamma.A' \vdash f' : C[p]$ and $\Gamma.B \vdash g : C[p]$. Assume there is an equivalence of types $\varphi : A \xrightarrow{\sim} A'$ such that $f' \circ \varphi$ is homotopic to f. Then, we get an induced equivalence $(\varphi, 1_B) : A \times_C B \simeq A' \times_C B$ such that $\mathrm{pr}_2 \circ (\varphi, 1_B)$ is homotopic to 1_B.

2.8.2 Homotopy cartesian squares

Remark 2.8.2.1. Let $\Gamma \vdash A, B, C$ type with two maps $\Gamma.A \vdash f : C[p]$ and $\Gamma.B \vdash g : C[p]$. We can regard a commutative square

$$\begin{array}{ccc} D & \xrightarrow{\alpha} & A \\ {\scriptstyle\beta}\downarrow & & \downarrow{\scriptstyle f} \\ B & \xrightarrow[g]{} & C \end{array}$$

equivalently as a function $\Gamma.D \vdash (\alpha, \beta) : (A \times_C B)[p]$ type by the universal property of the homotopy pullback.

Definition 2.8.2.2. Let $\Gamma \vdash A, B, C$ type with functions $\Gamma.A \vdash f : C[p]$ and $\Gamma.B \vdash g : C[p]$. A *homotopy cartesian square*, or *a homotopy pullback square*, (with vertex $\Gamma \vdash D$ type) is a commutative square $\Gamma.D \vdash (\alpha, \beta) : (A \times_C B)[p]$ type such that this function is an equivalence. Equivalently, given a commutative square, i.e. a homotopy between the composites in

$$\begin{array}{ccc} D & \xrightarrow{\alpha} & A \\ {\scriptstyle\beta}\downarrow & & \downarrow{\scriptstyle f}, \\ B & \xrightarrow[g]{} & C \end{array}$$

this is a homotopy pullback if the induced function $D \to A \times_C B$ is an equivalence of types.

Remark 2.8.2.3. In other words, a square as above is a homotopy pullback if the terms of D are in homotopy bijection, compatibly with base change, with the terms of $A \times_C B$, namely the triples $(a : A, b : B, \sigma : \mathrm{Id}_A(f(a), g(b)))$, and α and β map this triple to a and b respectively up to homotopy.

We can actually regard the homotopy pullback as a choice of a specific homotopy cartesian square. Since we do not care about strict choices but we want to work up to equivalence, we will be interested in manipulating homotopy cartesian squares: the homotopy pullback will just be regarded as one of them, useful to exhibit constructions. For this reason, from now on, we will treat the two indifferently, and we will refer to a homotopy cartesian squares as *a homotopy pullback*. This idea is supported by the fact that a homotopy cartesian square has the same universal property as the homotopy pullback, up to homotopy.

Remark 2.8.2.4 (Flipping)**.** It is clear from the definition that homotopy cartesian squares have a flipping property, namely the fact that the left square is homotopy cartesian if and only if the right square is:

$$\begin{array}{ccc} D & \xrightarrow{\alpha} & A \\ {\scriptstyle\beta}\downarrow & & \downarrow{\scriptstyle f} \\ B & \xrightarrow[g]{} & C \end{array} \qquad \begin{array}{ccc} D & \xrightarrow{\beta} & B \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array}.$$

2.8.3 Homotopy pullback and base change

We now explore a link between two construction which are the manifestation of the same categorical concept, but which were introduced very differently in the syntax: base change and homotopy pullbacks. For the reader familiar with the semantics of dependent type theory, this is not surprising, as the base change operation mimics the pullback of a fibration along a map, giving an operation of change of context, whereas the homotopy pullback is the pullback taking place in a fixed context. In particular, it is performed by factorising one map into an equivalence followed by a fibration and then computing the pullback of the obtained fibration. These two processes should be the same up to equivalence, and represent the two natures of pullback. Knowing this bridge will allow us to transport the properties of one interpretation, for instance the well-understood base change, onto the other one.

Theorem 2.8.3.1 (Base change is pullback)**.** Let $\vdash \Gamma\,\mathrm{ctx}$ and let $\Gamma \vdash A, B, P, Q$ type. Then, the commutative square

$$\begin{array}{ccc} D & \xrightarrow{f'} & C \\ {\scriptstyle g'}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & B \end{array}$$

is homotopy cartesian if and only if there is an equivalence of types $\mathrm{fib}_g \simeq f^*\mathrm{fib}_{g'}$ over $\Gamma.A$.

In particular, for any dependent type $\Gamma.B \vdash E$ type, the following square is homotopy cartesian

$$\begin{array}{ccc} \Sigma_A f^*(E) & \longrightarrow & \Sigma_B E \\ {\scriptstyle \mathrm{pr}_1}\downarrow & & \downarrow{\scriptstyle \mathrm{pr}_1} \\ A & \xrightarrow[f]{} & B \end{array}.$$

Proof. Let $\Gamma.D \vdash \mathrm{comm} : \mathrm{Id}_{B[p]}(g\circ f', f\circ g')$. First of all, we observe that we can write the homotopy pullback $A \times_B C$ as $\Sigma_A f^*\mathrm{fib}_g$ by definition. Thus, the square is homotopy cartesian if and only if the induced map $D \to \Sigma_A f^*\mathrm{fib}_g$ is an equivalence. We want to induce it as the dependent sum of a function $\mathrm{fib}_{g'} \to f^*\mathrm{fib}_g$ by functoriality, under the equivalence $D \simeq \Sigma_A \mathrm{fib}_{g'}$ seen in Proposition 2.6.9.3. We can construct such a function via induction principle from $\Gamma, a : A, d : D, \sigma : \mathrm{Id}_A(g'(d), a) \vdash \mathrm{pair}(f'(d), \mathrm{ap}_f(\sigma)\circ\mathrm{comm}) : f^*\mathrm{fib}_g \equiv \Sigma_{c:C}\,\mathrm{Id}_B(g(c), f(a))$. By construction, the induced function $\Sigma_A\mathrm{fib}_{g'} \to \Sigma_A f^*\mathrm{fib}_g$ precomposed with the equivalence $D \simeq \Sigma_A\mathrm{fib}_{g'}$ from Proposition 2.6.9.3 is homotopic to the canonical map $D \to A \times_B C \equiv \Sigma_A f^*\mathrm{fib}_g$. Thus, by 2-out-of-3 for equivalences seen in Lemma 2.4.11.5, we get that the square is homotopy cartesian if and only if $\Sigma_A\mathrm{fib}_{g'} \to \Sigma_A f^*\mathrm{fib}_g$ is an equivalence, if and only if, by Lemma 2.6.5.2, the function $\mathrm{fib}_{g'} \to f^*\mathrm{fib}_g$ is an equivalence over $\Gamma.A$. In particular, the second square in the claim is homotopy cartesian by Proposition 2.6.9.3, as we know that the fibre type of an isofibration is equivalent to the dependent type they come from. □

2.8.4 Trivial homotopy pullback

We want to prove in our setting the remarkable result from category theory for which a pullback of an isomorphism is an isomorphism, and conversely a square with two parallel isomorphisms is a pullback. First of all, we can unwind the meaning of previous results to get the most trivial homotopy pullback.

Remark 2.8.4.1. Note that Remark 2.7.0.8 means that we have a homotopy cartesian square

$$\begin{array}{ccc} A & = & A \\ \| & & \| \\ A & = & A \end{array}.$$

This is because the function of types $\Gamma, a : A \vdash \mathrm{pair}((a,a), \mathrm{refl}_a) : \Sigma_{(a,a'):A\times A} \mathrm{Id}_A(a,a')$ is an equivalence of types $A \simeq \Sigma_{(a,a'):A\times A} \mathrm{Id}_A(a,a')$ over Γ sending $a \mapsto (a,a,\mathrm{refl}_a)$.

More in general, whenever we have two parallel equivalences we get a homotopy cartesian square. This is particularly important because also the converse holds: a homotopy pullback of an equivalence is an equivalence.

Lemma 2.8.4.2. Let $\Gamma \vdash A, B, C, D$ type, and consider a commutative square

$$\begin{array}{ccc} D & \xrightarrow{\alpha} & A \\ {\scriptstyle\beta}\downarrow & & \downarrow{\scriptstyle f} \wr \\ B & \xrightarrow[g]{} & C \end{array}$$

with f an equivalence of types. Then, it is homotopy cartesian if and only if β is an equivalence of types.

Proof. By Theorem 2.8.3.1, the square is homotopy cartesian if and only if we have an equivalence $\mathrm{fib}_\beta \simeq g^*\mathrm{fib}_f \simeq$ over $\Gamma.B$. But now, by Proposition 2.6.10.1, we know that fib_f is a contractible type over $\Gamma.C$, so is its base change along g in context $\Gamma.B$ by Lemma 2.4.11.3. Therefore, the square is homotopy cartesian if and only if fib_β is contractible, as the map into a contractible type is essentially unique. By Proposition 2.6.10.1, this happens if and only if β is an equivalence. □

Homotopy pullbacks of this form will be the simplest ones we will encounter. The above result does not only give us examples of homotopy pullbacks. Instead, it also allows us to prove that a certain function is an equivalence.

2.8.5 Pasting of homotopy pullbacks

As the reader experienced in category theory might expect, the most useful tool to manipulate homotopy pullbacks is the pasting lemma. This easily allows us to prove that a square is homotopy cartesian, or to endow a type with the universal property of two different homotopy pullbacks.

Proposition 2.8.5.1 (Pasting of homotopy pullbacks)**.** Let $\Gamma \vdash A, B, C, D, B', D'$ type, and consider two commutative squares

$$\begin{array}{ccccc} Q & \xrightarrow{\overline{h}} & P & \xrightarrow{\overline{g}} & A \\ {\scriptstyle\overline{\overline{f}}}\downarrow & & {\scriptstyle\overline{f}}\downarrow & & \downarrow{\scriptstyle f} \\ D & \xrightarrow[h]{} & B & \xrightarrow[g]{} & C \end{array},$$

with the right square being homotopy cartesian. Then, the left square is homotopy cartesian if and only if the outer square is homotopy cartesian.

Proof. Because base change strictly enjoy the property of being compatible with composition, we rely on Theorem 2.8.3.1. Indeed, we have $P \simeq g^*\mathrm{fib}_f$ by assumption. By Lemma 2.4.11.5, the claim is equivalent to showing that $\mathrm{fib}_{\overline{\overline{f}}} \simeq h^*g^*\mathrm{fib}_f$ if and only if $\mathrm{fib}_{\overline{\overline{f}}} \simeq (g\circ h)^*\mathrm{fib}_f$. This follows by $h^*g^*\mathrm{fib}_f \equiv (g\circ h)^*\mathrm{fib}_f$ by the basic syntactic rules about base change. □

2.8.6 Homotopy pullback and product

Unsurprisingly, homotopy pullback is a generalisation of the product to an arbitrary base.

Lemma 2.8.6.1. Let $\Gamma \vdash A, B, C$ type.

(1) There is an equivalence of types $A \times_{\Delta^0_\Gamma} B \simeq A \times B$ over Γ.

(2) For every $\Gamma.A \vdash f : B[p]$, the following square is homotopy cartesian

$$\begin{array}{ccc} A \times C & \xrightarrow{\mathrm{pr}_1} & A \\ {\scriptstyle 1_B \times f}\downarrow & & \downarrow{\scriptstyle f} \\ B \times C & \xrightarrow[\mathrm{pr}_1]{} & B \end{array}.$$

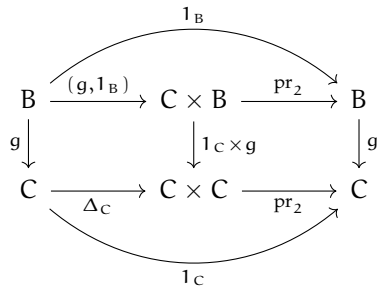
Proof. By Theorem 2.8.3.1, we need to show that the base change of the fibre type of $B \to \Delta^0_\Gamma$ along $A \to \Delta^0_\Gamma$ is equivalent to the fibre type of $A \times B \to A$. The first is, by Example 2.6.9.6, $\Gamma.\Delta^0_\Gamma \vdash B[p]$ type, and its base change along $!_A$ is $\Gamma.A \vdash B[p]$. Being it the fibre type of $A \times B \to A$, this concludes the claim. □

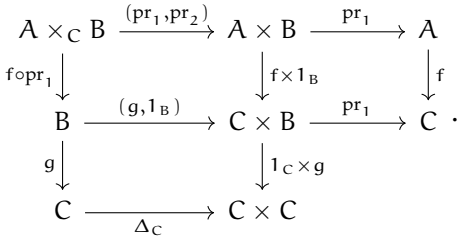
We can relate the homotopy pullback construction with that of product in another way. More explicitly, we may regard a homotopy pullback $A \times_C B$ as the homotopy pullback of $A \times B$ along the diagonal of C.

Lemma 2.8.6.2. Let $\Gamma \vdash A, B, C$ type, and consider $\Gamma.A \vdash f : C[p]$ and $\Gamma.B \vdash g : C[p]$. Then, we have a homotopy pullback square

$$\begin{array}{ccc} A \times_C B & \xrightarrow{(\mathrm{pr}_1, \mathrm{pr}_2)} & A \times B \\ {\scriptstyle f \circ \mathrm{pr}_1}\downarrow & & \downarrow{\scriptstyle f \times g} \\ C & \xrightarrow[\Delta_C]{} & C \times C \end{array}.$$

Proof. We have the commutative diagrams





By Pasting of homotopy pullbacks applied to the left diagram, since the outer square and the right square are both homotopy cartesian by Lemma 2.8.4.2 and Lemma 2.8.6.1, we deduce that the left square is a pullback. In particular, the lower square in the diagram on the right is homotopy cartesian. Now, we apply Pasting of homotopy pullbacks to the right diagram: the pasting of the two upper squares is a homotopy pullback as well as the right square by Lemma 2.8.6.1. We deduce that the upper left square is homotopy cartesian. Now, because we proved that the lower square is homotopy cartesian by means of the diagram on the left, a further iteration of Pasting of homotopy pullbacks on the right diagram gives the claim. □

2.8.7 Embeddings via homotopy pullbacks

Analogously to the characterisation of monos in ordinary categories via pullback squares, we can characterise embeddings by means of homotopy pullback squares.

Proposition 2.8.7.1. Let $\Gamma \vdash A, B$ type and let $\Gamma.A \vdash f : B[p]$ be a function of types. The following are equivalent.

(1) f is an embedding.

(2) f induces an equivalence of types $A \simeq \Sigma_{(a,a'):A\times A} \mathrm{Id}_B(f(a), f(a'))$ over Γ.

(3) The following square is homotopy cartesian

$$\begin{array}{ccc} A & = & A \\ \| & & \downarrow{\scriptstyle f} \\ A & \xrightarrow[f]{} & B \end{array}.$$

Proof. (2) is clearly equivalent to (3) by definition of homotopy cartesian square. Let us prove these are equivalent to (1). By Remark 2.7.0.8, we have an equivalence of types $\Sigma_{(a,a'):A\times A} \mathrm{Id}_A(a, a') \simeq A$ over Γ realised by the function $\Gamma, a : A \vdash \mathrm{pair}((a, a), \mathrm{refl}_a) : \Sigma_{(a,a'):A\times A} \mathrm{Id}_A(a, a')$. Postcomposed with the function $\Gamma, \sigma : \Sigma_{(a,a'):A\times A} \mathrm{Id}_A(a, a') \vdash (\Sigma_{(a,a'):A\times A} \mathrm{ap}_f)(\sigma) : \Sigma_{(a,a'):A\times A} \mathrm{Id}_B(f(a), f(a'))$, it is homotopic to $\Gamma, a : A \vdash (\Sigma_{(a,a'):A\times A} \mathrm{ap}_f)(\sigma) : \Sigma_{(a,a'):A\times A} \mathrm{Id}_B(f(a), f(a'))$. By the 2-out-of-3 property of equivalences seen in Lemma 2.4.11.5, we deduce that (2) holds if and only if $\Gamma, a : A \vdash (\Sigma_{(a,a'):A\times A} \mathrm{ap}_f)(\sigma) : \Sigma_{(a,a'):A\times A} \mathrm{Id}_B(f(a), f(a'))$ is an equivalence. Since Lemma 2.6.5.2 dependent sum reflects equivalences, this is equivalent to (3). □

2.9 Animae

We now want to explain the logical role of animae as contexts in our theory. This will lead us to formulate a fundamental principle governing the logical role of animae in the theory. The motivation to talk about animae at this point comes from very concrete reasons: something that is still missing from homotopy type theory is the presence of types whose terms are functions, which we call "collection of maps" types. Their necessity comes from many reasons that we will see, many of which are common with homotopy type theory, such as the ability to consider type families or statements depending on functions of types.

The way in which homotopy type theory introduces types of functions is by means of dependent products, which give rise to internal Homs. These always exist, and can be manipulated with ease since the rules of homotopy type theory are the same in all contexts. In particular, Meta Yoneda Lemma allows to show equivalences between function types very naively, almost set-theoretically.

However, key differences arise when we try to incorporate types of functions in the theory of categories. Dependent product is in fact introduced as a right adjoint to the pullback along a fibration. Thus, in our setting, dependent product cannot always exist, as the pullback functor does not preserve colimits in category theory. Getting rid of this principle is the first key step that distances the theory considerably from homotopy type theory. Semantically, this corresponds to the fact that the slice category of CAT over an arbitrary category does not have internal Homs. However, we do want to have *some* dependent products, e.g. for types in absolute context. This because we want to define the functor category at least in empty context.

The above is an instance of a crucial difference with homotopy type theory, which is the fact that the whole logic of the theory of categories is not invariant under slicing. Therefore, we need to consider a class of *special* contexts over which *every* rule, and in particular every operation which can be performed in absolute context, shall be allowed. These are exactly animae.

2.9.1 The logical principle of animae

The reason why we care about animae is, in the first place, their role in the logic of the theory. Despite the fact that our theory, up to now, has the same rules in every context and cannot tell whether we are in a context or in another, it is necessary to introduce an asymmetry. Indeed, we want our theory of categories to be agnostic about the context as long as it is an anima.

Fundamental principle of animae. Every rule of the theory will hold over any arbitrary anima context.

Some rules and operations will only be allowed over animae, the first being the existence of a type of functions. To obey to this principle, we will always make sure that any axiom we will introduce will hold over any anima. This fundamental logical property of animae justifies why we introduced the predicate of being an anima on contexts rather than on types, as it is a contextual property. Since, classically, this principle reflects into the fact that $\mathrm{CAT}_{/\Gamma}$ for Γ groupoid has the same internal logic as CAT, it will be our responsibility to characterise anima contexts as groupoids, morally those categories with all invertible arrows, later on. This principle is a big difference with homotopy type theory, where anima contexts are in fact all contexts, coherently with the interpretation of homotopy type theory as the theory of ∞-groupoids. This heavily links back to Types as theories, since it is the principle imposing animae, unlike arbitrary types, to behave like collections, or statements. In particular, we now see that we can formally characterise collections as types in Types as statements syntactically, through a kind of Yoneda Lemma.

2.9.2 Meta Yoneda Lemma for animae

We saw in Meta Yoneda Lemma that, in order to provide an equivalence between two types $\Gamma \vdash A, B$ type, it is sufficient to show a bijection between the terms of their weakenings up to homotopy which is compatible with change of context. The reason is the fact that we can weaken the types at themselves and consider the universal terms, and apply the bijection rules on these. However, this argument strongly relied on the fact that the same type constructor is well-defined over (arbitrary) context extension, or at least context extensions along themselves. Now, with types of functions, we will introduce type constructors that are defined only over animae. Therefore, in order to prove equivalences between them, such an argument will not be enough anymore. Indeed, if $\Gamma \vdash X$ type is an instance of a type constructor (depending on some types and terms) defined only for Γ anima, then its weakening at some $\Gamma \vdash A$ type might not be another instance of the type constructor (on the weakening of the types and terms), since $\Gamma.A$ is not necessarily an anima. In particular, we do not have any rules describing its terms. As just pointed out, the argument only made use of the weakening over the type itself, but a priori $\Gamma.X$ is not an anima in general. However, if the rules state that $\Gamma.X$ is in fact an anima (meaning that X will be a "groupoid" over Γ), then the Yoneda argument will work also for these types. In other words, for types which are defined only in anima contexts such that the context extension is still an anima, we keep the result:

Theorem 2.9.2.1 (Meta Yoneda Lemma for animae)**.** Let $\vdash \Gamma$ anima, and $\Gamma \vdash X, Y$ type. Assume that $\vdash \Gamma.X$ anima and $\vdash \Gamma.Y$ anima, and that we have rules

$$\frac{\Gamma \vdash x : X}{\Gamma \vdash f(x) : Y} \qquad \frac{\Gamma \vdash y : Y}{\Gamma \vdash g(y) : X}$$

$$\frac{\Gamma \vdash x : X}{\Gamma \vdash l(x) : \mathrm{Id}_X(g(f(x)), x)} \qquad \frac{\Gamma \vdash y : Y}{\Gamma \vdash r(y) : \mathrm{Id}_Y(f(g(y)), y)}$$

Then, we automatically have rules

$$\frac{\Gamma \vdash A \text{ type} \quad \vdash \Gamma.A \text{ anima} \quad \Gamma.A \vdash x : X[p]}{\Gamma.A \vdash f_A(x) : Y[p]} \qquad \frac{\Gamma \vdash A \text{ type} \quad \vdash \Gamma.A \text{ anima} \quad \Gamma.A \vdash y : Y[p]}{\Gamma.A \vdash g_A(y) : X[p]}$$

$$\frac{\Gamma \vdash A \text{ type} \quad \vdash \Gamma.A \text{ anima} \quad \Gamma.A \vdash x : X[p]}{\Gamma.A \vdash l_A(x) : \mathrm{Id}_{X[p]}(g_A(f_A(x)), x)} \qquad \frac{\Gamma \vdash A \text{ type} \quad \vdash \Gamma.A \text{ anima} \quad \Gamma.A \vdash y : Y[p]}{\Gamma.A \vdash r_A(y) : \mathrm{Id}_{Y[p]}(f_A(g_A(y)), y)}$$

with f_A, g_A, l_A and r_A compatible with base change under functions of types over Γ. Furthermore, $f_X(q)$ and $g_Y(q)$ determine an equivalence of types between X and Y.

Proof. The first part of the claim follows by the fundamental principle of animae, as the rules must canonically hold over any context extension to an anima. Once we have the rules below, the same proof of Theorem 2.4.12.1 applies as we can use the rules over $\Gamma.X$ and $\Gamma.Y$. □

Definition 2.9.2.2. Let $\vdash \Gamma$ anima. We define the judgement $\Gamma \vdash A$ anima to be the pair of judgements $\Gamma \vdash A$ type and $\vdash \Gamma.A$ anima.

Note that being an anima is still a contextual property, but for the mathematical intuition it is better to refer to animae over an anima as those types that extends it to an anima. Note that it is not a new primitive judgement, but just a name we give to a pair of judgements that happens to be particularly useful and intuitive, in the perspective that animae will be groupoids.

As one can expect, the condition in Meta Yoneda Lemma for animae of X being an anima over Γ will not be always fulfilled in our theory, and this will be clear from our first example, the dependent products. Because the context $\mathrm{Fun}(C, D)$ will not be an anima, the Yoneda argument will not work to provide equivalences of functor categories.

Remark 2.9.2.3 (Animae as animated collections)**.** Meta Yoneda Lemma for animae allows us to interpret animae over animae contexts even further. Indeed, given $\vdash \Gamma$ anima, those type constructors $\Gamma \vdash X$ anima can be interpreted as "collections" of terms: by the fundamental principle of animae, the syntactic proofs over Γ work by default also over the anima $\Gamma.X$, therefore X is only determined by its terms $\Gamma \vdash a : X$ up to homotopy. In homotopy type theory, the condition of being an anima is tautologically satisfied, thus all types can be interpreted as collections of terms and manipulated with ease. The removal of this assumption makes various subtleties arise in our theory, as we started observing in Remark 2.3.8.3.

Remark 2.9.2.4. We will see that the groupoid core will allow us to take a type constructor over animae, and turn it into an anima constructor over animae, by keeping the same terms over animae. The ability of performing category theory over it, then, allows to make naive manipulations with it: for instance, in order to prove an equivalence, it will suffice to show a bijection of terms up to homotopy. In homotopy type theory, we do not see or even need this operator, the reason being the fact that every context is an anima and every rule is the same over any context, therefore the groupoid core will be the identical assignment. In category theory, we can clearly see that it will be a non-trivial operator.

2.10 Mapping space

Following this premise about animae, we now can go back to the problem of constructing "collections of maps" types. These are extremely important for various reasons:

(1) We want to be able to make functions appear as variables in a context. For example, we want to have types or statements parametrised over a family of functions.

(2) We want to introduce a $\forall$ quantifier in our theory: the ability of putting a (dependent) function, from A to $x : A \vdash B(x)$ type, in the context means to work in the assumptions where *for every* $\vdash x : A$ we have a $\vdash f(x) : B(x)$. This dualises the $\exists$ quantifier encoded by the dependent sum with the same idea.

(3) We want to finally unstrictify the theory. Indeed, although we unstrictified the classical rules in many occasions, functions of types still obey to a strict calculus since they mirror that of context substitutions, fitting into an ordinary category. This has to be fixed, as functors between synthetic categories should not have a strict composition. The introduction of a "collection of maps type" whose terms are functions *up to homotopy*, will allow us to define the notion of *functor* of types. Being this correspondence only up to homotopy, functors will be endowed with a non-strictly associative composition.

(4) Because our theory is completely agnostic, it cannot speak about Hom-sets. However, in an ordinary category, Hom-sets contain a lot of information and provide proofs in the categorical world by means of set-theoretical manipulations. If we believe in the Yoneda lemma, the ability to have an internal replacement for these is crucial, namely a type of functions. This, however, should also possess the same advantages as the categorical Hom-set, first among all, the ability to prove categorical statements by means of naive proofs that resemble set theory.

(5) We want to have access to adjunctions, so that type constructors could be introduced directly as adjoints, and to an internal Yoneda Lemma.

Note that these are very concrete reasons which are funded on the necessity of replacing the Hom-sets manipulations in the syntax. Dependent products come, a priori, with further more sophisticated motivations, such as the necessity of having an actual internal Hom as a right adjoint to the product. Although we will need them as well, the first goal we have is to solve something more fundamental before. Summarising, we need to fulfil the following:

Goal. Given $\Gamma \vdash A, B$ type, we want to have a "collection of maps" type over Γ (or, at least, a context that does not contain A) whose terms are functions from A to B up to homotopy. Furthermore, a good "collection of maps" type should have the property of admitting naive manipulations as in Meta Yoneda Lemma for animae, as if they were sets, in order to access adjunctions and Yoneda Lemma internally.

As we pointed out in the previous section, in homotopy type theory this is realised by means of dependent products, and in particular function types. Up to unstrictifying the rules, we could in fact make the correspondence between functions and terms of the dependent product hold only up to homotopy. In category theory, however, we gave reasons for which we cannot have dependent products in general. It is still true, though, that we have them in empty context, as we want functor categories between categories to exist. By the fundamental principle of animae, since the empty context is an anima, then all types over animae will admit a dependent product. This will endow the theory of types over an anima with an internal Hom. It is quite unsatisfying, though, not to have internal Homs for *all* types, as we have in homotopy type theory, due to their importance seen above. Luckily, given $\vdash \Gamma$ anima, $\Gamma \vdash A$ type and $\Gamma.A \vdash B, C$ type, we will still be able to construct a local functor category $\mathrm{Fun}_A(B, C)$ out of the dependent products that exist. The price we pay is just that this will not be a type over $\Gamma.A$, but over Γ instead. This would seem to solve the problem, as we would get types of (local) functions for all types. However, this apparent correction would not be enough to make dependent products (and local functor categories) good "collection of maps" types in our theory, although they are in homotopy type theory. This is because the existence over not-every context makes them poorly behaved: it is not just an obstruction to their formation, but something deeper. Indeed, these (local) functor categories would not realise our goal, which aimed to make them as easy to manipulate as possible, in order to mimic the set-theoretic bijection proofs that we can classically perform over Hom-sets: we would like to obtain "collections" in the sense of Remark 2.9.2.3. In homotopy type theory, this still works with dependent products because the rules are the same in every context, thus every function type $\{X \to Y\}$ over Γ is, by default, such that the theory over $\Gamma.\{X \to Y\}$ has all the rules we wish, and we can perform any construction over it. For instance, there is a function type $\{X[p] \to Y[p]\}$ in context $\Gamma.\{X \to Y\}$ which comes equipped with a universal term, and this allows us to prove equivalences with ease thanks to Meta Yoneda Lemma. However, in category theory, functor categories $\mathrm{Fun}(A, B)$, introduced as internal Homs over an anima Γ, are *not* such that $\Gamma.\mathrm{Fun}(A, B)$ is still an anima. Therefore, due to the logical principle of animae, we cannot perform all of category theory over it, and thus we cannot provide naive manipulations to show equivalences between them. This means that, as opposed to homotopy type theory, in our setting we will need to distinguish internal Homs (dependent products) and "collections of maps", and we will refer to the latter ones as *mapping spaces*. Denoted with $\mathrm{Map}(A, B)$, these will be introduced as types only over animae Γ. Their terms will be in correspondence with functions of types, up to homotopy, just as internal Homs $\mathrm{Fun}(A, B)$, but with the difference that $\mathrm{Map}(A, B)$ will still be an anima over Γ. In particular, they will be "collections" as in Remark 2.9.2.3, of which we can show equivalence in a naive, almost set-theoretical, way by means of Meta Yoneda Lemma for animae.

Remark 2.10.0.1. As mentioned, this will become a familiar process encoded by the groupoid core, turning types into other types ("collections" in the sense of Remark 2.9.2.3) with the same terms (in anima contexts) over which we can perform category theory. This construction would allow us to construct mapping spaces out of functor categories, also philosophically. However, we want to have access to mapping spaces before anything else, as they will allow us to define the groupoid core operator, as a particular case, and to talk about adjunctions and much more. Then, we will make use of these in order to introduce the dependent product and *prove* that the groupoid core construction on functor categories does give rise to mapping spaces.

2.10.1 Rules for the dependent mapping space

We can now introduce mapping space type, that will give us all the advantages we mentioned. More generally than the reader might expect, we actually construct a dependent mapping space, whose terms are dependent functions of types. First of all, we have a *formation rule*

$$\frac{\vdash \Gamma \text{ anima} \quad \vdash \Gamma.\Delta \text{ ctx} \quad \Gamma.\Delta \vdash B \text{ type}}{\Gamma \vdash \mathrm{Sec}_\Delta(B) \text{ type}}.$$

Note that, because this type is well-formed only over animae, the rule of compatibility with base change will be stated only for base change along anima. Explicitly, it will be of the form

$$\frac{\vdash \Gamma, \Xi\,\mathrm{anima} \quad \vdash \gamma : \Xi \to \Gamma\,\mathrm{ctx} \quad \vdash \Gamma.\Delta\,\mathrm{ctx} \quad \Gamma.\Delta \vdash B\,\mathrm{type}}{\Xi \vdash (\mathrm{Sec}_{\Delta}\, B)[\gamma] \equiv \mathrm{Sec}_{\Delta[\gamma]}(B[\gamma.\Delta])\,\mathrm{type}}$$

where, as intuitive, if $\Delta \equiv \Delta_1 \dots \Delta_n$, the base change $\Delta[\gamma]$ stands for $\Gamma.\Delta_1[\gamma] \dots \Delta_n[\gamma.\Delta_1 \dots \Delta_{n-1}]$, and $\gamma.\Delta$ is the obvious extension of γ. The type $\mathrm{Sec}_{\Delta}\, B$ is called the *dependent mapping space from* Δ *to* B, or *collection of sections of* B *over* Δ, and is also denoted as $\Gamma \vdash \mathrm{Sec}_{x:\Delta_1,\dots x_n:\Delta_n}\, B(x)$ to emphasise the interpretation as collection of sections. As intuitive, the ordinary mapping space will coincide with the dependent mapping space when the type dependence is trivial (a weakening), in the same way functions are dependent functions with trivial dependence. We now state the *introduction rule*

$$\frac{\vdash \Gamma\,\mathrm{anima} \quad \vdash \Gamma.\Delta\,\mathrm{ctx} \quad \Gamma.\Delta \vdash B\,\mathrm{type} \quad \Gamma.\Delta \vdash f : B}{\Gamma \vdash \lambda(f) : \mathrm{Sec}_{\Delta}\, B},$$

associating to each dependent function a term of the dependent mapping space, and the *elimination rule*

$$\frac{\vdash \Gamma\,\mathrm{anima} \quad \vdash \Gamma.\Delta\,\mathrm{ctx} \quad \Gamma.\Delta \vdash B\,\mathrm{type} \quad \Gamma \vdash f : \mathrm{Sec}_{\Delta}\, B}{\Gamma.\Delta \vdash \mathrm{ev}_f : B},$$

which provides the converse assignment to each term of $\mathrm{Sec}_{\Delta}\, B$ of a dependent function from Δ to B.

Remark 2.10.1.1. With more explicit notation, given $\Gamma, a : A \vdash f(a) : B(a)$, we will denote $\lambda(f)$ as the dependent function $\Gamma \vdash \lambda a.f(a) : \mathrm{Sec}_{x:A}\, B(x)$. This suggests that a is a bounded variable: for the more mathematically-oriented reader, the notation $\lambda a.f(a)$ stands for $a \mapsto f(a)$.

Note that, in the case of length one $\Gamma \vdash \Delta \equiv A\,\mathrm{type}$, these rules say that there are two assignments in opposite directions between terms of $\mathrm{Sec}_A\, B$ and dependent functions from A to B, as we wish. In order to name such a constructor a mapping space, we want to make the correspondence bijective up to homotopy. For this, we have the *computation rule*

$$\frac{\vdash \Gamma\,\mathrm{anima} \quad \vdash \Gamma.\Delta\,\mathrm{ctx} \quad \Gamma.\Delta \vdash B\,\mathrm{type} \quad \Gamma.\Delta \vdash f : B}{\Gamma.\Delta \vdash \mathrm{comp}^{\mathrm{Sec}}_{\Delta;B}(f) : \mathrm{Id}_B(\mathrm{ev}_{\lambda(f)}, f)}.$$

In other words, given a dependent function f from A to B, the evaluation function of its associated term of $\mathrm{Sec}_A\, B$ is homotopic to f.

Remark 2.10.1.2 (Difference with classical rules)**.** An important difference with the ordinary rules for the dependent products, leaving aside the "anima" conditions on contexts, is that we did not impose any rule realising the other composite as the identity as well: in order to have a homotopy bijection of terms between B and $\mathrm{Sec}_A\, B$, given $\Gamma \vdash f : \mathrm{Sec}_A\, B$ we would expect the term $\lambda(\mathrm{ev}_f)$ to be homotopic to f. The rules of ordinary type theory do impose that $\lambda(\mathrm{ev}_f) \equiv f$. This difference is *not* related to a distinction between the dependent mapping space and the dependent product. Instead, it is a choice we make due to the unstrictification process. We will soon see how to recover the wished homotopy bijection in our setting, as a consequence of a more general phenomenon.

The next rule is peculiar to this theory, the *animation rule*

$$\frac{\vdash \Gamma\,\mathrm{anima} \quad \vdash \Gamma.\Delta\,\mathrm{ctx} \quad \Gamma.\Delta \vdash B\,\mathrm{type}}{\vdash \Gamma.\,\mathrm{Sec}_{\Delta}\, B\,\mathrm{anima}}.$$

This was widely motivated, and makes $\mathrm{Sec}_{\Delta}\, B$ behave like a "collection" of maps as in Remark 2.9.2.3, in the sense that we can provide equivalences of mapping spaces with naive proofs, in the same way we do with the Hom-sets in the homotopy category in the classical theory. Now, this means that the context $\Gamma.\,\mathrm{Sec}_A\, B$ is an anima, and thus admits dependent mapping spaces and any other construction over it.

Remark 2.10.1.3 (Difference with dependent product)**.** For the reader experienced in type theory, note that all the previous rules mimic the classical rules for dependent products over animae, in a non-strict sense.

However, what really forces the dependent mapping space to differ from a dependent product is this animation rule: the one that allows to manipulate $\mathrm{Sec}_\Delta\, B$ with ease, as opposed to the case of $\Pi_\Delta B$. Indeed, for the dependent product, such a rule would not be true, and to determine the type, since it is not a collection in the sense of Remark 2.9.2.3, we would need to manually state the terms of all the weakenings.

The animation rule, by Meta Yoneda Lemma for animae, allows us to determine the type constructor by means of the terms of its weakenings at animae only, which is what we started doing with the rules above. However, the rules we need in order to completely determine the dependent mapping space are not complete yet. Indeed, we are missing some important pieces of information, starting from Remark 2.10.1.2, and in general related to the notion of equality in this type:

Remark 2.10.1.4 (Pointwise and global equality)**.** The introduction of a new type $\Gamma \vdash \mathrm{Sec}_A\, B$ type whose terms correspond to terms of $\Gamma.A \vdash B$ type opens up for a new notion of equality of dependent functions. Indeed, there is an identity type $\Gamma.\mathrm{Sec}_A\, B.(\mathrm{Sec}_A\, B)[p] \vdash \mathrm{Id}_{\mathrm{Sec}_A\, B}$ type. In particular, given two terms $\Gamma \vdash f, f' : \mathrm{Sec}_A\, B$, for the rest of this section we will refer to terms of $\Gamma \vdash \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ type as *global equalities*, and to terms of $\Gamma.A \vdash \mathrm{Id}_B(\mathrm{ev}_f, \mathrm{ev}_{f'})$ as *pointwise equalities* between f and f'. Of course, the latter is just a homotopy between the underlying dependent functions, which has been the way we identified functions up to now. One of the key discussion in this section will revolve around the comparison of these two kinds of equalities between terms of the dependent mapping space. Unsurprisingly, this will also lead us to consider two, a priori different, notions of equivalences.

The next paragraph will be devoted to understanding the relation between these two notions of equality and figuring out what we are missing in this picture. Before, that, we want to highlight some aspects about the contextual role of the dependent mapping space.

Remark 2.10.1.5 (Contextual properties of Sec)**.** The dependent mapping space introduces into the theory the ability to form new kinds of contexts. This is important, since Δ^0 and Σ did not introduce any new context in the theory, up to equivalence: $\Gamma, x : \Delta^0$ and $\Gamma, s : \Sigma_{x:A} B(x)$ are equivalent to the basic contexts Γ and $\Gamma, x : A, y : B(x)$ respectively. Instead, the important feature they introduced into the theory was under the vests of types. Before that, we had no type with a unique term, or whose terms were pairs. Reformulated, these were representing pre-existing contexts into types. For the dependent mapping spaces, the role is quite reversed: although we do not have a homotopy bijection of terms yet, but only a correspondence, we already have a type whose terms are those of $\Gamma, x : A \vdash B(x)$ type, i.e. B itself. A first interest in the dependent mapping space is the fact that the type $\Gamma \vdash \mathrm{Sec}_{x:A}\, B(x)$ type has the same terms but over a different context. Even more importantly, though, it introduces the context $\Gamma, f : \mathrm{Sec}_{x:A}\, B(x)$ into the theory, and this is not equivalent to any other context we already had. This is because the context $\Gamma, x : A, y : B(x)$ is equivalent to $\Gamma, s : \Sigma_{x:A} B(x)$, that is the idea of working in the assumption of having an $x : A$ and a $y : B(x)$. Instead, the context $\Gamma, f : \mathrm{Sec}_{x:A}\, B(x)$ means that we are in the assumption of having *for every* $x : A$ an $f(x) : B(x)$. We will see that, compared to the ordinary dependent product, this feature of having a new context will be taken even further by the animation rule. Indeed, even when A will be a point, where quantifying over it should not change anything, the obtained context will differ from any other we have. This will give us access to the notion of contexts for objectwise statements.

2.10.2 The need of a function extensionality principle

We want to spend some words to some important observations, trying to understand what we are missing to complete the rules of the dependent mapping space as a type. This will lead us to motivate a *function extensionality* principle. Morally, this means that we want to be able to deduce homotopies in the dependent mapping spaces out of pointwise homotopies between the associated evaluation functions. Set-theoretically, it is the idea that two functions f and g are equal if their images coincide at each element. This is for sure a property we want, but is even more fundamental than one might expect, as it completes the picture we introduced up to now. First of all, since function extensionality would be a principle determining the identity type of a type constructor (the dependent mapping space), the reader might wonder why such a rule is necessary. Indeed, we never had the need to impose a rule of this kind for the the previous type constructors (Id, Σ, Δ^0). Instead, we were a priori able to construct homotopies inside them, or even

to determine their identity types (Remark 2.5.1.1, Proposition 2.6.6.1). The reason is that the dependent mapping space is a non-inductive and very different type in nature compared to the previous ones:

(1) Previously, the first tool we had to obtain homotopies in the just-introduced type constructors was by means of the computation rules. These allowed us to characterise all the terms up to homotopy. However, for the dependent mapping space, as we pointed out in Remark 2.10.1.2, we have only one computation rule, that realises the two assignments λ and ev as one-sided inverses, and not as a homotopy bijection of terms. Thus, we cannot deduce *yet* that *every* term of $\mathrm{Sec}_A\ B$ is a dependent function from A to B up to homotopy, although we wish it to be true.

(2) Previously, the main way we induced homotopies in the just-introduced type constructors, was by using Lemma 2.4.3.1 on the term constructors. Indeed, since these were automatically functorial, they preserved equalities. However, in the dependent mapping space, the λ term constructor is *not* functorial. Thus, we are not allowed to construct any homotopy in the dependent mapping space through this argument.

First of all, we want to show the reader why solving these is not just reasonable, but is even necessary: it is a key step in order to make the mapping space a uniquely determined type.

Remark 2.10.2.1 (Why solving these issues?)**.** As we mentioned previously, the rules we need in order to determine the dependent mapping space are not complete yet. This is because we are missing the two important pieces of information (1) and (2) above, as we will now motivate. Of course, for this purpose, (1) is essential: in order to say what the terms of $\mathrm{Sec}_A\ B$ are, for every $f : \mathrm{Sec}_A\ B$ *we need a homotopy equating* $\lambda(\mathrm{ev}_f)$ *and* f. However, this is not enough: even if we had a rule $\Gamma \vdash \sigma_f : \mathrm{Id}_{\mathrm{Sec}_A\ B}(\lambda(\mathrm{ev}_f), f)$, this would still not determine the type $\mathrm{Sec}_A\ B$ uniquely. Indeed, the reader should be careful with the interpretation of Meta Yoneda Lemma for animae: even by making explicit all the terms of the weakenings at animae, we still have no way of constructing homotopies inside the dependent mapping space, and this is fundamental to show that any other type satisfying the same rules is equivalent to it. More explicitly, if we had another type constructor $\mathrm{Sec}'_A\ B$ satisfying the same rules (up to replacing $\lambda, \mathrm{ev}, \mathrm{comp}^{\mathrm{Sec}}, \sigma$ with $\lambda', \mathrm{ev}', \mathrm{comp}^{\mathrm{Sec}'}, \sigma'$ respectively), we have the assignments $f : \mathrm{Sec}_A\ B \mapsto \lambda'(\mathrm{ev}_f) : \mathrm{Sec}'_A\ B$ and $g : \mathrm{Sec}'_A\ B \mapsto \lambda(\mathrm{ev}'_g) : \mathrm{Sec}_A\ B$, but these are not inverses a priori. In order to construct a homotopy from $\lambda(\mathrm{ev}'_{\lambda'(\mathrm{ev}_f)})$ to f, we would need to know how to induce equalities inside $\mathrm{Sec}_A\ B$, and the rule with σ would be of no help. For this, *we need* λ *to preserve equalities* as in (2): in such assumption, $\Gamma \vdash \mathrm{comp}^{\mathrm{Sec}'}_{A;B}(\mathrm{ev}_f) : \mathrm{Id}_{\mathrm{Sec}_A\ B}(\mathrm{ev}'_{\lambda'(\mathrm{ev}_f)}, \mathrm{ev}_f)$ provides a homotopy from $\lambda(\mathrm{ev}'_{\lambda'(\mathrm{ev}_f)})$ to $\lambda(\mathrm{ev}_f)$, and thus to f. The same holds for the other composite.

In conclusion, we need to impose rules about the behaviour of homotopies inside the dependent mapping space. We will now see, intuitively, how a *function extensionality* rule would allow us to overcome both (1) and (2), giving it a strong motivation. We wish to point out that the following discussion is independent of having the "animae" conditions on contexts.

Remark 2.10.2.2 (Overcoming (1) with function extensionality)**.** Although we want all terms of $\mathrm{Sec}_A\ B$ to be in homotopy bijection dependent functions from A to B up to homotopy, we have good reasons *not* to impose as a rule that, given $\Gamma \vdash f : \mathrm{Sec}_A\ B$, we have a computational homotopy between $\lambda(\mathrm{ev}_f)$ and f: First of all, we do not want to determine the type by forcing a bijection of terms, where we provide an assignment together with a converse assignment that is both a left and a right inverse. Secondly, such a rule would not be of much use in order to construct more general homotopies in the dependent mapping space. Indeed, if we start with f and f', this rule would only tell us that, if their evaluation functions are *judgementally* equal, then they are homotopic, which has a very limited usage. Although we do want the mentioned rule to hold, in order to get a homotopy bijection between terms of $\mathrm{Sec}_A\ B$ and dependent functions from A to B, this will be a *consequence* of function extensionality, as their evaluation functions will be homotopic via the computation rule. This is true in ordinary type theory as well. However, in the ordinary setting, the sole purpose of the rule equating $\lambda(\mathrm{ev}_f)$ and f, which we omitted, is to make this equality judgemental. Since we strongly want to avoid that in this setting, we will not state such a rule, and instead deduce a propositional equality from function extensionality.

Remark 2.10.2.3 (Overcoming (2) with function extensionality)**.** For the dependent mapping space, the λ operator is *not* a function of types, unlike all the other terms constructors we met along the way. This is crucial, because by Lemma 2.4.3.1 we know that all functions, by default, preserve equalities. If this was

the case for λ, we would consequently be able to induce equalities in the mapping space out of pointwise equalities, which is the core idea of function extensionality. The reason this does not happen a priori is that we cannot even form a functorial statement introducing λ: by Remark 2.10.1.5, $\Gamma \vdash x : A, f(x) : B(x)$ means that we are in the assumptions of having an $x : A$ and an $f(x) : B(x)$, and *not* in the assumptions of having a dependent function $\Gamma, x : A \vdash f(x) : B(x)$, which means having an $f(x) : B(x)$ *for every* $x : A$. This is, conceptually, very different, as in the first situation it does not make sense to form a $\lambda(f)$ if we only have an $x : A$ and its image $f(x) : B(x)$. Therefore, there is no way of making λ a functorial assignment, even by leaving aside the problems of working over animae. Indeed, this problem is present even in ordinary type theory. As a consequence, we do not inherit for free the ability to induce equality in mapping spaces from pointwise equality through Lemma 2.4.3.1. Because we want this function extensionality property to hold, even without a sort of functoriality of λ, this must be introduced independently as a rule in the theory.

Conceptually, as the reader might suspect, this problem does not subsist for the ev constructor. Indeed, we can make ev fully functorial thanks to the animation rule. This will play a key role, as it will make ev compatible with equalities by Lemma 2.4.3.1.

Remark 2.10.2.4 (Towards function extensionality)**.** The animation rule has a further scope: it allows us to make ev functorial. Indeed, being $\mathrm{Sec}_A\, B$ an anima over Γ, we can consider the universal term $\Gamma, f : \mathrm{Sec}_{x:A}\, B(x) \vdash f : \mathrm{Sec}_{x:A}\, B(x)$, and thus use the elimination rule to form $\Gamma, f : \mathrm{Sec}_{x:A}\, B(x), a : A \vdash \mathrm{ev}_f(a) : B(a)$. This promotes evaluation to a function ev_q from $(\mathrm{Sec}_A\, B)[p]$ to B over $\Gamma.A$, which we just name ev for simplicity. By Lemma 2.4.3.1, ev becomes fully compatible with equalities in $\mathrm{Sec}_A\, B$: we have a functorial homotopy $\Gamma, a : A, f, f' : \mathrm{Sec}_{x:A}\, B(x), \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f') \vdash \mathrm{ap}_{\mathrm{ev}}(\alpha) : \mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))$ type. This is a function of types from the weakening of $\mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ at A to $\mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))$ over $\Gamma, a : A, f, f' : \mathrm{Sec}_{x:A}\, B(x)$, that means that global homotopy implies pointwise homotopy between the evaluation functions. Function extensionality reverses the implication, by saying that these two kinds of equality coincide. However, this does *not* correspond to saying that such a function is an equivalence of types, as $\mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ is forced to live over A. More explicitly, having a function in the opposite direction would mean that, fixed $a : A$ and given $f, f' : \mathrm{Sec}_{x:A}\, B(x)$ with $\alpha(a) : \mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))$, we would get a global homotopy between f and f'. Instead, given $f, f' : \mathrm{Sec}_A\, B$, we want to say that if *for every* $a : A$ we have a homotopy $\alpha(a) : \mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))$, then we have a homotopy between f and f'. Therefore, we do not want to compare the two types above. Instead, we define the *type of pointwise homotopies* as

$$\Gamma, f : \mathrm{Sec}_{x:A}\, B(x), f' : \mathrm{Sec}_{x:A}\, B(x) \vdash f \simeq f' :\equiv \mathrm{Sec}_{x:A}\, \mathrm{Id}_{B(x)}(\mathrm{ev}_f(x), \mathrm{ev}_{f'}(x)),$$

whose terms are indeed in correspondence with pointwise homotopies of the evaluation functions, which function extensionality will promote to a homotopy bijection. We wish to compare the types $\mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ and $f \simeq f'$. Unfortunately, $\mathrm{ap}_{\mathrm{ev}}$ does not automatically promote to a function comparing the desired types. More precisely, it is true that for every $\Gamma, f, f' : \mathrm{Sec}_A\, B \vdash \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ we get a $\Gamma \vdash \lambda x.\mathrm{ap}_{\mathrm{ev}}(\alpha(x)) : f \simeq f'$, and the same holds over any anima, but we are not able to express this functorially in α in the sense of Remark 2.3.8.2, i.e. as a judgement, since it is realised over animae only. This means that we do *not* have any judgement of the form $\Gamma, f, f' : \mathrm{Sec}_A\, B, \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f') \vdash \lambda x.\mathrm{ap}_{\mathrm{ev}}(\alpha(x)) : f \simeq f'$, because we do not know whether the identity type is an anima. Hence we cannot perform the λ construction over the identity type and consider its universal term. Function extensionality wishes to say that an assignment of this form, if it existed, would be an equivalence. This is what happens in the ordinary case, as there are no issues with certain contexts being animae: all meta-theoretical assignments on terms get promoted to functorial assignments as seen in Remark 2.3.8.3.

Remark 2.10.2.4 is a first step towards the formulation of function extensionality, as we proved that homotopy in the dependent mapping space induces a homotopy between the evaluation dependent functions, and we would like to reverse such implication. However, because this meta-theoretical assignment holds only over animae, it is too early to assemble this to some kind of functorial assignment between identity types in the sense of Remark 2.3.8.1, to be imposed to be an equivalence. Therefore, we first need to find a way to express this partially-functorial assignment internally, as a judgement, hence as a proper function of types. In this way, function extensionality will be expressed as it being an equivalence. This means that, before stating function extensionality, we need to pass through the following intermediate steps:

(a) We introduce two particular cases of dependent mapping spaces: groupoid cores and ordinary (and

local) mapping spaces. Groupoid cores will be a tool to express internally, as judgements, those partially functorial (meta-theoretical) statements over animae, mimicking what happens with fully functorial statements as in Remark 2.3.8.2.

(b) Given a function of types f, we construct a type whose terms exhibit f as an equivalence. This allows us to introduce rules that impose equivalences of types.

(c) After the above points, we can formulate function extensionality: (a) allows us to assemble Remark 2.10.2.4 to some kind of functorial assignment, (b) enables us to impose it to be an equivalence.

2.10.3 Groupoid core: a first taste

The first particular case of dependent mapping space we want to study is the groupoid core. This will have a fundamental role throughout the whole theory, and is a canonical way to turn any type over an anima Γ into an anima. The whole next section will be devoted to discussing its properties, for upcoming needs. However, we introduce this concept right now because, in the first place, the groupoid core allows us to internalise those meta-theoretical assignments that hold only over animae. The idea is to find an analogous correspondence to Remark 2.3.8.2 in the case of partially-functorial assignments over animae, making us introduce the notion of *objectwise statements*. In particular, we will be able to make Remark 2.10.2.4 functorial, in some sense. This is a first key step towards the formulation of function extensionality.

Construction 2.10.3.1 (Groupoid core)**.** Let $\vdash \Gamma$ anima and $\Gamma \vdash A$ type. The *groupoid core of* A is defined as $\Gamma \vdash A^{\simeq} :\equiv \mathrm{Sec}_{\emptyset}(A)$ type. In particular, for every $\Gamma \vdash S$ type such that $\vdash \Gamma.S$ anima, the terms of $\Gamma.S \vdash A^{\simeq}[p] \equiv (A[p])^{\simeq}$ type are in correspondence with the terms of $\Gamma.S \vdash A$ type. We will promote this to a homotopy bijection later, although for the moment we only know that we have two assignments in opposite directions. Explicitly, we have

$$\frac{\vdash \Gamma \text{ anima} \quad \Gamma \vdash a : A}{\Gamma \vdash \lambda(a) : A^{\simeq}} \qquad \frac{\vdash \Gamma \text{ anima} \quad \Gamma \vdash A \text{ type} \quad \Gamma \vdash a : A^{\simeq}}{\Gamma \vdash \mathrm{ev}_a : A}.$$

Furthermore, we inherit an *animation rule*

$$\frac{\vdash \Gamma \text{ anima} \quad \Gamma \vdash A \text{ type}}{\vdash \Gamma.A^{\simeq} \text{ anima}},$$

which is, $A^{\simeq}$ is always an anima in context Γ. In particular, by Remark 2.10.2.4, this animation rule allows to make the assignment ev functorial, in the sense that we have $\Gamma.A^{\simeq} \vdash \mathrm{ev} :\equiv \mathrm{ev}_q : A[p]$. This determines a function of types ev from $A^{\simeq}$ to A over Γ, mapping any generalised term a of $A^{\simeq}$ into the generalised ev_a of A. The lack of functoriality of λ observed previously, means that we do not have a function of types in the opposite direction, as we wish from a categorical perspective. The obstacle here is the animation rule.

The lack of an assignment that goes in the opposite direction is linked to the fact that $A^{\simeq}$ and A are very different type, in particular, $A^{\simeq}$ contains less information but enjoys much better contextual properties. The reader might now wonder where the difference comes from, since the rules of the groupoid core resemble those of the identical operator. These explicitly force a homotopy bijection between the generalised terms over animae of $\Gamma \vdash A$ type and $\Gamma \vdash A^{\simeq}$ type, but contain no information about other generalised terms since the groupoid core cannot be formed over non-anima contexts, and this does not allow to form a functorial assignment from A to $A^{\simeq}$. In particular, because A is not an anima in general, Meta Yoneda Lemma for animae does not apply to deduce an equivalence between A and $A^{\simeq}$. As a consequence, the replacement of a type with its groupoid core loses information about terms over more generic contexts that are not animae, which are crucial data as long as A is not an anima. Then, how can we recover the whole groupoid core $A^{\simeq}$ knowing rules determining only some of its terms as those of A? The secret for this lies in the animation rule. Thanks to it, we can uniquely determines $A^{\simeq}$ out of the few terms it has in common with A (and with function extensionality), using the fact it can be treated as a "collection". In particular, despite the information we lose, considering the groupoid core makes us gain very nice properties, such as the fact that the theory over $A^{\simeq}$ possesses all the rules of category theory.

Linked to the above discussion, we are finally ready to introduce the very concept for which we anticipated the discussion about the groupoid core: objectwise statements.

Remark 2.10.3.2 (Objectwise statements)**.** Assume that we a rule of the form

$$\frac{\vdash \Gamma \,\mathrm{anima} \quad \Gamma \vdash A, B\,\mathrm{type} \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}.$$

By the logical principle of animae, such a rule holds automatically over any context extension of Γ along an anima. More explicitly, it is equivalent to a rule

$$\frac{\vdash \Gamma \,\mathrm{anima} \quad \Gamma \vdash S, A, B\,\mathrm{type} \quad \vdash \Gamma.S\,\mathrm{anima} \quad \Gamma.S \vdash a : A[p]}{\Gamma.S \vdash f_S(a) : B[p]}$$

compatible with base change. Then, because $\Gamma \vdash A^{\simeq}$ anima, we can apply the rule to $S \equiv A^{\simeq}$ and $\Gamma.A^{\simeq} \vdash a \equiv \mathrm{ev} : A[p]$, to get a judgement $\Gamma.A^{\simeq} \vdash f :\equiv f_{A^{\simeq}}(\mathrm{ev}) : B[p]$. Conversely, given $\Gamma.A^{\simeq} \vdash f : B[p]$, we can form a rule as above by putting, for $\Gamma \vdash S$ anima and $\Gamma.S \vdash a : A[p]$, $f_S(a) :\equiv f(\lambda(a))$.

We refer to this situation as an *objectwise statement over* A: the formulations via rule are called *minimal* and *exhaustive meta-theoretical formulations* respectively, whereas the one via judgement is called the *internal formulation*. Note that this allows us to write any objectwise statement over A as a functorial statement over $A^{\simeq}$, namely as a judgement $\Gamma, x : A^{\simeq} \vdash f(x) : B$. We can in fact show that if we start from a meta-theoretical formulation with maps $\Gamma.S \vdash f_S : B[p]$ for S anima, take the associated internal judgement, and then consider the rules associated to it, we get back the original assignments. Indeed, for any $\Gamma \vdash S$ anima and $\Gamma.S \vdash a : A[p]$, the functoriality of $f_{A^{\simeq}}$ induces, by Lemma 2.4.3.1, a homotopy $\Gamma \vdash \mathrm{ap}_{f_{A^{\simeq}}}(\mathrm{comp}^{\mathrm{Sec}}_{\emptyset;A}(a)) : \mathrm{Id}_{B[p]}(f_{A^{\simeq}}(\mathrm{ev}) \circ (\lambda(a)) \equiv f_S(\mathrm{ev}_{\lambda(a)}), f_S(a))$. On the other hand, it is still early to show that the converse composite gives back what we start from. In other words, we cannot deduce that the two formulations are indeed equivalent in the homotopical sense. This will be a further consequence of function extensionality.

Furthermore, we mentioned in Remark 2.3.8.3 that the minimal meta-theoretical formulation in homotopy type theory is automatically a functorial statement as in Remark 2.3.8.2. We observed that, in our setting, this was not equivalent to a functorial statement. Here, when over an anima, we finally can characterise it automatically as an objectwise statement, which is a weaker notion than a functorial statement that can be seen only in the present theory. In particular, we figured out the best kind of functoriality which our theory can equip these kinds of statements with, which is the functoriality in the groupoid core. An instance of this fact is the definition of embedding in homotopy type theory given in [BGL+17]. There, it is just given the minimal formulation of the definition, saying that $f : A \to B$ is an embedding if for all $\Gamma \vdash a, b : A$ we have $\mathrm{Id}_A(a, b) \simeq \mathrm{Id}_B(f(a), f(b))$. If we formulated this definition in this way in our setting, in the case Γ is an anima, it would hold objectwise only, whereas the correct definition of embedding is functorial in all generalised terms of A. Thus, we need to use the meta-theoretical or, even better, the internal formulation of a functorial statement as in Remark 2.3.8.2.

Let us give the first crucial example of objectwise statement which we will soon need.

Construction 2.10.3.3. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash B$ type. We restrict to the case $\Delta \equiv A$ with $\Gamma \vdash A$ type for the sake of simplicity. We want to internalise Remark 2.10.2.4 to make it a functorial statement. We saw that for every $\Gamma, f, f' : \mathrm{Sec}_A\, B \vdash \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ we get a $\Gamma \vdash \lambda x.\mathrm{ap}_{\mathrm{ev}}(\alpha)(x) : f \simeq f'$. Because this is the minimal meta-theoretical formulation of an objectwise statement as in Remark 2.10.3.2, we know we can make it exhaustive by extending it to every anima. In particular we can consider the extension to the groupoid core $\mathrm{Id}_{\mathrm{Sec}_A\, B}(f(s), f'(s))^{\simeq}$ with the evaluation $\Gamma, \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')^{\simeq} \vdash \mathrm{ev}_\alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ as in Remark 2.10.3.2, to get a judgement

$$\Gamma, f, f' : \mathrm{Sec}_A\, B, \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')^{\simeq} \vdash \lambda x.\mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)(x) : f \simeq f'.$$

In particular, its evaluation is a function $\Gamma, f, f' : \mathrm{Sec}_A\, B, \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')^{\simeq}, a : A \vdash \mathrm{ev}_{\lambda x.\mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)(x)}(a) : \mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))$. We can compute this explicitly as, by the computation rule, we have a homotopy $\mathrm{comp}^{\mathrm{Sec}}_{\mathrm{Id}_{\mathrm{Sec}_A\, B};A[p^3]}(\mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)) : \mathrm{Id}_{B(a)}(\mathrm{ev}_{\lambda x.\mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)(x)}(a), \mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)(a))$, where the latter is well-understood.

Function extensionality will realise the above function as an equivalence of types. However, in order to get there, it is necessary to understand how we can impose a function to be an equivalence inside a rule. The manipulation of mapping spaces will allow us to do so. Before that, we will discuss the other important

particular case of dependent mapping space, that is, when the type dependence is trivial. This introduces the notion of *functors* in our setting, and equivalences will be a special case.

2.10.4 Ordinary mapping space and functors of types

In the same way functions of types in some context have a special role among all dependent functions, as they encode the idea of equivalences, we might look at what happens when we consider the non-dependent version of the mapping space. Indeed, maps of types shall represent functors of ∞-categories, but the notion of function of types was too strict in order to do so. Thus, we take advantage of the non-strict translation of dependent functions into terms of the dependent mapping space, and we introduce the notion of *functor* of types in the theory. These should represent functions of types only up to homotopy, and, thanks to this, they will be much more suited to represent functors of ∞-categories. For this purpose, let us now introduce ordinary mapping spaces, which are the collections consisting of non-dependent functions.

Definition 2.10.4.1 (Mapping space)**.** Let $\vdash \Gamma$ anima and $\Gamma \vdash A, B$ type.

(1) The *mapping space of* A *and* B is $\Gamma \vdash \mathrm{Map}(A, B) :\equiv \mathrm{Sec}_A(B[p])$ type.

(2) A *functor* from A to B is a term $\Gamma \vdash f : \mathrm{Map}(A, B)$, which is denoted with $f : A \to B$.

Let us show some examples of how we can turn some well-known functions we met into functors.

Definition 2.10.4.2 (Identity functor)**.** Let $\vdash \Gamma$ anima and $\Gamma \vdash A$ type. Then, its universal term $\Gamma.A \vdash q : A[p]$ turns into a functor of types $1_A :\equiv \lambda(q) : A \to A$, which we call the *identity functor of* A.

Example 2.10.4.3. Let $\vdash \Gamma$ anima and $\Gamma \vdash A$ type. Then, the functorial assignment $\Gamma, a : A^{\simeq} \vdash \mathrm{ev}_a : A$ determines a functor $\gamma_A :\equiv \lambda(\mathrm{ev}) : A^{\simeq} \to A$ over Γ. Categorically-speaking, this corresponds to the inclusion of the groupoid core inside a category. Its evaluation $\mathrm{ev}_{\gamma_A}(a)$ is homotopic to the term ev_a in A that canonically corresponds to a.

Example 2.10.4.4. Let $\vdash \Gamma$ anima and let $\Gamma \vdash A$ type. Then we have a canonical functor $\lambda(!_A) : A \to \Delta^0_\Gamma$ over Γ. Its evaluation functor is the canonical, unique up to homotopy, function of types from A to Δ^0.

Construction 2.10.4.5 (The functor for function extensionality)**.** Construction 2.10.3.3 provides us with

$$\mathrm{pwise}(f, f') :\equiv \lambda\alpha.(\lambda x.\, \mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)(x)) : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')^{\simeq} \to f \simeq f'$$

over $\Gamma, f : \mathrm{Sec}_A\, B, f' : \mathrm{Sec}_A\, B$, which is the functor we want to invert through function extensionality. Its evaluation fitting into the homotopy $\Gamma, f, f' : \mathrm{Sec}_A\, B, \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')^{\simeq} \vdash \mathrm{comp}^{\mathrm{Sec}}_{\mathrm{Id}^{\simeq}_{\mathrm{Sec}_A\, B}(f,f');f\simeq f'}(\lambda x.\mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)(x)) : \mathrm{Id}_{f\simeq f'}(\mathrm{ev}_{\mathrm{pwise}(f,f')(\alpha)}, \lambda x.\mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)(x))$. We can write the evaluation of the second by Construction 2.10.3.3, from which, by functoriality of ev combined with Lemma 2.4.3.1, we obtain a witness of $\Gamma, f, f' : \mathrm{Sec}_A\, B, \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f'), a : A \vdash \mathrm{Id}_{B(a)}(\mathrm{ev}_{\mathrm{ev}_{\mathrm{pwise}(f,f')(\alpha)}}(a), \mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)(a))$ type, where the latter equality is the well-known one we induced in Remark 2.10.2.4.

Remark 2.10.4.6. Functors preserve equalities in the sense that their evaluations do so by Lemma 2.4.3.1.

The reader might notice that we can construct ordinary mapping spaces only for types over animae. It is true that for any $\Gamma \vdash S$ type with a pair $\Gamma.S \vdash A, B$ type, we can still consider the base changes $\Gamma, s : S^{\simeq} \vdash A(\mathrm{ev}_s), B(\mathrm{ev}_s)$ type and then form $\Gamma, s : S^{\simeq} \vdash \mathrm{Map}(A(\mathrm{ev}_s), B(\mathrm{ev}_s))$. However, this is a loss of information, as we lose the full functoriality in S. In fact, as we mentioned for functor categories, we can introduce local mapping spaces $\mathrm{Map}_S(-,-)$ of types over a context extensions $\Gamma.S$, where S is not an anima. The price we pay is that these will not be proper mapping spaces in the usual way, as they will not have a type dependence over S. Instead, they will be types in the base anima Γ only:

Definition 2.10.4.7 (Local mapping space)**.** Let $\vdash \Gamma$ anima and let $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type

(1) The *local mapping space of* A *and* B *over* Δ is $\Gamma \vdash \mathrm{Map}_\Delta(A, B) :\equiv \mathrm{Sec}_{\Delta.A}(B[p])$ type.

(2) A *local functor over* Δ from A to B is a term $\Gamma \vdash f : \mathrm{Map}_\Delta(A, B)$, which is again denoted with $f : A \to B$ when the context Δ is clear.

In other words, in the case Δ is a type S over Γ, we have $\mathrm{Map}_S(A,B) = \mathrm{Sec}_{s:S,a:A(s)} B(s)$, which is well-defined even when $\mathrm{Sec}_A B$ in context $\Gamma.S$ does not exist.

Example 2.10.4.8. Let $\vdash \Gamma$ anima and let $\Gamma \vdash A, B$ type. We can consider the assignment $\Gamma, f : \mathrm{Map}(A,B), x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \mathrm{ap}_{\mathrm{ev}_f}(\alpha) : \mathrm{Id}_B(\mathrm{ev}_f(x), \mathrm{ev}_f(y))$ by Lemma 2.4.3.1. Being this a function in context $\Gamma.\mathrm{Map}(A,B).A[p].A[p^2]$, we can promote it to a local functor $\Gamma \vdash \lambda(\mathrm{ap}_{\mathrm{ev}_f}(\alpha)) : \mathrm{Map}_{A[p].A[p^2]}(\mathrm{Id}_A, \mathrm{ev}_f^* \mathrm{Id}_B)$. To avoid cluttering of notation, when we start from f functor, we write $\mathrm{ap}_f : \mathrm{Id}_A(x,y) \to \mathrm{Id}_B(\mathrm{ev}_f(x), \mathrm{ev}_f(y))$ over $\Gamma, x : A, y : A$ with no ambiguity.

More in general, for any $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash C, D$ type and local functor $g : C \to D$ over $\Gamma.\Delta$, we have a local functor $\mathrm{ap}_g : \mathrm{Id}_C(x,y) \to \mathrm{Id}_D(\mathrm{ev}_g(x), \mathrm{ev}_g(y))$ over $\Gamma.\Delta, x : C, y : D$ that is a term of $\Gamma \vdash \mathrm{Map}_{\Delta.C.C[p]}(\mathrm{Id}_C, g^* \mathrm{Id}_D)$

Because we can talk about local functors over $\Gamma.S$ only as terms of a local mapping space that lives over Γ, we lose the type dependence over S as opposed to the case of functors over animae. This means that we will need to be slightly careful about when dealing with local functors, but these are a necessary part of the theory in order to capture all functions of types, and not just those over animae. In order to give a detailed treatment of both ordinary and local functors, we will usually sit in the situation of having $\vdash \Gamma$ anima and $\vdash \Gamma.\Delta$ ctx any context extension (possibly empty) over which we work. Removing the Δ's in the result will yield the non-local version. Let us focus our attention on some observations.

Remark 2.10.4.9. The reader might be doubtful about the fact that, once we have $\vdash \Gamma.\Delta$ ctx, the choice of Γ as the "anima component" seems arbitrary. For instance, Γ might be the extension of another anima, or even the whole $\Gamma.\Delta$ might be an anima. Thus, it seems that it is up to us to fix the anima over which we compute the dependent/local mapping space. More concretely, there does not seem to exist a well-defined notion of functor between $\Gamma.\Delta \vdash A, B$ type unless we specify the anima we consider it over. Thankfully, we will see that, by function extensionality, this ambiguity will be removed. In other words, all the local mapping spaces $\mathrm{Map}_{\Delta''}(A,B)$ that live in context $\Gamma.\Delta'$, where $\Delta \equiv \Delta'.\Delta''$ and $\vdash \Gamma.\Delta'$ anima have the same terms up to homotopy bijection, that are terms of $\Gamma.\Delta.A \vdash B[p]$ type. Therefore, the notion of local functor does not depend on the choice of the anima component up to homotopy. However, because we want to be precise about the homotopies, we will always point out the anima component we consider.

Remark 2.10.4.10 (Functors and functions)**.** We know that, given $\vdash \Gamma$ anima and $\Gamma \vdash A, B$ type, we have a correspondence between functions from A to B and functors from A to B. In particular, given a functor $\Gamma \vdash f : \mathrm{Map}(A,B)$ we can associate its evaluation function $\Gamma.A \vdash \mathrm{ev}_f : B[p]$, and given a function $\Gamma.A \vdash g : B[p]$ we can associate a functor $\Gamma \vdash \lambda(g) : \mathrm{Map}(A,B)$. Furthermore, there is a homotopy $\Gamma.A \vdash \mathrm{comp}^{\mathrm{Sec}}_{A;B[p]}(g) : \mathrm{Id}_{B[p]}(\mathrm{ev}_{\lambda(g)}, g)$ saying that the evaluation function of the functor associated to g is equal to g. With function extensionality, this correspondence will become a homotopy bijection. Furthermore, by Remark 2.10.2.4, we can make the first assignment functorial as $\Gamma, f : \mathrm{Map}(A,B), a : A \vdash \mathrm{ev}_f(a) : B$. The exact same idea holds for the local mapping spaces.

Because we plan to make the correspondence between functions and (local) functors up to homotopy, we want to turn the strict calculus for functions into a non-strict calculus for functors. This is a key step into the direction of unstrictifying the syntax, as it would realise functors, rather than functions of types, as the correct representation of functors of ∞-categories. This means that functors will play a key role from now on, and explains why we care more about the notion of functor than that of function of types. For that, we need to translate all the operations we can perform with functions into respective operations with functors. After we have function extensionality, it will be our goal to prove that these indeed become non-strict.

Definition 2.10.4.11 (Composition of functors)**.** Let $\vdash \Gamma$ anima, and let $\Gamma \vdash A, B, C$ type together with two functors $f : A \to B$ and $g : B \to C$. The *composition* of f and g is the functor $\lambda(\mathrm{ev}_g \circ \mathrm{ev}_f) : A \to C$.

The same constructions happens for local functors. We will soon prove that this composition is unital with respect to the identity functor. In order to promote composition of functor to a functor itself, let us show that, in general, the dependent mapping space is a functorial construction.

Construction 2.10.4.12 (Functoriality of dependent mapping space)**.** Since we introduced the dependent mapping space construction, we want to make it functorial. Let $\vdash \Gamma$ anima and consider $\Gamma \vdash A$ type and

$\Gamma.A \vdash B, C$ type. We want to lift a local functor $g : B \to C$ over A to a functor $g_* : \mathrm{Sec}_A B \to \mathrm{Sec}_A C$. Analogously, we want to construct a precomposition functor $f^* : \mathrm{Map}_A(B, C) \to \mathrm{Sec}_A C$ along any $f : \mathrm{Sec}_A B$. These will provide a composition functor where f and g are allowed to vary together. In order to do this, note that we can construct $\Gamma, g : \mathrm{Map}_A(B, C), x : A, y : B(a) \vdash \mathrm{ev}_g(y) : C(x)$, from which $\Gamma, g : \mathrm{Map}_A(B, C), f : \mathrm{Sec}_{x:A} B(x) \vdash g \circ f :\equiv \lambda a.\, \mathrm{ev}_g(\mathrm{ev}_f(a)) : \mathrm{Sec}_{x:A} C(x)$, thanks to the animation rule for $\mathrm{Map}_A(B, C)$. Now we can turn it into a functor in three ways:

(1) $\Gamma, g : \mathrm{Map}_A(B, C) \vdash g_* :\equiv \lambda f.g \circ f : \mathrm{Map}(\mathrm{Sec}_{x:A} B(x), \mathrm{Sec}_{x:A} C(x))$, which we call the *postcomposition functor*. This gives a functor $(-)_* : \mathrm{Map}_A(B, C) \to \mathrm{Map}(\mathrm{Sec}_A B, \mathrm{Sec}_A C)$. For any $g : \mathrm{Map}_A(B, C)$ its evaluation $\mathrm{ev}_{(-)_*}(g)$ is homotopic to $\Gamma \vdash g_* :\equiv \lambda f.g \circ f : \mathrm{Map}(\mathrm{Sec}_{x:A} B(x), \mathrm{Sec}_{x:A} C(x))$, which we call the *postcomposition functor with* g. In particular, the evaluation of g_* at any $\Gamma \vdash f : \mathrm{Sec}_{x:A} B(x)$ is equal to $g \circ f$, and fits into a homotopy $\Gamma, a : A \vdash \mathrm{evpost}_{g,f}(a) : \mathrm{Id}_{B(a)}(\mathrm{ev}_{\mathrm{ev}_{g_*}(f)}(a), \mathrm{ev}_g(\mathrm{ev}_f(a)))$.

(2) $\Gamma, f : \mathrm{Sec}_{x:A} B(x) \vdash f^* :\equiv \lambda g.g \circ f : \mathrm{Map}(\mathrm{Map}_A(B, C), \mathrm{Sec}_{x:A} C(x))$, which we call the *precomposition functor*. This gives a functor $(-)^* : \mathrm{Sec}_A B \to \mathrm{Map}(\mathrm{Map}_A(B, C), \mathrm{Sec}_A C)$. For any $f : \mathrm{Sec}_A C$ its evaluation $\mathrm{ev}_{(-)^*}(f)$ is homotopic to $\Gamma \vdash f^* :\equiv \lambda g.g \circ f : \mathrm{Map}(\mathrm{Sec}_{x:A} B(x), \mathrm{Sec}_{x:A} C(x))$, which we call the *precomposition functor with* f. In particular, the evaluation of f_* at any $\Gamma \vdash g : \mathrm{Map}_A(B, C)$ is equal to $g \circ f$, and fits into a homotopy $\Gamma, a : A \vdash \mathrm{evpre}_{f,g}(a) : \mathrm{Id}_{B(a)}(\mathrm{ev}_{\mathrm{ev}_{f^*}(g)}(a), \mathrm{ev}_g(\mathrm{ev}_f(a)))$.

(3) By construction of the product, composition provides a functor $- \circ - :\equiv \lambda(f, g).g \circ f : \mathrm{Map}_A(B, C) \times \mathrm{Sec}_A B \to \mathrm{Sec}_A C$, which we call the *composition functor*. In particular, for any $f : \mathrm{Map}(\mathrm{Sec}_A B, \mathrm{Sec}_A C)$ and $g : \mathrm{Map}_A(B, C)$, its evaluation $\mathrm{ev}_{-\circ-}(g, f)$ is homotopic to $g \circ f$ in $\mathrm{Sec}_A C$.

Let us look at the non-dependent case, and induce the intuitive composition functors between ordinary (or local) mapping spaces we expect our theory to have.

Construction 2.10.4.13 (Ordinary composition functors)**.** Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B, C$ type. We have an assignment $\Gamma, f : \mathrm{Map}_\Delta(A, B), g : \mathrm{Map}_\Delta(B, C) \vdash g \circ f : \mathrm{Map}_\Delta(A, C)$ of the composite functor.

(1) By regarding it as a function of types in f over $\Gamma.\mathrm{Map}_\Delta(A, B)$, for any $g : B \to C$ we obtain a *postcomposition functor with* g of the form $\Gamma \vdash g_* : \mathrm{Map}(\mathrm{Map}_\Delta(A, B), \mathrm{Map}_\Delta(A, C))$, which realises the assignment $f \mapsto g \circ f$.

(2) By regarding it as a function of types in g over $\Gamma.\mathrm{Map}_\Delta(B, C)$, for any $f : A \to B$ we obtain a *precomposition functor with* f, of the form $\Gamma \vdash f^* : \mathrm{Map}(\mathrm{Map}_\Delta(B, C), \mathrm{Map}_\Delta(A, C))$, which realises the assignment $g \mapsto g \circ f$.

(3) By the induction principle of the dependent sum, it defines a composition function $\Gamma.\mathrm{Map}_\Delta(A, B) \times \mathrm{Map}_\Delta(B, C) \vdash - \circ - : \mathrm{Map}_\Delta(A, C)$, which then gives rise to a *composition functor* $\Gamma \vdash \mathrm{comp} :\equiv \lambda(- \circ -) : \mathrm{Map}(\mathrm{Map}_\Delta(A, B) \times \mathrm{Map}_\Delta(B, C), \mathrm{Map} : \Delta(A, C))$, which realises the assignment $(f, g) \mapsto g \circ f$, i.e. $\mathrm{ev}_{-\circ-}(f, g)$ is homotopic to $g \circ f$.

Remark 2.10.4.14. Observe that $(-)_*$ is compatible with homotopy by Lemma 2.4.3.1, because we have $\Gamma, g : \mathrm{Map}_A(B, C) \vdash g_* :\equiv \lambda f.g \circ f : \mathrm{Map}(\mathrm{Sec}_{x:A} B(x), \mathrm{Sec}_{x:A} C(x))$. In particular, we have $\Gamma, g, g' : \mathrm{Map}_A(B, C), \beta : \mathrm{Id}_{\mathrm{Map}_A(B,C)} \vdash \mathrm{ap}_{(-)_*}(\beta) : \mathrm{Id}_{\mathrm{Map}(\mathrm{Sec}_A B, \mathrm{Sec}_A C)}(g_*, g'_*)$. The same holds for precomposition.

We can already show some properties about the evaluation functions, such as the fact that its compatibility with composition of functors and with identity.

Lemma 2.10.4.15 (Evaluation respects composition and identity)**.** Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B, C$ type. We have the following homotopies:

(1) *Evaluation of* $- \circ -$*:* $\Gamma.\Delta, a : A, f : \mathrm{Map}_\Delta(A, B), g : \mathrm{Map}_\Delta(B, C) \vdash \mathrm{evcomp}_{f,g}(a) :\equiv \mathrm{comp}^{\mathrm{Sec}}_{\Delta.A;C[p]}(g \circ f)(a) : \mathrm{Id}_C(\mathrm{ev}_{g\circ f}(a), \mathrm{ev}_g(\mathrm{ev}_f(a)))$.

(2) *Evaluation of* 1_A*:* $\Gamma.\Delta, a : A \vdash \mathrm{evid}_A(a) :\equiv \mathrm{comp}^{\mathrm{Sec}}_{\Delta.A;A[p]}(a) : \mathrm{Id}_{A[p]}(\mathrm{ev}_{1_A}(a), a)$.

(3) $\Gamma.\Delta, f : \mathrm{Map}_\Delta(A, B), a : A \vdash \mathrm{ap}_{\mathrm{ev}_f}(\mathrm{evid}_A(a)) \circ \mathrm{evcomp}_{1_A,f} : \mathrm{Id}_{A[p]}(\mathrm{ev}_{f\circ 1_A}(a), \mathrm{ev}_f(a))$.

(4) $\Gamma.\Delta, f : \mathrm{Map}_\Delta(A, B), a : A \vdash \mathrm{evid}_B(f(a)) \circ \mathrm{evcomp}_{f,1_B} : \mathrm{Id}_{A[p]}(\mathrm{ev}_{1_B\circ f}(a), \mathrm{ev}_f(a))$.

Proof. For (1), note that, by definition, $g \circ f \equiv \lambda(\mathrm{ev}_g \circ \mathrm{ev}_f)$, and thus by the computation rule we have a homotopy $\Gamma.\Delta, x : A \vdash \mathrm{comp}^{\mathrm{Sec}}_{A,C[p]}(g \circ f)(x) : \mathrm{Id}_C(\mathrm{ev}_{g\circ f}(x), \mathrm{ev}_g \circ \mathrm{ev}_f(x))$. The construction on the universal

terms yields the desired homotopy. (2) analogously follows by definition of identity together with the computation rule. (3) and (4) follow by (1) and (2) combined, and are defined as in the claim by making use of Lemma 2.4.3.1. □

In order to even prove that the composition is associative, and only up to homotopy, we need to dig into the notion of equality inside mapping spaces, which is determined by function extensionality. This is because, up to now, we have no way of producing homotopies inside the dependent mapping space.

2.10.5 Equivalence types

We now exploit a further advantage of having a type of functors, namely the fact that we can construct new types out of it. The introduction of a type "collection of maps", indeed, allows us to introduce a type "collection of proofs of f being an equivalence" for a functor f. This will be extremely useful to impose new rules, first of which, the function extensionality property for functors. However, we have to clash with the fact that, given a functor f, there is more than one way it can be an equivalence: pointwise or globally, as we will now see.

Definition 2.10.5.1 (Global and pointwise section/retraction)**.** Let $\vdash \Gamma$ anima and let $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type. Let $f : A \to B$ be a functor over $\Gamma.\Delta$.

(1) A *global retraction* of f is a functor $g : B \to A$ over $\Gamma.\Delta$ with a homotopy $\Gamma \vdash l : \mathrm{Id}_{\mathrm{Map}_\Delta(A,A)}(g \circ f, 1_A)$.

(2) A *pointwise retraction* of f is a functor $g : B \to A$ over $\Gamma.\Delta$ with a homotopy $\Gamma, a : A \vdash l(a) : \mathrm{Id}_A(\mathrm{ev}_g(\mathrm{ev}_f(a)), a)$, i.e. the function ev_g is a retraction of ev_f. Equivalently, by Lemma 2.10.4.15, $g \circ f$ is pointwise homotopic to 1_A.

(3) A *global section* of f is a functor $h : B \to A$ over $\Gamma.\Delta$ with a homotopy $\Gamma \vdash r : \mathrm{Id}_{\mathrm{Map}_\Delta(B,B)}(f \circ h, 1_B)$.

(4) A *pointwise retraction* of f is a functor $h : B \to A$ over $\Gamma.\Delta$ with a homotopy $\Gamma, b : B \vdash r(b) : \mathrm{Id}_B(\mathrm{ev}_f(\mathrm{ev}_h(a)), a)$, i.e. the function ev_h is a section of ev_f. Equivalently, by Lemma 2.10.4.15, $f \circ h$ is pointwise homotopic to 1_A.

This immediately yields the two following notions of equivalence.

Definition 2.10.5.2 (Global and pointwise equivalence)**.** Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type.

(1) A functor $f : A \to B$ over $\Gamma.\Delta$ is a *global equivalence* if it admits a global retraction and a global section, i.e. two functors $g, h : B \to A$ over $\Gamma.\Delta$ with two homotopies $\Gamma \vdash l : \mathrm{Id}_{\mathrm{Map}_\Delta(A,A)}(g \circ f, 1_A)$ and $\Gamma \vdash r : \mathrm{Id}_{\mathrm{Map}_\Delta(B,B)}(f \circ g, 1_B)$.

(2) A functor $f : A \to B$ over $\Gamma.S$ is a *pointwise equivalence* if it admits a pointwise retraction and a pointwise section, i.e. two functors $g, h : B \to A$ over $\Gamma.\Delta$ with two homotopies $\Gamma, a : A \vdash l(a) : \mathrm{Id}_A(\mathrm{ev}_g(\mathrm{ev}_f(a)), a)$ and $\Gamma, b : B \vdash r(b) : \mathrm{Id}_B(\mathrm{ev}_f(\mathrm{ev}_h(b)), b)$. This is equivalent to saying that $g \circ f$ and $f \circ h$ are pointwise homotopic to 1_A and 1_B respectively by Lemma 2.10.4.15.

Although the pointwise inverses are required to come from functors, the latter condition coincides with the notion of equivalence of types we have been using previously. Indeed:

Lemma 2.10.5.3. Let $\vdash \Gamma$ anima and let $\Gamma \vdash \Delta$ type and $\Gamma.\Delta \vdash A, B$ type. Consider a functor $f : A \to B$ over $\Gamma.\Delta$. Then, a pointwise retraction (section) of f is the same as a retraction (section) of the function ev_f. In particular, f is a pointwise equivalence if and only if the function ev_f is an equivalence in the usual sense.

Proof. A pointwise retraction (section) $g : B \to A$ provides ev_g as a retraction (section) of ev_f. Conversely, if ev_f admits a retraction function $\Gamma.\Delta.B \vdash g'.A[p]$, the associated $\Gamma \vdash \lambda(g') : \mathrm{Map}_\Delta(B, A)$ is a pointwise inverse of ev_f because we have $\Gamma.\Delta, b : B \vdash \mathrm{comp}^{\mathrm{Sec}}_{\Delta.B;A[p]}(b) : \mathrm{Id}_A(\mathrm{ev}_{\lambda(g')}(b), g'(b))$ by the computation rule. Whiskering this equality with ev_f yields the claims. □

The reader might be confused by the fact that we now have two notions of equivalence of types. The pointwise equivalence of types is the notion we have been using up to now, as it was the only one we could express before, but the introduction of mapping spaces carries a new variant into the picture, i.e. the global

equivalence. Although these look similar, they do not coincide a priori, as we miss the characterisation of the identity type of the dependent mapping space. We will soon see that, a priori, this new condition of being "globally equivalent" is stronger that of being "pointwise equivalent". Later, as we might expect, function extensionality will reverse the implication.

Remark 2.10.5.4. By Lemma 2.10.5.3, all the results of compatibility with equivalences, such as Proposition 2.4.11.7 and Proposition 2.6.3.2, translate into the setting of pointwise equivalences between types.

The next step is to assemble each given definition to a type, thanks to the existence of mapping spaces. In particular, we want to construct an equivalence type, exhibiting a given functor as a pointwise or functorial equivalence. In particular, we construct an equivalence type in multiple ways. This is extremely useful as we will inherit the feature of writing rules that impose some functor being an equivalence, and will open up the way to introducing further types that break down to some map being an equivalence.

Construction 2.10.5.5 (The section and retraction types)**.** Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type, with a functor $f : A \to B$ over $\Gamma.\Delta$. We define the following types:

(a) $\Gamma \vdash \text{has-}\simeq\text{-Retr}_\Delta(f) :\equiv \Sigma_{g:\text{Map}_\Delta(B,A)}\, g \circ f \simeq_\Delta 1_A$. Its terms are in correspondence with pointwise retractions of f.

(b) $\Gamma \vdash \text{has-}=\text{-Retr}_\Delta(f) :\equiv \Sigma_{g:\text{Map}_\Delta(B,A)}\, \text{Id}_{\text{Map}_\Delta(A,A)}(g \circ f, 1_A)^\simeq$. Its terms are in correspondence with global retractions of f.

(c) $\Gamma \vdash \text{has-}\simeq\text{-Sect}_\Delta(f) :\equiv \Sigma_{h:\text{Map}_\Delta(B,A)}\, f \circ h \simeq_\Delta 1_B$. Its terms are in correspondence with pointwise sections of f.

(d) $\Gamma \vdash \text{has-}=\text{-Sect}_\Delta(f) :\equiv \Sigma_{h:\text{Map}_\Delta(B,A)}\, \text{Id}_{\text{Map}_\Delta(B,B)}(f \circ h, 1_B)^\simeq$. Its terms are in correspondence with global sections of f.

Function extensionality will characterise the correspondences as actual homotopy bijections. Note that in (b) and (d), the lack of a homotopy bijection a priori is due to the choice of putting the groupoid core on the identity type. This will turn out to be irrelevant as we will prove in the future, but is needed for the moment. We will often consider the case $\Delta \equiv \emptyset$, i.e. of the trivial extension of Γ.

Construction 2.10.5.6 (The equivalence types)**.** Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type with a functor $f : A \to B$. We define the following types:

(1) $\Gamma \vdash \text{is-}\simeq\text{-Equiv}_\Delta(f) :\equiv \text{has-}\simeq\text{-Retr}_\Delta(f) \times \text{has-}\simeq\text{-Sect}_\Delta(f)$. Its terms are in correspondence with the datum of two functors $g, h : B \to A$ together with *pointwise* homotopies from $g \circ f$ to 1_A and from $f \circ h$ to 1_B respectively. In other words, a term of this type exhibits f as a *pointwise* equivalence of types.

(2) $\Gamma \vdash \text{is-}=\text{-Equiv}_\Delta(f) :\equiv \text{has-}=\text{-Retr}(f) \times \text{has-}=\text{-Sect}(f)$. Its terms are in correspondence with the datum of two functors $g, h : B \to A$ together with *global* homotopies from $g \circ f$ to 1_A and from $f \circ h$ to 1_B respectively. In other words, a term of this type exhibits f as a *global* equivalence of types.

(3) $\Gamma \vdash \text{has-}\simeq\text{-Inv}(f) :\equiv \Sigma_{g:\text{Map}_\Delta(B,A)}(g \circ f \simeq 1_A \times f \circ g \simeq 1_B)$. Its terms are in correspondence with the datum of a functor $g : B \to A$ over $\Gamma.\Delta$ that is both a pointwise section and a pointwise retraction of f.

(4) $\Gamma \vdash \text{has-}=\text{-Inv}(f) :\equiv \Sigma_{g:\text{Map}_\Delta(B,A)}\, \text{Id}_{\text{Map}(A,A)}(g \circ f, 1_A)^\simeq \times \text{Id}_{\text{Map}(B,B)}(f \circ g, 1_B)^\simeq$. Its terms are in correspondence with the datum of a functor $g : B \to A$ over $\Gamma.\Delta$ that is both a global section and a global retraction of f.

Remark 2.10.5.7. We will soon prove that the mentioned correspondences are in fact homotopy bijections. Furthermore, (3) and (4) are variants of (1) and (2) respectively, where we force the two one-sided inverses to coincide. The two pairs of types will in fact be logically equivalent, although showing this for (2) and (4) will require function extensionality. However, these will not be equivalences of types. Indeed, (1) and (2) will fall in the class of tpes which we will name *(mere) propositions*, as we will soon prove, as opposed to (3) and (4). These will be types which are contractible whenever they are inhabited, which means that we care more about the existence or not of an inhabitant, rather than the provision of a specific on. Roughly speaking, this would promote the condition of a functor being an equivalence to an actual property rather than data, up to homotopy. In other words, the section and retraction exhibiting it as an equivalence will not matter, as they will be uniquely determined. However, forcing the two inverses to coincides, makes the condition of having an inverse not a property of the functor, but the provision of data. The reason is that

higher equalities are not uniquely determined. One can show, following [BGL+17], that imposing an odd number of coherences (such as one of the triangle identities for the homotopies) for the two inverses yields a proposition, but an even number makes this property disappear. For this reason, the types (1) and (2) are the ones of our main interest. Another interesting question we will explore in the next paragraphs will be how the two types (1) and (2), i.e. these two notion of equivalence, compare.

Let us start by showing the relations between these types that we already have a priori, before having access to function extensionality. Because these types are explicitly introduced as statements, we interpret functions between them as logical implications.

Lemma 2.10.5.8. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type.

(1) The types is-$\simeq$-Equiv$_\Delta(f)$ and has-$\simeq$-Inv$_\Delta(f)$ are logically equivalent over $\Gamma, f : \mathrm{Map}_\Delta(A, B)$.

(2) There is a functor is-$=$-Retr$_\Delta(f) \to$ is-$\simeq$-Retr$_\Delta(f)$ over Γ. In other words, being a global retraction implies being a pointwise retraction.

(3) There is a functor is-$=$-Sect$_\Delta(f) \to$ is-$\simeq$-Sect$_\Delta(f)$ over Γ. In other words, being a global section implies being a pointwise equivalence.

(4) There is a functor is-$=$-Equiv$_\Delta(f) \to$ is-$\simeq$-Equiv$_\Delta(f)$ over Γ. In other words, being a global equivalence implies being a pointwise section.

Proof. There is an obvious functor has-$\simeq$-Inv$_\Delta(f) \to$ is-$\simeq$-Equiv$_\Delta(f)$. Conversely, we can construct a functor in the opposite direction by means of Remark 2.4.11.4: we know that if we have a pointwise section and a pointwise retraction of f, then their evaluations coincide up to homotopy, and in particular they are two-sided pointwise inverses. This proves (1). For (2), by Construction 2.10.4.5 we have the functor $\mathrm{pwise}(g \circ f, 1_A) : \mathrm{Id}_{\mathrm{Map}_\Delta(A,B)}(g \circ f, 1_A)^{\simeq} \to \mathrm{Sec}_{\Delta, a:A} \mathrm{Id}_B(\mathrm{ev}_{g\circ f}(a), \mathrm{ev}_{1_A}(a))$ over $\Gamma, f : \mathrm{Map}_\Delta(A, B), g : \mathrm{Map}_\Delta(B, A)$ Because we have an equivalence $\mathrm{Id}_B(\mathrm{ev}_{g\circ f}, \mathrm{ev}_{1_A}) \simeq \mathrm{Id}_B(\mathrm{ev}_g(\mathrm{ev}_f(a)), a)$ by Proposition 2.4.7.3 and Lemma 2.10.4.15, taking the dependent sum over $g : \mathrm{Map}_\Delta(B, A)$ and using Proposition 2.6.3.2 yields the desired functor. (3) follows by an analogous argument, and combining it with (2) yields (4). □

As already mentioned, function extensionality will allow us to prove a result analogous to (1) for the types is-$=$-Equiv$_\Delta(f)$ and has-$=$-Inv$_\Delta(f)$, and to exhibit (2), (3) and thus (4) as an equivalence of types. In particular, global equivalence and pointwise equivalence will be equivalent not only logically, but also as types. For the moment, being a global equivalence of types will be a stronger condition than being a pointwise equivalences, which is the well-understood notion we have been using until now.

The equivalence types open up for the introduction of various new type, namely those predicates that are determined by a functor being an equivalence. For instance, the property of being an embedding, being a homotopy cartesian square or being contractible reduce to showing that some function is an equivalence. Therefore, we can introduce types representing, and finally internalising, these notions. However, as we saw two kinds of equivalence types, we would get two variants representing each of the notions we mentioned as well. We will often choose to present the pointwise only, as the choice is not going to matter once we have function extensionality.

Construction 2.10.5.9 (The contractibility types)**.** Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A$ type. Consider the canonical functor $!_A : A \to \Delta^0_{\Gamma.\Delta}$. We define the following types:

(1) $\Gamma \vdash$ is-$\simeq$-Contr$_\Delta(A) :\equiv$ has-$\simeq$-Retr$_\Delta(!_A)$ type. Its terms correspond to pointwise retractions of $!_A$.

(2) $\Gamma \vdash$ is-$=$-Contr$_\Delta(A) :\equiv$ has-$=$-Retr$_\Delta(!_A)$ type. Its terms correspond to global retractions of $!_A$.

Construction 2.10.5.10 (The isEmb type)**.** Let $\vdash \Gamma$ anima, with $\Gamma \vdash A, B$ type together with a functor $f : A \to B$. Consider the functor $\Gamma \vdash \mathrm{ap}_f : \mathrm{Map}_{A\times A}(\mathrm{Id}_A(a, a'), \mathrm{Id}_B(f(a), f(a')))$ from Example 2.10.4.8. We define

$$\Gamma \vdash \mathrm{isEmb}(f) :\equiv \text{is-}\simeq\text{-Equiv}_{A\times A}(\mathrm{ap}_f)\,\mathrm{type}\,.$$

A witness of this type is the exhibition of ap_f as a pointwise equivalence from $\mathrm{Id}_A(a, a')$ to $\mathrm{Id}_B(f(a), f(a'))$, which is exactly the exhibition of f as an embedding of types.

Construction 2.10.5.11 (The isCart type)**.** Let $\vdash \Gamma$ anima and consider a commutative square

$$\begin{array}{ccc} D & \xrightarrow{\alpha} & A \\ {\scriptstyle\beta}\downarrow & & \downarrow{\scriptstyle f} \\ B & \xrightarrow[g]{} & C \end{array}$$

over it. This is the same as providing the associated function $(\alpha, \beta) : D \to A \times_C B$. We construct the type $\Gamma \vdash \mathrm{isCart}(\alpha, \beta) :\equiv \text{is-}\simeq\text{-Equiv}(\alpha, \beta)$. Its terms are proofs of the map being a pointwise equivalence, i.e. of the square being homotopy cartesian.

These types are extremely important, as we are now allowed to can introduce rules stating equivalences or homotopy cartesianity of commutative squares.

2.10.6 Function extensionality for dependent mapping spaces

We are now ready to describe the notion of equality in mapping spaces by stating the function extensionality rule. Since its terms are in correspondence with functions, we expect to recover the same notion of equality of functions we have been using until now, i.e. pointwise equality. In Remark 2.10.2.4 we saw that homotopy in the dependent mapping space implies pointwise homotopy of the evaluation functions. This is encoded by Construction 2.10.4.5, where, given $\vdash \Gamma$ anima $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type, we were able to construct a functor

$$\mathrm{pwise}(f, f') : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')^{\simeq} \to f \simeq f'$$

in context $\Gamma, f, f' : \mathrm{Sec}_A\, B$. In particular, we saw that this extends Remark 2.10.2.4 in the sense that for any $\Gamma \vdash \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ and $\Gamma \vdash a : A$ we have a homotopy from $\mathrm{ev}_{\mathrm{ev}_{\mathrm{pwise}(f,f')(\alpha)}(a)}$ to $\mathrm{ap}_{\mathrm{ev}}(\mathrm{ev}_\alpha)(a)$ in $B(a)$.

Now we would like to reverse this implication: we would expect a homotopy $\Gamma \vdash \alpha : \mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ to be *the same* as a pointwise homotopy $\Gamma, x : A \vdash \alpha(x) : \mathrm{Id}_B(f(x), f'(x))$. This largely anticipated property carries the name of *function extensionality*, and would finally give a tool to provide homotopies inside the mapping spaces. The name has a clear interpretation, and derives from a conceptually-analogous rule in homotopy type theory, where mapping spaces are replaced with internal Homs (and more in general with arbitrary dependent products).

Remark 2.10.6.1. It is well-known that, in homotopy type theory, function extensionality is implied by univalence. However, in the present setting we do not have a homotopy type theoretical univalence. Therefore, we must impose function extensionality as a reasonable rule in the theory.

We can construct the above functor more in general for every context extension of Γ. The *function extensionality rule* reverses the implication by imposing the functor to be an equivalence:

$$\frac{\vdash \Gamma \text{ anima} \qquad \vdash \Gamma.\Delta \text{ ctx} \qquad \Gamma.\Delta \vdash B \text{ type}}{\Gamma.\mathrm{Sec}_\Delta\, B.(\mathrm{Sec}_\Delta\, B)[p] \vdash \mathrm{funext}_{\Delta;B} : \text{is-}\simeq\text{-Equiv}(\mathrm{pwise}(q[p], q))}.$$

This, when $\vdash \Gamma$ anima, $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash B$ type, realises a pointwise equivalence of types

$$\mathrm{pwise}(f, f') : \mathrm{Id}_{\mathrm{Sec}_\Delta\, B}(f, f')^{\simeq} \xrightarrow{\sim} f \simeq f'$$

in context $\Gamma, f, f' : \mathrm{Sec}_\Delta\, B$. Note that the type $\text{is-}\simeq\text{-Equiv}(\mathrm{pwise}(f, f'))$ is well-defined because, by the animation rule, it is constructed over an anima context. Therefore, we do not need to consider the local version. In particular, because this rule provides a term, it will come equipped with a further rule making it compatible with base change along animae.

Remark 2.10.6.2. The reader experienced in homotopy type reader may notice that, up to replacing Sec with Π, we recover the ordinary function extensionality for dependent products, which motivated us to introduce this rule, with a peculiar adjustment: we are describing the *groupoid core* of the identity type of the mapping space $\mathrm{Id}_{\mathrm{Sec}_A\, B}(f, f')$ as the mapping space of the identity type $\mathrm{Sec}_{a:A}(\mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a)))$.

Of course we could not see such a difference in homotopy type theory, because the groupoid core is the identical operator. However, in similarity with the homotopy type theoretical rule (under replacement of Sec with Π) we would like to recover *this* stronger rule for mapping spaces without any groupoid core. We will soon see that groupoid core operator is inert on animae and conversely, so that this stronger condition can coexist with the one we found if and only if the fibres of the identity type $\mathrm{Id}_{\mathrm{Sec}_A\ B}(f, f')$ are animae. This is something we do not know yet a priori (and in homotopy type theory this problem does not arise as every context is an anima). Yet, we would like it to be true, and we will show it once we will have a criterion to prove that some type is an anima.

The reader might be surprised by the fact that we were able to construct the functor of function extensionality only with the appearance of the groupoid core. However, it is correct in principle: up to now we might be doing (∞, n)-category theory, where it is not true that the fibres of the identity type are animae. This will follow from rules that make the theory we present differ from (∞, n)-category theory.

Function extensionality rule finally completes the rules of the dependent mapping space. For the rest of this paragraph, we aim to show its first consequences and how it solves some problems we addressed. After that, we will discuss further consequences that deserve a dedicated paragraph.

Note that, for any $\vdash \Gamma$ anima, $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash B$ type, function extensionality provides a a quasi-inverse function $\Gamma, f, f' : \mathrm{Sec}_\Delta B, \alpha : f \simeq f' \vdash \mathrm{extend}_{f,f'}(\alpha) : \mathrm{Id}_{\mathrm{Sec}_A\ B}(f, f')^\simeq$ of the evaluation of $\mathrm{pwise}(f, f') : \mathrm{Id}_{\mathrm{Sec}_\Delta\ B}(f, f')^\simeq \xrightarrow{\sim} f \simeq f'$ by construction. For the moment, until we prove that this choice till not matter, we will fix this inverse assignment. In particular, we have an inhabitant of $\Gamma.\Delta, f, f' : \mathrm{Sec}_\Delta B, \alpha : f \simeq f' \vdash \mathrm{Id}_{f \simeq f'}(\mathrm{ev}_{\mathrm{pwise}(f,f')}(\mathrm{extend}_{f,f'}(\alpha)), \alpha)$ type. We will soon spell-out the meaning of this identity type.

Homotopies in mapping spaces First of all, the naive interpretation of function extensionality is that we can deduce that two functors with pointwise homotopic evaluations are themselves homotopic.

Remark 2.10.6.3 (Pointwise homotopy extends to homotopy in Sec)**.** Let $\vdash \Gamma$ anima, with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash B$ type. Function extensionality determines the quasi-inverse functor $f \simeq f' \to \mathrm{Id}_{\mathrm{Sec}_A\ B}(f, f')$, which we can postcompose with $\mathrm{Id}_{\mathrm{Sec}_A\ B}(f, f')^\simeq \to \mathrm{Id}_{\mathrm{Sec}_A\ B}(f, f')$. The underlying evaluation function, by Lemma 2.10.4.15 carries $\alpha : f \simeq f'$ into the image of $\mathrm{extend}_{f,f'}(\alpha)$ under the evaluation of $\mathrm{Id}_{\mathrm{Sec}_A\ B}(f, f')^\simeq \to \mathrm{Id}_{\mathrm{Sec}_A\ B}(f, f')$, which we may denote by $\overline{\mathrm{extend}_{f,f'}(\alpha)} : \mathrm{Id}_{\mathrm{Sec}_A\ B}(f, f')$.

Remark 2.10.6.3 means that the function extensionality rule does indeed extend pointwise equality of dependent functions to global equality in the dependent mapping space. We will soon prove that this implication will not explicitly depend on the choice of the quasi-inverse provided by function extensionality. Secondly, function extensionality finally allows to finally determine all terms of the dependent mapping space up to homotopy, by promoting the correspondence provided by the rules to a homotopy bijection:

Lemma 2.10.6.4. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash B$ type. Then, for any $\Gamma \vdash f : \mathrm{Sec}_\Delta B$, we have an inhabitant of $\Gamma \vdash \mathrm{Id}_{\mathrm{Sec}_\Delta\ B}(\lambda(\mathrm{ev}_f), f)$ type. In particular, we have a homotopy bijection between terms of $\Gamma.\Delta \vdash B$ and terms of $\Gamma \vdash \mathrm{Sec}_\Delta B$.

Proof. For simplicity we assume $\Gamma \vdash A \equiv \Delta$ type. By function extensionality, it suffices to provide an equality between the evaluation functions. Indeed, we have $\Gamma, a : A \vdash \mathrm{comp}^{\mathrm{Sec}}_{A;B}(\mathrm{ev}_f)(a) : (\mathrm{ev}_{\lambda x.\, \mathrm{ev}_f(x)}(a), \mathrm{ev}_f(a))$. Therefore, the fixed inverse functor yields global homotopy from $\lambda x.\, \mathrm{ev}_f(x)$ to f as claimed. □

In particular in the non-dependent case, we can finally describe the correspondence between functions and functors:

Corollary 2.10.6.5 (Functors and functions are in homotopy bijection)**.** Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx, and let $\Gamma.\Delta \vdash A, B$ type. Then, we have a homotopy bijection between terms of $\Gamma \vdash \mathrm{Map}_\Delta(A, B)$ type, namely (local) functors $A \to B$, and terms of $\Gamma.\Delta.A \vdash B[p]$ type, namely functions of types from A to B.

As a further corollary of Lemma 2.10.6.4, we can turn the correspondences in Construction 2.10.5.5, Construction 2.10.5.6 and Construction 2.10.5.9 into actual homotopy bijections, to have a full understanding of the retraction, section, equivalence and contractibility types. Analogous considerations hold for Construction 2.10.5.10 and Construction 2.10.5.11.

Corollary 2.10.6.6. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type. For any functor $f : A \to B$:

(1) The terms of $\Gamma \vdash$ has-$\simeq$-Retr$_\Delta(f)$ type are in homotopy bijection with pointwise retractions.
(2) The terms of $\Gamma \vdash$ has-$=$-Retr$_\Delta(f)$ type are in homotopy bijection with global retractions.
(3) The terms of $\Gamma \vdash$ has-$\simeq$-Sect$_\Delta(f)$ type are in homotopy bijection with pointwise sections.
(4) The terms of $\Gamma \vdash$ has-$=$-Sect$_\Delta(f)$ type are in homotopy bijection with global sections.
(5) The terms of $\Gamma \vdash$ is-$\simeq$-Equiv$_\Delta(f)$ type are in homotopy bijection with proofs of f being a pointwise equivalence.
(6) The terms of $\Gamma \vdash$ is-$=$-Equiv$_\Delta(f)$ type are in homotopy bijection with proofs of f being a global equivalence.
(7) The terms of $\Gamma \vdash$ is-$\simeq$-Contr$_\Delta(A)$ are in homotopy bijection with pointwise retractions of $A \to \Delta^0_\Gamma$.
(8) The terms of $\Gamma \vdash$ is-$=$-Contr$_\Delta(A)$ are in homotopy bijection with global retractions of $A \to \Delta^0_\Gamma$.

Two notions of equivalences Among the various consequences, function extensionality allows us to prove in a straightforward way that the types is-$\simeq$-Equiv(f) and is-$=$-Equiv(f) are equivalent. In particular, this says that we can go back and forth between data exhibiting a functor f as a global equivalence or as a pointwise equivalence, which is the notion of equivalence we have been using up to here.

Proposition 2.10.6.7. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type.

(1) The functor has-$=$-Retr$_\Delta(f) \to$ has-$\simeq$-Retr$_\Delta(f)$ of Lemma 2.10.5.8 is an equivalence of types.
(2) The functor has-$=$-Sect$_\Delta(f) \to$ has-$\simeq$-Sect$_\Delta(f)$ of Lemma 2.10.5.8 is an equivalence of types.
(3) The functor has-$=$-Equiv$_\Delta(f) \to$ has-$\simeq$-Equiv$_\Delta(f)$ of Lemma 2.10.5.8 is an equivalence of types.

Proof. By construction, these functors are induced in Proposition 2.6.3.1 by taking the dependent sum of the functors claimed to be equivalences by function extensionality. Therefore, by Proposition 2.6.3.2 we conclude the claim. □

Note, in particular, that we could have stated the function extensionality rule by means of the equivalence type is-$=$-Equiv(f) as every deduction we made would still apply by Lemma 2.10.5.8, included Proposition 2.10.6.7. In particular, this result is independent on the choice of the equivalence type in the function extensionality rule.

Iterated dependent mapping space Function extensionality allows us to show that the dependent mapping space is distributive over the context.

Proposition 2.10.6.8. Let $\vdash \Gamma$ anima and let $\vdash \Gamma.\Delta_1 \dots \Delta_i$ anima for every $i \in \{1, \dots, n\}$. Let $\Delta :\equiv \Delta_1 \dots \Delta_n$, and consider $\Gamma.\Delta \vdash B$ type. Then, we have an equivalence $\mathrm{Sec}_\Delta B \simeq \mathrm{Sec}_{\Delta_1} \dots \mathrm{Sec}_{\Delta_n} B$ of types over Γ.

Proof. The terms of both types are in homotopy bijection with the terms of $\Gamma.\Delta \vdash B$ type by Lemma 2.10.6.4. This provides an equivalence of types. □

The reader should observe that, in Proposition 2.10.6.8, we showed $\mathrm{Sec}_\Delta B \simeq \mathrm{Sec}_{\Delta_1} \dots \mathrm{Sec}_{\Delta_n} B$ of types over Γ, where the first type is always defined as long as Γ is an anima, whereas the latter needs the extra assumption that $\vdash \Gamma.\Delta_1 \dots \Delta_i$ anima for every $i \in \{1, \dots, n\}$. However, in homotopy type theory we can interpret dependent mapping spaces as dependent products, which always exist. We can then understand better how to regard the poverty of our theory and why it will not often be a problem.

Remark 2.10.6.9. We said that the dependent mapping space is different from the dependent product we will define, coming from to the fact that the first is a "collection" while the second will not. Still, this distinction is not seen in homotopy type theory, where the function type corresponds to the ordinary mapping space. In other words, we can think of dependent products from homotopy type theory as being translated in two distinct ways: into dependent mapping spaces or into dependent products. For this discussion, let

us interpret dependent mapping spaces (and constructions with them) as the borrowing of some dependent products (and constructions with them) from homotopy type theory. In this way, we can see with more clarity, out of all the constructions usually allowed in homotopy type theory, which ones we can perform in the present setting. Consider $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash B$ type, where $\Delta \equiv \Delta_1 \dots \Delta_n$. In our theory, we can form $\Gamma \vdash \mathrm{Sec}_\Delta B$ type. Up to replacing Sec with Π in homotopy type theory, this corresponds to $\Gamma \vdash \Pi_\Delta B$ type. Because there dependent products always exist, by the same argument of Proposition 2.10.6.8 it is equivalent to $\Gamma \vdash \Pi_{\Delta^1} \dots \Pi_{\Delta n} B$ type. However, this relies on the existence of the intermediate dependent products. In our theory, the corresponding dependent mapping spaces will exist up to the Δ_i such that $\vdash \Gamma.\Delta_1 \dots \Delta_{i-1}$ anima. Still, although some of these may not exist, in our theory we are always able to borrow the iterated dependent product $\Gamma \vdash \Pi_{\Delta^1} \dots \Pi_{\Delta n} B$ type. Therefore, we see that there are dependent products our theory will not borrow, under the vests of dependent mapping spaces, but some (iterated) dependent product of it will always fit into our theory. To sum up, given a dependent type $\Gamma.\Delta_1 \dots \Delta_n \vdash B$ type, with $\vdash \Gamma$ anima, we can form a dependent product if we iterate enough of them, namely until we reach the anima Γ. What the theory lacks is the presence of all those intermediate dependent products over some of the Δ_i's. Even if we had the intermediate dependent mapping spaces, the terms would be the same as $\Gamma \vdash \mathrm{Sec}_\Delta B$ type up to homotopy bijection, and this is the reason why we are still able to construct a mapping space, an equivalence type, a contractibility type locally. However, what we miss is the type dependence: the lack of the intermediate dependent product does not allow us to lift $\Gamma \vdash \mathrm{Sec}_\Delta B$ type to a dependent type over $\Gamma.\Delta_1 \dots \Delta_i$ for an arbitrary i.

For instance, let us understand how to interpret $\mathrm{Map}_T(-,-)$ in homotopy type theory.

Example 2.10.6.10. Since the terminology of local mapping space is new, we question ourselves about the relation between the local mapping space $\Gamma \vdash \mathrm{Map}_\Delta(A,B)$ type and the ordinary mapping space $\Gamma.\Delta \vdash \mathrm{Map}(A,B)$ type is. This link is not always explicit in our theory, as we will be interested in cases where the latter does not exist. Therefore, it is better to understand this relation it in homotopy type theory, and see what the dependent mapping space construction correspond to, in a setting where we have all dependent products. By what just discussed, up to substituting Sec with Π, the local mapping space $\Gamma \vdash \mathrm{Map}_S(A,B)$ type corresponds to $\Gamma \vdash \Pi_{t:T, a:A(t)} B(t)$ type, i.e. to $\Gamma \vdash \Pi_{t:T} \Pi_{a:A(t)} B(t)$ type. In our setting, the latter is not well-formed unless T is an anima. This means that, when T is an anima, then $\mathrm{Map}_T(A,B) \simeq \mathrm{Sec}_T \mathrm{Map}(A,B)$. If T is not an anima, our theory is still able to borrow $\Gamma \vdash \Pi_{t:T}\{A(t) \to B(t)\,\mathrm{type}\}$ from homotopy type theory. However, this will not be the dependent mapping space over T of some type, i.e. it cannot be lifted to a type $\Gamma.T \vdash \mathrm{Map}(A,B)$ type with the same terms up to homotopy.

The fact that we are unable to construct a mapping space $\Gamma.S \vdash \mathrm{Map}(A,B)$ type, when S is not an anima, but instead we are able to construct $\Gamma \vdash \mathrm{Map}_S(A,B)$, which we could think as a dependent mapping space of it, is an instance of a recurrent phenomenon in this theory, which we will need to face multiple times. Let us see a further instance of this phenomenon.

Remark 2.10.6.11. Let $\vdash \Gamma$ anima with $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. In homotopy type theory, one can construct the contractibility type in every context, and not just over animae, meaning that there exists a type $\mathrm{isContr}(B)(a)$ parametrised over $\Gamma, a : A$. As an immediate consequence of compatibility with base change, for every $\Gamma \vdash a : A$ we have $\mathrm{isContr}(B)(a) \equiv \mathrm{isContr}(B(a))$ over Γ. In particular, if B is contractible over $\Gamma.A$, then so is $B(a)$ for every $a : A$. However, in our setting this idea is more subtle. Indeed, following Remark 2.10.6.9, the lack of some dependent mapping spaces forces us to quantify over a as well, so that the contractibility type $\Gamma \vdash \text{is-}\simeq\text{-}\mathrm{Contr}_A(B)$ lives over Γ only, and this would be interpret in homotopy type theory as $\Gamma \vdash \Pi_{a:A} \mathrm{isContr}(B)(a)$. This lives in the same context as $\Gamma \vdash \text{is-}\simeq\text{-}\mathrm{Contr}(B(a))$ for any $\Gamma \vdash a : A$. On the other hand, we still want to recover the implication through the explicit construction of a functor $\text{is-}\simeq\text{-}\mathrm{Contr}_A(B) \to \text{is-}\simeq\text{-}\mathrm{Contr}(B(a))$ over Γ. This comes from the fact that a retraction of $B \to \Delta^0_{\Gamma.A}$ gives, by base change along $a : A$, a retraction of $B(a) \to \Delta^0_\Gamma$, which is a term of $\Gamma \vdash \text{is-}\simeq\text{-}\mathrm{Contr}(B(a))$ type.

Compatibility with equivalence of contexts We now show that the dependent mapping space of a type $\Gamma.\Delta \vdash B$ type is invariant under equivalences of contexts $\Gamma.\Xi \simeq \Gamma.\Delta$ over Γ.

Lemma 2.10.6.12. Let $\vdash \Gamma$ anima and let $\vdash \Gamma.\Delta, \Delta'$ ctx be two equivalent contexts over Γ by means of $\gamma : \Gamma.\Delta' \to \Gamma.\Delta$. Then, for each $\Gamma.\Delta \vdash B$ type, we have an equivalence $\mathrm{Sec}_\Delta B \simeq \mathrm{Sec}_{\Delta'} B[\gamma]$ over Γ.

Proof. By Theorem 2.4.14.5, there is a homotopy bijection of terms between $\Gamma.\Delta \vdash B$ type and $\Gamma.\Delta' \vdash B[\gamma]$ type realised by the base change. By Lemma 2.10.6.4, we inherit a homotopy bijection between $\mathrm{Sec}_{\Delta} B$ and $\mathrm{Sec}_{\Delta'} B[\gamma]$ over Γ. We conclude by Meta Yoneda Lemma for animae thanks to the animation rule. □

Corollary 2.10.6.13. Let $\vdash \Gamma$ anima and let $\vdash \Gamma.\Delta, \Gamma : \Delta'$ ctx be two equivalent contexts over Γ by means of $\gamma : \Gamma.\Delta' \to \Gamma.\Delta$. Then, for each $\Gamma.\Delta \vdash A, B$ type, we have an equivalence of types $\mathrm{Map}_{\Delta'}(A[\gamma], B[\gamma]) \simeq \mathrm{Map}_{\Delta}(A, B)$.

Proof. By Corollary 2.4.14.6, we have that $\gamma.A : \Gamma.\Delta'.A[\gamma] \to \Gamma.\Delta.A$ is an equivalence of contexts, thus the claim follows form Lemma 2.10.6.12. □

As a consequence, whenever we have $\vdash \Gamma$ anima and $\Gamma.\Delta.A.B \vdash C, D$ type, we have an equivalence $\mathrm{Map}_{\Delta.A.B}(C, D) \simeq \mathrm{Map}_{\Delta.\Sigma_A B}(C, D)$, where C and D in the latter denote their appropriate base change.

Example 2.10.6.14. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type. Then, for every local functor $f : A \to B$ we have an equivalence of types $\mathrm{Map}_{\Delta.A.A[p]}(\mathrm{Id}_A, \mathrm{ev}_f^* \mathrm{Id}_B) \simeq \mathrm{Map}_{\Delta.A\times A}(\mathrm{Id}_A, \mathrm{ev}_f^* \mathrm{Id}_B)$, where in the latter we denote with Id_A and $\mathrm{ev}_f^* \mathrm{Id}_B$ respectively their base change. In particular, we can translate the functor ap_f from Example 2.10.4.8 into $\mathrm{ap}_f : \mathrm{Id}_A(\mathrm{pr}_1(s), \mathrm{pr}_2(s)) \to \mathrm{Id}_B(\mathrm{ev}_f(\mathrm{pr}_1(s)), \mathrm{ev}_f(\mathrm{pr}_2(s)))$ over $\Gamma.\Delta, s : A \times A$.

Higher equalities of functors Any equivalence of types carries with itself, by Proposition 2.4.11.7, an equivalence of the respective identity types. Interpreting this in the setting of function extensionality will tell us not only that we can check global equalities of functors pointwise, but also that we can prove equalities of global equalities of functors by considering the respective equalities of pointwise equalities. In other words, if we consider the functor $\mathrm{pwise}(f, f') : \mathrm{Id}_{\mathrm{Map}(A,B)}(f, f') \to f \simeq f'$, whenever two equalities in the source gets mapped into the same equality in $f \simeq f'$, then they coincide. This allows us to provide more easily equalities of global equalities of functors. As an intermediate step, we will also unwind the meaning of equality in $f \simeq f'$.

Proposition 2.10.6.15. Let $\vdash \Gamma$ anima with $\Gamma \vdash A, B$ type.

(1) We have an equivalence of types $\mathrm{Id}_{f\simeq f'}(\alpha, \beta) \simeq \mathrm{Sec}_{a:A} \mathrm{Id}_{\mathrm{Id}_B(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))}(\mathrm{ev}_\alpha(a), \mathrm{ev}_\beta(a))$ over the context $\Gamma, f : \mathrm{Map}(A, B), f' : \mathrm{Map}(A, B), \alpha : \mathrm{Id}_{\mathrm{Map}(A,B)}(f, f'), \beta : \mathrm{Id}_{\mathrm{Map}(A,B)}(f, f')$.

(2) The functor $\mathrm{ap}_{\mathrm{pwise}(f,f')} : \mathrm{Id}_{\mathrm{Id}_{\mathrm{Map}(A,B)}(f,f')^{\simeq}}(\alpha, \beta) \to \mathrm{Id}_{f\simeq f'}(\mathrm{ev}_{\mathrm{pwise}(f,f')}(\alpha), \mathrm{ev}_{\mathrm{pwise}(f,f')}(\beta))$ is a local equivalence over $\Gamma, f : \mathrm{Map}(A, B), f' : \mathrm{Map}(A, B), \alpha : \mathrm{Id}_{\mathrm{Map}(A,B)}(f, f'), \beta : \mathrm{Id}_{\mathrm{Map}(A,B)}(f, f')$, and induces a local equivalence $\mathrm{Id}_{\mathrm{Id}_{\mathrm{Map}(A,B)}(f,f')^{\simeq}}(\alpha, \beta) \simeq \mathrm{Id}_{f\simeq f'}(\mathrm{ap}_{\mathrm{ev}}(\overline{\alpha}), \mathrm{ap}_{\mathrm{ev}}(\overline{\beta}))$ where we denote with $\overline{\alpha}, \overline{\beta} : \mathrm{Id}_{\mathrm{Map}(A,B)}(f, f')$ the images under the canonical map from the groupoid core.

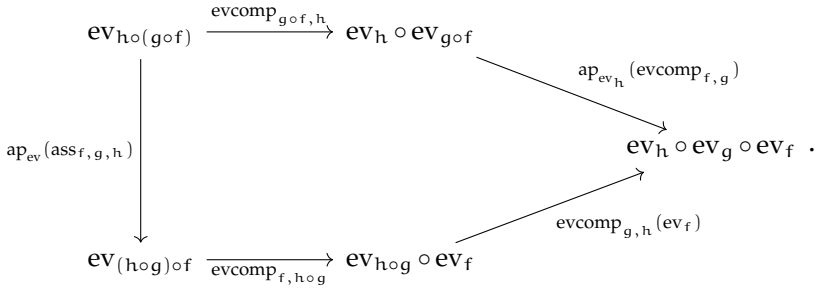
Proof. (1) is a direct consequence of function extensionality. For (2), the functor is an equivalence because $\mathrm{pwise}(f, f') : \mathrm{Id}_{\mathrm{Map}(A,B)}(f, f')^{\simeq} \to f \simeq f'$ over $\Gamma, f : \mathrm{Map}(A, B), f' : \mathrm{Map}(A, B)$ is, by using Proposition 2.4.11.7. By Construction 2.10.4.5, evaluating $\mathrm{pwise}(f, f')$ means to apply $\mathrm{ap}_{\mathrm{ev}}$ on the image under the inclusion of the groupoid core, and this yields the claim. □

This result can be interpreted as follows: in order to show an equality of global equalities of functors f and f', it suffices to show that the induced equalities of pointwise equalities from ev_f to $\mathrm{ev}_{f'}$ coincide. We are not over with the properties we wish to prove about equivalence types. Indeed, we will now introduce the notion of proposition and prove that the equivalence types is-$\simeq$-Equiv(f) and is-$=$-Equiv(f) are instances of it. This means that the data exhibiting f as a global, or pointwise, equivalences are contractible. Before that, we will explore further properties about functors, starting from their composition.

2.10.7 Composition of functors

As widely anticipated, we now move our attention to the properties of composition of functors. In particular, the reason why we want to regard functors, rather than functions of types, as functors of ∞-categories is that functors behave non-strictly. Function extensionality finally allows us to prove that composition of functors is associative only up to homotopy.

Proposition 2.10.7.1 (Composition of functors is non-strict)**.** Let $\vdash \Gamma$ anima and let $\Gamma \vdash A, B, C, D$ type. Then we have an *associativity term* $\Gamma, f : \mathrm{Map}(A,B), g : \mathrm{Map}(B,C), h : \mathrm{Map}(C,D) \vdash \mathrm{ass}_{f,g,h} : \mathrm{Id}_{\mathrm{Map}(A,D)}(h \circ (g \circ f), (h \circ g) \circ f)$. In particular, this term fits into the following commutative diagram of equalities over A



Proof. By function extensionality, it suffices to provide a homotopy between the evaluation functions. The main idea is that, since ev_f, ev_g and ev_h are functions, they compose strictly, so that we have a judgemental equality $\mathrm{ev}_h \circ (\mathrm{ev}_g \circ \mathrm{ev}_f) \equiv (\mathrm{ev}_h \circ \mathrm{ev}_g) \circ \mathrm{ev}_f$ of pointwise functions $A \to D$. Explicitly, following the diagram, we construct a homotopy $\Gamma, f : \mathrm{Map}(S,B), g : \mathrm{Map}(B,C), h : \mathrm{Map}(C,D), a : A \vdash (\mathrm{evcomp}_{f,h\circ g} \circ \mathrm{evcomp}_{g,h}(\mathrm{ev}_f))^{-1} \circ (\mathrm{ap}_{\mathrm{ev}_h}(\mathrm{evcomp}_{f,g}) \circ \mathrm{evcomp}_{g\circ f,h})(a) : \mathrm{Id}_D(\mathrm{ev}_{h\circ(g\circ f)}(a), \mathrm{ev}_{(h\circ g)\circ f}(a))$ by combining Lemma 2.10.4.15 and Lemma 2.4.3.1 and Remark 2.4.3.3. In particular, we can consider take the dependent mapping space over A, so that the inverse functor fixed by means of function extensionality, provides a homotopy $\Gamma, f : \mathrm{Map}(S,B), g : \mathrm{Map}(B,C), h : \mathrm{Map}(C,D) \vdash \overline{\mathrm{ass}}_{f,g,h} :\equiv \mathrm{extend}_{h\circ(g\circ f),(h\circ g)\circ f}(\lambda a.(\mathrm{evcomp}_{f,h\circ g} \circ \mathrm{evcomp}_{g,h}(\mathrm{ev}_f))^{-1} \circ (\mathrm{ap}_{\mathrm{ev}_h}(\mathrm{evcomp}_{f,g}) \circ \mathrm{evcomp}_{g\circ f,h})(a)) : \mathrm{Id}_{\mathrm{Map}(A,D)}(h \circ (g \circ f), (h \circ g) \circ f)^{\simeq}$. Including the groupoid core in the identity type yields $\mathrm{ass}_{f,g,h}$. Now, we study the evaluation of $\mathrm{pwise}(h \circ (g \circ f), (h \circ g) \circ f)$ at $\overline{\mathrm{ass}}_{f,g,h}$. This lies in $f \simeq f'$, and in particular has an evaluation function over $a : A$. We can compute this by Construction 2.10.4.5: $\mathrm{ev}_{\mathrm{ev}_{\mathrm{pwise}(h\circ(g\circ f),(h\circ g)\circ f)}(\overline{\mathrm{ass}}_{f,g,h})}(a)$ is homotopic to the application $\mathrm{ap}_{\mathrm{ev}}(\mathrm{ass}_{f,g,h})(a)$ for the explicit construction of pwise. On the other hand, by definition of $\mathrm{ass}_{f,g,h}$, it is also homotopic to the composite $(\mathrm{evcomp}_{f,h\circ g} \circ \mathrm{evcomp}_{g,h}(\mathrm{ev}_f))^{-1} \circ (\mathrm{ap}_{\mathrm{ev}_h}(\mathrm{evcomp}_{f,g}) \circ \mathrm{evcomp}_{g\circ f,h})(a) : \mathrm{Id}_D(\mathrm{ev}_{h\circ(g\circ f)}(a), \mathrm{ev}_{(h\circ g)\circ f}(a))$. This is because $\mathrm{extend}_{h\circ(g\circ f),(h\circ g)\circ f}$ is an inverse of $\mathrm{pwise}(h \circ (g \circ f), (h \circ g) \circ f)$, and ev preserves equalities via Lemma 2.4.3.1. We conclude the claim. □

We would like to point out two facts. First of all, the associativity equality $\mathrm{ev}_h \circ (\mathrm{ev}_g \circ \mathrm{ev}_f) \equiv (\mathrm{ev}_h \circ \mathrm{ev}_g) \circ \mathrm{ev}_f$ between the functions of types is judgemental, mirroring the strict calculus of substitutions, as functions of types fit into an ordinary category even though they have a notion of homotopy. However, even if it was propositional, there would be no way of constructing a functorial associativity term as we did. Indeed, we used mapping spaces to have functions appearing as variables in the context.

By Proposition 2.10.7.1, we finally traded the strictly-associative composition of functions of types for an associative composition of functors *up to homotopy*. This was exactly our main goal in the unstrictification procedure: to have a good notion of functors of ∞-categories that compose non-strictly. This reconnects to what we pointed out in Remark 2.3.2.1: we have an ordinary category Ctx which has a close structure to that of a tribe, and then we have some sort of localisation $\mathrm{Type} := \mathrm{L}(\mathrm{Ctx})$, which is a non-strict structure consisting of types and their functors, and the same for types over some context. We wish $\mathrm{Type}_{/\Gamma}$ to represent the theory of ∞-categories. Indeed, although it is not an ordinary category, we can still consider its objects and its arrows to be respectively types in context Γ and functors of types in context Γ. Our goal will be to determine rules so that the "objects" and the "arrows" of such non-strict structure deserve to be called ∞-categories and functors of ∞-categories respectively.

Remark 2.10.7.2. We may consider the homotopy category $\mathrm{Ho}(\mathrm{Type}_{/\Gamma})$ in the most obvious way: consisting of types as objects and of homotopy classes of functors as arrows. By function extensionality, this coincides with the ordinary category we previously defined under the same name, by considering types and homotopy classes of functions of types. This is an ordinary category, and contains a lot of information about the theory in the same way the homotopy category of Cat_∞ does for the theory of ∞-categories.

2.10.8 The pentagon non-axiom for composition of functors

Because functors of types do not have a strictly associative composition, then the problem of higher coherences arises. For example, we may wonder if the associativity homotopies satisfy a pentagon coherence, as we proved for composition of equalities. We claim that the theory already possesses this coherence, as we wish, and for this we will rely on Proposition 2.10.6.15, since we need to show an equality of global equalities of functors. In order to prove this result, though, we will need to go through some general technical properties of the function extensionality map first. We excuse with the reader in advance for the technical and heavy notation in this upcoming paragraph.

Remark 2.10.8.1 (λ preserves equalities)**.** Let $\vdash \Gamma$ anima with $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. Consider two functions $\Gamma.A \vdash f, f' : B[p]$ with a homotopy $\Gamma.A \vdash \alpha : \mathrm{Id}_{B[p]}(f, f')$. Let us consider the conjugate equality $\Gamma.A \vdash \overline{\alpha} :\equiv \mathrm{comp}^{\mathrm{Sec}}_{A;B}(f')^{-1} \circ (\alpha \circ \mathrm{comp}^{\mathrm{Sec}}_{A;B}(f)) : \mathrm{Id}_{B[p]}(\mathrm{ev}_{\lambda(f)}, \mathrm{ev}_{\lambda(f')})$ under the computation terms. Then, the inverse assignment that we fixed by function extensionality induces a homotopy $\mathrm{extend}_{\lambda x.f(x),\lambda x.f'(x)}(\lambda x.\overline{\alpha}(x)) : \mathrm{Id}_{\mathrm{Sec}_A\ B}(\lambda x.f(x), \lambda x.f'(x))^{\simeq}$. In particular, under the inclusion of the groupoid core, it provides $\overline{\mathrm{extend}_{\lambda x.f(x),\lambda x.f'(x)}}(\lambda x.\overline{\alpha}(x)) : \mathrm{Id}_{\mathrm{Sec}_A\ B}(\lambda x.f(x), \lambda x.f'(x))$.

Lemma 2.10.8.2. Let $\vdash \Gamma$ anima and let $\Gamma \vdash A, B$ type with two functions of types $\Gamma.A \vdash f, f' : B[p]$. Assume we have a homotopy $\Gamma.A \vdash \alpha : \mathrm{Id}_{B[p]}(f, f')$. Then, we have the following commutative square of equalities

$$\begin{array}{ccc} \mathrm{ev}_{\lambda x.f(x)} & \xrightarrow{\mathrm{ap}_{\mathrm{ev}}(\overline{\mathrm{extend}_{\lambda x.f(x),\lambda x.f'(x)}}(\lambda x.\overline{\alpha}(x)))} & \mathrm{ev}_{\lambda x.f'(x)} \\ {\scriptstyle \mathrm{comp}^{\mathrm{Sec}}_{A;B}(f)}\downarrow & & \downarrow{\scriptstyle \mathrm{comp}^{\mathrm{Sec}}_{A;B}(f')} \\ f & \xrightarrow[\alpha]{} & f' \end{array}$$

in $B[p]$ over $\Gamma.A$, in the notation of Remark 2.10.8.1.

Proof. We know, by Construction 2.10.4.5, that the upper equality is homotopic to the evaluation of the functor $\mathrm{pwise}(\lambda x.f(x), \lambda x, f'(x))$ at $\mathrm{extend}_{\lambda x.f(x),\lambda x,f'(x)}(\lambda x.\overline{\alpha}(x))$. Now, because $\mathrm{extend}_{\lambda x.f(x),\lambda x,f'(x)}$ is an inverse to $\mathrm{pwise}(\lambda x.f(x), \lambda x, f'(x))$, this evaluation is equal to $\lambda x.\overline{\alpha}(x) : \mathrm{Sec}_{x:A}(\mathrm{Id}_{B(x)}(\mathrm{ev}_{\lambda x.f(x)}, \mathrm{ev}_{\lambda x.f'(x)}))$. In particular, at a point $a : A$, its evaluation is homotopic to $\overline{\alpha}(a) \equiv \mathrm{comp}^{\mathrm{Sec}}_{A;B}(f')^{-1} \circ (\alpha \circ \mathrm{comp}^{\mathrm{Sec}}_{A;B}(f)) : \mathrm{Id}_{B(a)}(\mathrm{ev}_{\lambda x.f(x)}(a), \mathrm{ev}_{\lambda x.f'(x)}(a))$. Postcomposing with $\mathrm{comp}^{\mathrm{Sec}}_{A;B}(f')$, this gives the claim. □

In order to state and prove a pentagon axiom we also need to define the whiskering operator.

Construction 2.10.8.3 (Whiskering)**.** Let $\vdash \Gamma$ anima and let $\Gamma \vdash A, B, C$ type. Consider two functors $f, f' : A \to B$, and assume there exists a homotopy $\Gamma \vdash \alpha : \mathrm{Id}_{\mathrm{Map}(A,B)}(f, f')$. Then, we can form $\Gamma, g : \mathrm{Map}(B, C), a : A \vdash \mathrm{ap}_{\mathrm{ev}_g}(\mathrm{ap}_{\mathrm{ev}}(\alpha)(a)) : \mathrm{Id}_C(\mathrm{ev}_g(\mathrm{ev}_f(a)), \mathrm{ev}_g(\mathrm{ev}_{f'}(a)))$. Taking the conjugate as in Remark 2.10.8.1 along the equalities evcomp seen in Lemma 2.10.4.15, and taking the dependent mapping space over A, yields $\Gamma, g : \mathrm{Map}(B, C) \vdash \lambda a.\overline{\mathrm{ap}_{\mathrm{ev}_g}(\mathrm{ap}_{\mathrm{ev}}(\alpha)(a))} : g \circ f \simeq g \circ f'$. Now, the inverse we fixed of the functor $\mathrm{pwise}(\lambda x.f(x), \lambda x, f'(x))$, followed by the inclusion of the groupoid core, provides a homotopy of the form $\Gamma, g : \mathrm{Map}(B, C) \vdash g\alpha :\equiv \overline{\mathrm{extend}_{g\circ f, g\circ f'}}(\lambda a.\overline{\mathrm{ap}_{\mathrm{ev}_g}(\mathrm{ap}_{\mathrm{ev}}(\alpha)(a))}) : \mathrm{Id}_{\mathrm{Map}(A,C)}(g \circ f, g \circ f')$, which we call *whiskering of α with g on the left*. An analogous construction provides $\Gamma, h : \mathrm{Map}(C, A) \vdash \alpha h : \mathrm{Id}_{\mathrm{Map}(C,B)}(f \circ h, f' \circ h)$, which we call *whiskering of α with h on the right*. In particular, by Lemma 2.10.8.2 and by construction of the whiskering, for all $\Gamma \vdash g : \mathrm{Map}(B, C)$ we have a commutative square of equalities over $\Gamma.A$

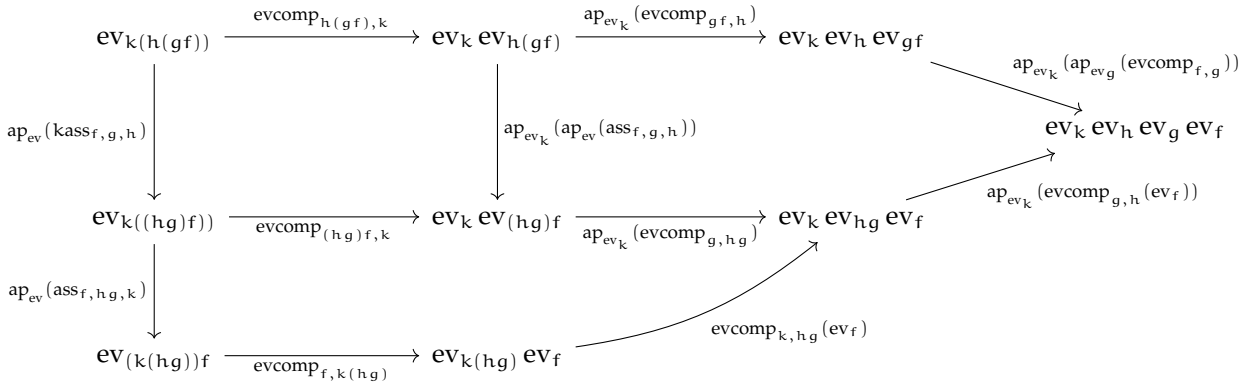
$$\begin{array}{ccc} \mathrm{ev}_{g\circ f} & \xrightarrow{\mathrm{ap}_{\mathrm{ev}}(g\alpha)} & \mathrm{ev}_{g\circ f'} \\ {\scriptstyle \mathrm{evcomp}_{f,g}}\downarrow & & \downarrow{\scriptstyle \mathrm{evcomp}_{f',g}} \\ \mathrm{ev}_g \circ \mathrm{ev}_f & \xrightarrow[\mathrm{ap}_{\mathrm{ev}_g}(\mathrm{ap}_{\mathrm{ev}}(\alpha))]{} & \mathrm{ev}_g \circ \mathrm{ev}_{f'} \end{array}.$$

This, and its variants, will have a key role in proving the pentagon coherence for functor composition.

Theorem 2.10.8.4 (Pentagon non-axiom for functors)**.** Let $\vdash \Gamma$ anima and let $\Gamma \vdash A, B, C, D, E$ type, and consider functors $A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D \xrightarrow{k} E$. Then, the following diagram of equalities commutes

$$\begin{array}{ccccc} k \circ (h \circ (g \circ f)) & \xrightarrow{\mathrm{ass}_{g\circ f,h,k}} & (k \circ h) \circ (g \circ f) & \xrightarrow{\mathrm{ass}_{f,g,k\circ h}} & ((k \circ h) \circ g) \circ f \\ {\scriptstyle k\,\mathrm{ass}_{f,g,h}}\downarrow & & & & \uparrow{\scriptstyle \mathrm{ass}_{g,h,k}\,f} \\ k \circ ((h \circ g) \circ f) & & \xrightarrow[\mathrm{ass}_{f,h\circ g,k}]{} & & (k \circ (h \circ g)) \circ f \end{array}.$$

Proof. The proof strategy will remind that of Pentagon non-axiom for equalities. By Proposition 2.10.6.15, it suffices to provide a proof that the diagram between the evaluations commutes. The strategy is to unwind the definition of the associativity and whiskering terms on the evaluations, using Construction 2.10.8.3. In particular, for any pair of two consecutive sides, we make them fit into a commutative diagram of equalities with vertex the pointwise function $\mathrm{ev}_k\,\mathrm{ev}_h\,\mathrm{ev}_g\,\mathrm{ev}_f$. Let us consider, for instance, the pair given by the left and the bottom sides: they fit into a commutative diagram



where the upper left square commutes by Construction 2.10.8.3, whereas the other two subdiagrams commute by construction of the associativity term seen in Proposition 2.10.7.1 and by application of functions preserving composition of equalities from Lemma 2.4.7.1. An analogous decomposition holds for all the other pairs of sides. Thus, we end up with a filling of the pentagon with commutative subdiagrams, as all sides are equated to $\mathrm{ev}_k\,\mathrm{ev}_h\,\mathrm{ev}_g\,\mathrm{ev}_f$. From this we conclude that the pentagon diagram commutes. □

2.10.9 Internal Yoneda Lemma

First of all, we now show that postcomposition is compatible with composition and identity, and consequently with equivalences. This will lead us to state and prove an internal version of the Yoneda lemma using mapping spaces.

Lemma 2.10.9.1 ($(-)_*$ is compatible with composition)**.** Let $\vdash \Gamma$ anima with $\Gamma \vdash A$ type and $\Gamma.A \vdash B, C, D$ type and consider two functors $f : B \to C$ and $g : C \to D$ over A. Then, we have $\Gamma \vdash \mathrm{Id}_{\mathrm{Map}(\mathrm{Sec}_A B, \mathrm{Sec}_A D)}((g \circ f)_*, g_* \circ f_*)$.

Proof. By function extensionality, it suffices to show this pointwise on their evaluation functions. Our goal is then to show that $\mathrm{ev}_{(g\circ f)_*}$ is homotopic to $\mathrm{ev}_{g_*\circ f_*}$ pointwise at any $b : \mathrm{Sec}_{x:A} B(x)$ inside $\mathrm{Sec}_{x:A} D(x)$. An iteration of function extensionality reduces this to showing that pointwise at any $a : A$ we have $\mathrm{ev}_{\mathrm{ev}_{(g\circ f)_*}(a)}$ homotopic to $\mathrm{ev}_{\mathrm{ev}_{g_*\circ f_*}(a)}$ inside $D(a)$. The strategy is to show that both of them are homotopic, for any $a : A$, to $\mathrm{ev}_g(\mathrm{ev}_f(\mathrm{ev}_b(a)))$ in $D(a)$. The tools we will be using are Remark 2.10.2.4, Lemma 2.10.4.15 and Construction 2.10.4.12.

(1) For the first, by Construction 2.10.4.12 followed by Lemma 2.10.4.15 we have $\Gamma, b : \mathrm{Sec}_{x:A} B(x), a : A \vdash \mathrm{evcomp}_{f,g}(\mathrm{ev}_b(a)) \circ \mathrm{evpost}_{g\circ f,b}(a) : \mathrm{Id}_{D(a)}(\mathrm{ev}_{\mathrm{ev}_{(g\circ f)_*}(b)}(a), \mathrm{ev}_g(\mathrm{ev}_f(\mathrm{ev}_b(a))))$.

(2) For the second, Lemma 2.10.4.15 and Remark 2.10.2.4, namely evaluation preserving equality, yield $\Gamma, b : \mathrm{Sec}_{x:A} B(x), a : A \vdash \mathrm{ap}_{\mathrm{ev}_q}(\mathrm{evcomp}_{f_*,g_*}(b))(a) : \mathrm{Id}_{D(a)}(\mathrm{ev}_{\mathrm{ev}_{g_*\circ f_*}(b)}(a), \mathrm{ev}_{(\mathrm{ev}_g)_*((\mathrm{ev}_f)_*(b))}(a))$.

Now, by Construction 2.10.4.12, we can write the latter $\Gamma, b : \mathrm{Sec}_{x:A} B(x), a : A \vdash \mathrm{evpost}_{g,(\mathrm{ev}_f)_*(b)}(a) : \mathrm{Id}_{D(a)}(\mathrm{ev}_{\mathrm{ev}_{g_*}(\mathrm{ev}_{f_*}(b))}(a), \mathrm{ev}_g(\mathrm{ev}_{\mathrm{ev}_{f_*}(b)}(a)))$. The same result, combined with ev_g preserving equalities by Lemma 2.4.3.1, yields a homotopy of the form $\Gamma, b : \mathrm{Sec}_{x:A} B(x), a : A \vdash \mathrm{ap}_{\mathrm{ev}_g}(\mathrm{evpost}_{\mathrm{ev}_f,b}) : \mathrm{Id}_{D(a)}(\mathrm{ev}_g(\mathrm{ev}_{\mathrm{ev}_{f_*}(b)}(a)), \mathrm{ev}_g(\mathrm{ev}_f(\mathrm{ev}_b(a))))$. Overall, composing all the above produces a homotopy $\Gamma, b : \mathrm{Sec}_{x:A} B(x), a : A \vdash \mathrm{ap}_{\mathrm{ev}_g}(\mathrm{evpost}_{\mathrm{ev}_f,b}) \circ (\mathrm{evpost}_{\mathrm{ev}_g,(\mathrm{ev}_f)_*(b)}(a) \circ \mathrm{ap}_{\mathrm{ev}}(\mathrm{evcomp}_{f_*,g_*}(b))(a)) : \mathrm{Id}_{D(a)}(\mathrm{ev}_{\mathrm{ev}_{g_*\circ f_*}(b)}(a), \mathrm{ev}_g(\mathrm{ev}_f(\mathrm{ev}_b(a))))$ as claimed.

Now, by (1) and (2), we obtain a homotopy from $\mathrm{ev}_{\mathrm{ev}_{(g\circ f)_*}(b)}(a)$ to $\mathrm{ev}_{\mathrm{ev}_{g_*\circ f_*}(b)}(a)$ over $\Gamma, b : \mathrm{Sec}_{x:A} B(x), a : A$. Function extensionality provides an equality $\Gamma, b : \mathrm{Sec}_A B \vdash \mathrm{Id}_{\mathrm{Sec}_A D}(\mathrm{ev}_{g_*\circ f_*}(b), \mathrm{ev}_{(g\circ f)_*}(b))$ type, and again function extensionality yields an inhabitant of $\Gamma \vdash \mathrm{Id}_{\mathrm{Map}(\mathrm{Sec}_A B, \mathrm{Sec}_A D)}(g_* \circ f_*, (g \circ f)_*)$ type. □

Lemma 2.10.9.2 ($(-)_*$ is compatible with identity)**.** Let $\vdash \Gamma$ anima with $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. Then, we have an inhabitant of $\Gamma \vdash \mathrm{Id}_{\mathrm{Map}(\mathrm{Sec}_A B, \mathrm{Sec}_A B)}((1_B)_*, 1_{\mathrm{Sec}_A B})$ type.

Proof. By function extensionality it suffices to show their evaluation functions are pointwise equal. We have a homotopy $\Gamma, f : \mathrm{Sec}_{x:A} B(x), a : A \vdash (\mathrm{comp}^{\mathrm{Sec}}_{A,B}(f) \circ \mathrm{evpost}_{1_B,f})(a) : \mathrm{Id}_{\mathrm{Sec}_A B}(\mathrm{ev}_{\mathrm{ev}_{(1_B)_*}(f)}(a), \mathrm{ev}_{1_B} f(a))$, which by function extensionality provides a homotopy between $\mathrm{ev}_{(1_B)_*}(f)$ and f in $\mathrm{Sec}_{x:A} B(x)$. Moreover, we have $\Gamma, f : \mathrm{Sec}_{x:A} B(x) \vdash \mathrm{comp}^{\mathrm{Sec}}_{\mathrm{Sec}_A B, (\mathrm{Sec}_A B)[p]}(q)(f) : \mathrm{comp}_{\mathrm{Sec}_A B, \mathrm{Sec}_A B} \mathrm{Id}_{\mathrm{Sec}_A B}(q)(f)(\mathrm{ev}_{1_{\mathrm{Sec}_A B}}(f), f)$. The combination of the two homotopies yields the claim. □

As a consequence of the previous results, we can now prove another fundamental property for the Sec_A operator that we were missing a priori, that is, its compatibility with equivalences. In other words, the postcomposition functor, as in Construction 2.10.4.12, along an equivalence is an equivalence.

Proposition 2.10.9.3. Let $\vdash \Gamma$ anima and let $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash B, C$ type with a local equivalence $f : B \xrightarrow{\sim} C$ over Δ. Then, the postcomposition functor $f_* : \mathrm{Sec}_\Delta B \to \mathrm{Sec}_\Delta C$ is an equivalence.

Proof. For simplicity in the notation, we will assume $\Gamma \vdash \Delta \equiv A$ type. Let us consider a left inverse $g : C \to B$ over A, so that $g \circ f$ is homotopic to 1_B in $\mathrm{Map}_A(B, B)$. By Remark 2.10.4.14, the map $\mathrm{ap}_{(-)_*}(\beta) : \mathrm{Id}_{\mathrm{Map}_A(B,B)}(h, h') \to \mathrm{Id}_{\mathrm{Map}(\mathrm{Sec}_A B, \mathrm{Sec}_A B)}(h_*(k), h'_*(k))$ over the context $\Gamma, h : \mathrm{Map}_A(B, B), h' : \mathrm{Map}_A(B, B), k : \mathrm{Sec}_{x:A} B(x)$ maps the homotopy we have into a homotopy from $(g \circ f)_*$ to $1_{\mathrm{Sec}_A B}$ by Lemma 2.10.9.2. By Lemma 2.10.9.1, we know that $(g \circ f)_*$ is homotopic to $g_* \circ f_*$, hence the claim. □

This specialises to the case of trivial type dependence, where B and C are of the form $B'[p]$ and $C'[p]$ for some $\Gamma \vdash B, C$ type. In this non-dependent case, this result says that if $f : B \to C$ is an equivalence, then $f_* : \mathrm{Map}(A, B) \to \mathrm{Map}(A, C)$ is an equivalence. In this special situation, we will can show a similar property for the precomposition functor. This is an instance of a very important result, which we will now see, that is, an internal version of the Yoneda Lemma. Since we have access to mapping spaces, which shall be a good replacement of Hom-sets of ordinary categorical semantics, we expect to recover an internal version of the Yoneda Lemma that relies on them, as it is one of the reasons they were introduced in the first place. This internal Yoneda Lemma will have a key role in the theory as much as Yoneda Lemma has in ordinary category theory. This might recall Meta Yoneda Lemma to the reader, but instead holds on a different level, as it will be internal in the theory rather than meta-theoretical.

Theorem 2.10.9.4 (Internal Yoneda Lemma)**.** Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx, and let $\Gamma.\Delta \vdash A, B$ type with a local functor $f : A \to B$. The following conditions are equivalent:

(1) $f : A \to B$ is an equivalence.

(2) For every $\Gamma \vdash X$ type and $f_* : \mathrm{Map}_\Delta(X, A) \to \mathrm{Map}_\Delta(X, B)$ is an equivalence.

(3) $f_* : \mathrm{Map}_\Delta(A, A) \to \mathrm{Map}_\Delta(A, B)$ and $f_* : \mathrm{Map}_\Delta(B, A) \to \mathrm{Map}_\Delta(B, B)$ are equivalences.

(4) For every $\Gamma \vdash X$ type and $f^* : \mathrm{Map}_\Delta(B, X) \to \mathrm{Map}_\Delta(A, X)$ is an equivalence.

(5) $f^* : \mathrm{Map}_\Delta(B, A) \to \mathrm{Map}_\Delta(A, A)$ and $f^* : \mathrm{Map}_\Delta(B, B) \to \mathrm{Map}_\Delta(A, B)$ are equivalences.

Proof. (1) $\implies$ (2) by Proposition 2.10.9.3, and an analogous argument proves the same for precomposition, so that (1) $\implies$ (4) as well. Because (2) and (4) are obviously stronger than (3) and (5) respectively, we conclude that (1) implies all the other conditions.

We are left to show that (3) and (5) imply (1). Assume that (3) holds, and consider $g : B \to A$ such that $f_* : g \mapsto 1_B$. In particular, by construction of f_*, this means that g is a right inverse of f, and now we need to show it is in fact a left inverse as well. Consider the composite $g \circ f : A \to A$, and note that its image $f_*(g \circ f) : A \to B$ is equal to f by associativity of composition seen in Proposition 2.10.7.1. Since f_* is an equivalence, this implies that $g \circ f$ is homotopic to 1_A as well. An analogous argument follows for (5). □

Internal Yoneda Lemma is particularly useful because we are able to show equivalences of mapping spaces naively, thanks to the animation rule, and thus the fact that they are "collections" as in Remark 2.9.2.3. This means that we can show that two types are equivalent roughly by showing that homotopy classes functors into them are the same. In particular, we showed weaker versions of this as we saw that (3) and (5) are in fact equivalent to (2) and (4) respectively. Indeed, we saw that instead of checking f_* and f^* being equivalences with arbitrary types X, it suffices to consider the universal cases of $X \equiv A, B$.

Corollary 2.10.9.5. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B, C, D$ type. There is a functor of types $\text{is-}\simeq\text{-Equiv}_\Delta(f) \to \text{is-}\simeq\text{-Equiv}(f_*) \times \text{is-}\simeq\text{-Equiv}(f^*)$ over $\Gamma, f : \text{Map}_\Delta(A, B)$, where $f_* : \text{Map}_\Delta(C, A) \to \text{Map}_\Delta(C, B)$ and $f^* : \text{Map}_\Delta(B, D) \to \text{Map}_\Delta(A, D)$.

Proof. Follows by (1) implying (2) and (4) in Internal Yoneda Lemma. □

2.10.10 Propositions

We now introduce the notion of propositions. We pointed out the interpretation of types as statements, though we saw that in general they even admit more than one non-equal proofs. This happens with the identity type as well. In particular, it is a fundamental idea in type theory that we do not care about the existence of an inhabitant of a type, instead we always need to provide one, and this choice matters. Now, we focus on those types for which this choice does not really matter, and the only relevant information is whether they are inhabited or not. In other words, they are types that define a property rather than data that must be given. These are the so-called *propositions*, and one of the goals of this paragraph will be to show that the equivalence type is a proposition.

Definition 2.10.10.1 (Proposition)**.** Let $\vdash \Gamma$ anima, and let $\vdash \Gamma.\Delta$ ctx with $\Gamma.\Delta \vdash A$ type. We define the type $\Gamma \vdash \text{isProp}_\Delta(A) :\equiv \text{Map}(\text{Sec}_\Delta A, \text{is-}\simeq\text{-Contr}_\Delta A)$. We say that A is a *proposition* if this type is inhabited.

In particular, if $\Delta \equiv \emptyset$, we have $\Gamma \vdash \text{isProp}(A) \equiv \text{Map}(A^\simeq, \text{is-}\simeq\text{-Contr}(A)$ type.

Let us reflect on the definition. Being a proposition means that there is a functor $\text{Sec}_\Delta A \to \text{is}_\Delta(A)$, which means that if $\Gamma.\Delta \vdash A$ type is inhabited, then so is $\Gamma \vdash \text{is-}\simeq\text{-Contr}_\Delta(A)$ type, i.e. A is contractible. This resembles our motivation: the cases we are interested in are when A is empty (false proposition) or when A is inhabited (true proposition), as in the second case the inhabitant is essentially unique. Therefore, being a proposition means being a statement with a contractible choice of proofs.

Now, we want to show that those conditions we introduced that we want to regard as "properties" (of types, terms, functors) are indeed equivalences. The goal of this paragraph is now to show that the type $\text{is-}\simeq\text{-Equiv}(f)$ is a proposition, so that we can conclude that being an equivalence is in fact a property of a functor, rather than actual data to provide that are not uniquely determined. Consequently, by Proposition 2.10.6.7, also the type $\text{is-}=\text{-Equiv}(f)$ will be a proposition. A first step is to note that the equivalence type is logically equivalent to the type of having contractible fibres.

Remark 2.10.10.2. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and let $\Gamma.\Delta \vdash A, B$ type. We can construct $\Gamma.\Delta, f : \text{Map}_\Delta(A, B), b : B \vdash \text{fib}_f(b) :\equiv \text{fib}_{\text{ev}_f}(b)$ and inherit all the properties of the fibre of a function.

Proposition 2.10.10.3. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type. For any local functor $f : A \to B$ the types $\text{is-}\simeq\text{-Equiv}_\Delta(f)$ and $\text{is-}\simeq\text{-Contr}_{\Delta.B}(\text{fib}_f)$ are logically equivalent over Γ.

Proof. Follows from Proposition 2.6.10.1. □

Lemma 2.10.10.4. Let $\vdash \Gamma$ anima and let $\vdash \Gamma.\Delta$ ctx.

(1) The type constructor Sec_Δ is compatible with products of types over $\Gamma.\Delta$.

(2) The type constructor is-$\simeq$-Contr is compatible with products of types over Γ.

Proof. Let $\Gamma.\Delta \vdash A, B$ type For the sake of simplicity, we will assume $\Gamma \vdash \Delta \equiv T$ type. For (1), we need to show $\mathrm{Sec}_T(A \times B) \simeq \mathrm{Sec}_T A \times \mathrm{Sec}_T B$. In one direction, we have the functor $\mathrm{Sec}_T(A \times B) \to \mathrm{Sec}_T A \times \mathrm{Sec}_T B$ assigning to each $h : \mathrm{Sec}_{t:T}(A(t) \times B(t))$ the term $\mathrm{pair}(\lambda t.\, \mathrm{pr}_1(\mathrm{ev}_h(t)), \lambda t.\, \mathrm{pr}_1(\mathrm{ev}_h(t)))$, in the other direction we use the induction principle of the product and consider $\Gamma, f : \mathrm{Sec}_{t:T} A(t), g : \mathrm{Sec}_{t:T} B(t) \vdash \lambda t.\mathrm{pair}(\mathrm{ev}_f(t), \mathrm{ev}_g(t)) : \mathrm{Sec}_{t:T}(A(t) \times B(t))$. Both composites are homotopic to the identical assignment by function extensionality and Proposition 2.6.2.1. For (2), note that for any $\Gamma \vdash X$ type, we have is-$\simeq$-Contr$(X) \simeq \Sigma_{x:\mathrm{Map}(\Delta^0_\Gamma, X)} \mathrm{Sec}_{x':X} \mathrm{Id}_{\mathrm{Map}(X,X)}(\mathrm{ev}_y(!_A(x')), x) \simeq \Sigma_{x':X} \mathrm{Sec}_{x:X} \mathrm{Id}_X(x', x)$ by Lemma 2.10.6.12 and Proposition 2.10.9.3. By (1) and Proposition 2.6.3.3, when $X \equiv S \times T$, we can rewrite $\Sigma_{x:S\times T} \mathrm{Sec}_{x':S\times T} \mathrm{Id}(x', x)$ as $(\Sigma_{s':S} \mathrm{Sec}_{s:S} \mathrm{Id}_S(s', s)) \times (\Sigma_{t':T} \mathrm{Sec}_{t:T} \mathrm{Id}_T(t', t)) \simeq$ is-$\simeq$-Contr$(S) \times$ is-$\simeq$-Contr(T). □

Proposition 2.10.10.5. Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type and let $f : A \to B$ be a functor. Then, the type is-$\simeq$-Equiv$_\Delta(f)$, and thus is-$=$-Equiv$_\Delta(f)$, is a proposition.

Proof. In order to show that is-$\simeq$-Equiv$_\Delta(f)$ is a proposition, we construct a functor is-$\simeq$-Equiv$_\Delta(f) \to$ is-$\simeq$-Contr(is-$\simeq$-Equiv$_\Delta(f)$) over Γ. By Corollary 2.10.9.5, we get a functor

$$\text{is-}\simeq\text{-Equiv}_\Delta(f) \to \text{is-}\simeq\text{-Equiv}(f_*) \times \text{is-}\simeq\text{-Equiv}(f^*),$$

where $f_* : \mathrm{Map}_\Delta(B, A) \to \mathrm{Map}_\Delta(B, B)$ and $f^* : \mathrm{Map}_\Delta(B, A) \to \mathrm{Map}_\Delta(A, A)$. We can postcompose this with the following functor from Proposition 2.10.10.3

$$\text{is-}\simeq\text{-Equiv}(f_*) \times \text{is-}\simeq\text{-Equiv}(f^*) \to \text{is-}\simeq\text{-Contr}_{\mathrm{Map}_\Delta(B,B)}(\mathrm{fib}_{f_*}) \times \text{is-}\simeq\text{-Contr}_{\mathrm{Map}_\Delta(A,A)}(\mathrm{fib}_{f_*}).$$

In particular, by Remark 2.10.6.11, we get functors to the contractibility types of the fibres over the identities:

$$\text{is-}\simeq\text{-Contr}_{\mathrm{Map}_\Delta(B,B)}(\mathrm{fib}_{f_*}) \times \text{is-}\simeq\text{-Contr}_{\mathrm{Map}_\Delta(A,A)}(\mathrm{fib}_{f_*}) \to \text{is-}\simeq\text{-Contr}(\mathrm{fib}_{f_*}(1_B)) \times \text{is-}\simeq\text{-Contr}(\mathrm{fib}_{f^*}(1_A)).$$

The trick is now to rewrite the two target types in a friendlier way. Indeed, observe that the fibre of $\Gamma, f : \mathrm{Map}_\Delta(A, B), g : \mathrm{Map}_\Delta(A, A) \vdash \mathrm{fib}_{f^*}(g)$ at 1_A is equivalent to has-$=$-Retr$_\Delta(f)$. This is because $\Gamma, f : \mathrm{Map}_\Delta(A, B) \vdash \mathrm{fib}_{f^*}(1_A) \equiv \Sigma_{g:\mathrm{Map}_\Delta(B,A)} \mathrm{Id}_{\mathrm{Map}_\Delta(A,A)}(\mathrm{ev}_{f^*}(g), 1_A) \simeq \Sigma_{g:\mathrm{Map}_\Delta(B,A)} \mathrm{Id}_{\mathrm{Map}_\Delta(A,A)}(g \circ f, 1_A)$ by construction of f^* and Proposition 2.6.3.2. On the other hand, we have $\mathrm{fib}_{f_*}(1_B) \simeq$ has-$=$-Sect$_\Delta(f)$. In particular, by Proposition 2.10.6.7, we can replace the "$=$" with a "$\simeq$", to finally get a composite functor

$$\text{is-}\simeq\text{-Equiv}_\Delta(f) \to \text{is-}\simeq\text{-Contr}(\text{has-}\simeq\text{-Retr}_\Delta(f)) \times \text{is-}\simeq\text{-Contr}(\text{has-}\simeq\text{-Retr}_\Delta(f)).$$

Finally, by Lemma 2.10.10.4, we can rewrite the target up to equivalence as is-$\simeq$-Contr(has-$\simeq$-Retr$_\Delta(f)) \times$ is-$\simeq$-Contr(has-$\simeq$-Retr$_\Delta(f)) \simeq$ is-$\simeq$-Contr(is-$\simeq$-Equiv$_\Delta(f)$). This determines a functor is-$\simeq$-Equiv$_\Delta(f) \to$ is-$\simeq$-Contr(is-$\simeq$-Equiv$_\Delta(f)$), hence the claim. □

As a consequence of Proposition 2.10.10.5, given $\vdash \Gamma$ anima with $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash A, B$ type, if a functor $f : A \to B$ is an equivalence (global or pointwise), then there is a contractible choice of retraction and section together with the respective homotopies. In particular, the fixed choice of inverse pwise(f, f') did not matter and the whole discussion is independent on that.

2.10.11 Dependent mapping space as $\forall$ quantifier

As we did for the dependent sum, we are going to interpret the dependent mapping space as a $\forall$ quantifier. In this case, being Sec introduced as an anima, we will not necessarily need to sit in the setting of homotopy type theory for the following discussion, as long as we assume of dealing with animae. Indeed, although Sec alters many of the features of Π from homotopy type theory, it preserves its interpretation as a quantifier for the formation of statements. However, in order to interpret the input data as statements themselves, it will still be a wiser choice to assume being in homotopy type theory. More explicitly, given $\vdash \Gamma$ anima and $\Gamma \vdash A$ type, we can look at a predicate over A given by a dependent type $\Gamma, a : A \vdash B(a)$ type. As we remarked, $B(a)$ is a parametrised statement over A, and an inhabitant of $B(a)$ would say that $B(a)$

is true. Taking the dependent mapping space $\Gamma \vdash \mathrm{Sec}_{a:A}\, B(a)$ type we get a statement whose proofs are the datum of a conditional proof of $\Gamma, a : A \vdash B(a)$ type, that is, for every $a : A$ a proof that $B(a)$ holds. In other words, we can interpret $\Gamma \vdash \mathrm{Sec}_{a:A}\, B(a)$ type as the statement "for all $a : A$, $B(a)$ holds". This means that the introduction of dependent mapping spaces into the present setting formalises the idea of the $\forall$ quantifier in the logic of the theory. In particular, when the type dependence is trivial, i.e. $\Gamma \vdash C$ type, a term of $\Gamma \vdash \mathrm{Sec}_{a:A}\, C \equiv \mathrm{Map}(A, C)$ type is the statement whose proofs are terms of $\Gamma, a : A \vdash C$ type, that are proofs that A implies C. In other words, $\Gamma \vdash \mathrm{Sec}_{a:A}\, C \equiv \mathrm{Map}(A, C)$ type is the statement "A implies C". Indeed, the terms are in correspondence with functions between statement which we regarded as implications between statement.

It is not surprising, then, the way we formed various of the type constructors we encountered. Let us review some previous parts of the chapter under this light:

Remark 2.10.11.1. Let $\vdash \Gamma$ anima and let $\Gamma \vdash A, B$ anima.

(1) $\Gamma, f : \mathrm{Map}(A, B), f' : \mathrm{Map}(A, B) \vdash f \simeq f' \equiv \mathrm{Sec}_{a:A}\, \mathrm{Id}_B(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))$ type is the statement "for all $a : A(t)$, $f(a)$ is equal to $f'(a)$ in B"

(2) $\Gamma, f : \mathrm{Map}(A, B) \vdash \text{has-}\simeq\text{-Retr}(f) \equiv \Sigma_{g:\mathrm{Map}(B,A)}\, g \circ f \simeq 1_A$ is the statement ''there exists a $g : \mathrm{Map}(B, A)$ such that for all $a : A$, $\mathrm{ev}_{g\circ f}(a))$ is equal to a in B".

Let us now consider $\Gamma \vdash T$ type with $\vdash \Gamma.T$ anima and $\Gamma, t : T \vdash C(t)$ type.

(1) Consider $\Gamma, t : T \vdash D(t)$ type. Proposition 2.10.6.8 means that the statement "for all $t : T$, for all $c : C(t)$, $D(t)$ holds" is equivalent to "for all $t : T$, the statement '$C(t)$ implies $D(t)$' holds".

(2) Consider $\Gamma, t : T, c : C(t) \vdash D(t, c)$ type. Proposition 2.10.6.8 means that the statement "for all $t : T$ and for all $c : C(t)$, $D(t, c)$ holds" is equivalent to "for all $t : T$, the statement 'for all $c : C(t)$, $D(t, c)$' holds". These are properties we expect from a $\forall$ quantifier.

2.10.12 The non-axiom of choice

Under the homotopy type theoretical interpretation of types as statements, where Sec is regarded as $\forall$ and Σ is closely related to $\exists$, we can state and prove a result that could remind of the set-theoretic axiom of choice. This is true only at a first sight, because the correct axiom of choice should be formulated with the correct version of the $\exists$ quantifier, and this can be either imposed or not as an axiom of the theory. However, the result is still of practical relevance as it give some sort of commutativity rule between Sec and Σ.

Theorem 2.10.12.1 (Non-axiom of choice)**.** Let $\vdash \Gamma$ anima with $\Gamma \vdash X$ type, $\Gamma.X \vdash A$ type and $\Gamma.X.A \vdash P$ type. We have an equivalence of types $\mathrm{Sec}_{x:X}\, \Sigma_{a:A(x)} P(x, a) \simeq \Sigma_{g:\mathrm{Sec}_{x:X}\, A(x)}\, \mathrm{Sec}_{x:X}\, P(x, \mathrm{ev}_g(x))$.

Proof. First of all, we construct the two assignments: in one direction we have $\Gamma, f : \mathrm{Sec}_{x:X}\, \Sigma_{a:A(x)} P(x, a) \vdash \mathrm{pair}(\lambda x.\, \mathrm{pr}_1(\mathrm{ev}_f(x)), \lambda x.\, \mathrm{pr}_2(\mathrm{ev}_f(x))) : \Sigma_{g:\mathrm{Sec}_{x:X}\, A(x)}\, \mathrm{Sec}_{x:X}\, P(x, \mathrm{ev}_g(x))$ and in the other one we have $\Gamma, \gamma : \Sigma_{g:\mathrm{Sec}_{x:X}\, A(x)}\, \mathrm{Sec}_{x:X}\, P(x, \mathrm{ev}_g(x)) \vdash \lambda x.\mathrm{pair}(\mathrm{ev}_{\mathrm{pr}_1(\gamma)}(x), \mathrm{ev}_{\mathrm{pr}_2(\gamma)}(x)) : \mathrm{Sec}_{x:X}\, \Sigma_{a:A(x)} P(x, a)$. The postcomposition of the former with the latter gives rise to the following assignment $\Gamma, f : \mathrm{Sec}_{x:X}\, \Sigma_{a:A(x)} P(x, a) \vdash \lambda x.\mathrm{pair}(\mathrm{ev}_{\mathrm{pr}_1(\mathrm{pair}(\lambda x.\, \mathrm{pr}_1(\mathrm{ev}_f(x)), \mathrm{pr}_2(\mathrm{ev}_f(x))))}(x), \mathrm{ev}_{\mathrm{pr}_2(\mathrm{pair}(\lambda x.\, \mathrm{pr}_1(\mathrm{ev}_f(x)), \mathrm{pr}_2(\mathrm{ev}_f(x))))}(x)) : \mathrm{Sec}_{x:X}\, \Sigma_{a:A(x)} P(x, a)$. This can be easily equated to the identical assignment, as it consists of pairs of terms that cancel themselves thanks to the equalities. The composite in the reversed order assigns to $\Gamma, \gamma : \Sigma_{g:\mathrm{Sec}_{x:X}\, A(x)}\, \mathrm{Sec}_{x:X}\, P(x, \mathrm{ev}_g(x))$ the following term: $\mathrm{pair}(\lambda x.\, \mathrm{pr}_1(\mathrm{ev}_{\lambda x.\mathrm{pair}(\mathrm{ev}_{\mathrm{pr}_1(\gamma)}(x), \mathrm{ev}_{\mathrm{pr}_2(\gamma)}(x))}(x)), \lambda x.\, \mathrm{pr}_2(\mathrm{ev}_{\lambda x.\mathrm{pair}(\mathrm{ev}_{\mathrm{pr}_1(\gamma)}(x), \mathrm{ev}_{\mathrm{pr}_2(\gamma)}(x))}(x)))$: $\Sigma_{g:\mathrm{Sec}_{x:X}\, A(x)}\, \mathrm{Sec}_{x:X}\, P(x, \mathrm{ev}_g(x))$. Analogously, this is clearly the identity consisting of iterations of pairs of terms that simplify eachother. □

Remark 2.10.12.2. The name of Non-axiom of choice is quite peculiar, thus let us give an interpretation of it. For the sake of simplicity let us consider the case $\Gamma.X \vdash A \equiv B[p]$ type for some $\Gamma \vdash B$ type, which comes equipped with a functor $f : B \to X$, and $\Gamma, x : X, b : B \vdash P \equiv \mathrm{Id}_B(x, \mathrm{ev}_f(b))$ type. The non-axiom of choice than gives an equivalence $\mathrm{Sec}_{x:X}\, \Sigma_{b:B}\, \mathrm{Id}_B(x, \mathrm{ev}_f(b)) \simeq \Sigma_{g:\mathrm{Map}(X,B)}\, \mathrm{Sec}_{x:X}\, \mathrm{Id}_B(x, \mathrm{ev}_f(\mathrm{ev}_g(x))$. It is clear that the type on the right-hand side has witnesses given by sections of f. However, because Σ is not exactly the $\exists$ quantifier, the left-hand side is not exactly the statement "f is surjective", as a witness would be an explicit exhibition, for any $b : B$, of a $x : X$ such that $\mathrm{ev}_f(x)$ is equal to b. In other words, more than the property of f being surjective, it consists itself of a section of f given pointwise. The point of the axiom of choice is really

the ability to choose witnesses in each preimage. In conclusion, although Theorem 2.10.12.1 can be useful for computations, its logical content does not add anything to the theory, as it simply states an equivalence between two ways of saying that a functor has a section. This is coherent with our wishes, since the axiom of choice should be an axiom which we may or may not impose to hold in the theory.

2.10.13 Local mapping space as base change

We now want to show that, given $\vdash \Gamma$ anima with $\Gamma \vdash S$ type and $\Gamma.S \vdash A, B$ type, we can obtain the local mapping space $\mathrm{Map}_S(A,B)$ as a base change of a type $\Gamma.\mathrm{Map}(\Sigma_S A, S) \vdash \mathrm{Map}^{\mathrm{over}\ S}(A,B)$ at the term $\Gamma \vdash \mathrm{pr}_1 : \mathrm{Map}(\Sigma_S A, S)$ detecting the isofibration. First of all we need to make sense of the dependent type, in order to introduce a dependent version of mapping pace from A to B dependent over $\mathrm{Map}(\Sigma_S A, S)$.

Construction 2.10.13.1. Let $\vdash \Gamma$ anima and let $\Gamma \vdash X, S$ type and $\Gamma.S \vdash B$ type. Consider the dependent type $\Gamma, f : \mathrm{Map}(X,S), x : X \vdash B(f(x))$ type. Since $\Gamma.\mathrm{Map}(X,S)$ is an anima, we can consider $\Gamma, f : \mathrm{Map}(X,S) \vdash \mathrm{Sec}_{x:X} B(f(x))$ type. Now, in the case $\Gamma.S \vdash A$ type, with $X \equiv \Sigma_S A$, we can define $\Gamma, f : \mathrm{Map}(\Sigma_S A, S) \vdash \mathrm{Map}^{\mathrm{over}\ S}(A,B)(f) :\equiv \mathrm{Sec}_{x:\Sigma_S A} B(f(x))$ type. In particular, we can consider its fibre at $\mathrm{pr}_1 : \Sigma_S A \to S$.

Proposition 2.10.13.2 (Local mapping space as base change)**.** Let $\vdash \Gamma$ anima and let $\Gamma \vdash X, S$ type and $\Gamma.S \vdash B$ type. We have an equivalence of types $\Sigma_{f:\mathrm{Map}(X,S)} \mathrm{Sec}_{x:X} B(f(x)) \simeq \mathrm{Map}(X, \Sigma_{s:S} B(s))$ over Γ. In particular, if $\Gamma.S \vdash A$ type and $X \equiv \Sigma_S A$, we get an equivalence $\Sigma_{f:\mathrm{Map}(\Sigma_S A, S)} \mathrm{Map}^{\mathrm{over}\ S}(A,B)(f) \simeq \mathrm{Map}(\Sigma_S A, \Sigma_S B)$ over Γ. Thus, by Theorem 2.8.3.1, we get a homotopy cartesian square

$$\begin{array}{ccc} \mathrm{Map}_S(A,B) & \longrightarrow & \mathrm{Map}(\Sigma_S A, \Sigma_S B) \\ \downarrow & & \downarrow{\scriptstyle (\mathrm{pr}_1)_*} \\ \Delta^0_\Gamma & \xrightarrow[\mathrm{pr}_1]{} & \mathrm{Map}(\Sigma_S A, S) \end{array} .$$

Proof. This follows from Non-axiom of choice. □

This means that, in the above assumptions, functions of types $A \to B$ over $\Gamma.S$ are the same as functions $\Sigma_S A \to \Sigma_S B$ that commute with the isofibrations pr_1. Semantically, this corresponds to the theory over Γ being the expressed by the local tribe, with arrows being commutative triangles over fibrations.

2.10.14 Adjunctions

The language of mapping spaces gives us access to the fundamental manipulations of Hom-sets in the present setting. We saw in Internal Yoneda Lemma that we were able to recover an internal version of the Yoneda Lemma making use of mapping spaces. Now we complete the picture by showing that we can in fact express adjunctions. We will do this by translating the meta-theoretical adjunction in Proposition 2.6.8.2 into the language of our theory.

Proposition 2.10.14.1. Let $\vdash \Gamma$ anima, and consider $\Gamma \vdash A$ type. For every $\Gamma.A \vdash B, D$ type and $\Gamma.A.B \vdash C$ type, we have an equivalence of types over $\Gamma.A$

$$\mathrm{Map}_A(\Sigma_B C, D) \simeq \mathrm{Map}_{A.B}(C, D[\mathrm{p}]).$$

Proof. By Meta Yoneda Lemma for animae, to show an equivalence of mapping spaces it suffices to provide a bijection between their terms. Thus the claim follows by Proposition 2.6.8.2 combined with function extensionality. □

Inspired by these, we can introduce the dependent product Π_A, when it exists, as a right adjoint to the weakening as happens in the semantics. This will fully determine the desired type.

2.11 Groupoid core

Up to now, groupoid core was introduced to formalise the idea of objectwise statements. We can now study more extensively the key properties of the groupoid core and understand its picture in the logic of the theory. This is one of the key concepts of the theory, enabling us to automatically turn any type into an "animated collection", or "statement". Let us summarise the similarities and difference between a type $\Gamma \vdash A$ type over $\vdash \Gamma$ anima and its groupoid core $A^{\simeq}$:

(1) $\Gamma.A^{\simeq}$ is an anima, in particular a "collection", thus can be manipulated with ease.

(2) $A^{\simeq}$ and A have the same generalised terms, under the functorial assignment $\gamma_A : A^{\simeq} \to A$.

(3) $A^{\simeq}$ does not have all the generalised terms of A, indeed there is no functor in the opposite direction of γ_A in general.

(4) Arbitrary functors from some $\Gamma \vdash S$ type into A are not the same as functors from S into $A^{\simeq}$, unless S is an anima.

Now, a first point of interest is to understand what happens when we apply the groupoid core operator onto an anima. Under the perspective of groupoid core as the canonical way to turn types into animae, we expect it not to have an action on animae, as we anticipated. This is a consequence of the rules of the groupoid core.

Lemma 2.11.0.1 (Groupoid core is inert on animae)**.** Let $\vdash \Gamma$ anima and $\Gamma \vdash A$ anima. Then, the function $\Gamma.A^{\simeq} \vdash \mathrm{ev} : A[p]$ is an equivalence of types over Γ.

Proof. We can construct an inverse $\Gamma.A \vdash \lambda(q) : A^{\simeq}[p] \equiv A[p]^{\simeq}$ out of the universal term of A, thanks to the fact that A is an anima. This is an inverse to ev. Indeed, one composite is given by $\Gamma.A \vdash \mathrm{ev}_{\lambda(q)} : A[p]$ which is homotopic to the identical assignment q via $\mathrm{comp}^{\mathrm{Sec}}_{A.A[p]}(q)$. Conversely, function extensionality will help us to equate the other composite to the identical assignment: by Lemma 2.10.6.4, we get $\lambda(\mathrm{ev}_q) = q$. □

Therefore, when A is itself an anima, the groupoid core is the identical operator. This is the case of what morally happens in homotopy type theory for every type, as all of them are collections. Indeed, every context is well-behaved, i.e. an anima, and the canonical operator turning a type into an anima is identical. This is the reason why we do not see or even need such an operator.

2.11.1 Functoriality of groupoid core

As the dependent mapping space is functorial by Construction 2.10.4.12, then so is the groupoid core, being the dependent mapping space from the empty extension. Because the groupoid core has a deep interest on its own, let us recover the construction in this specific case. In particular, this will be linked to the interpretation of the groupoid core as a right adjoint to the inclusion of animae into types.

Lemma 2.11.1.1 ($(-)^{\simeq}$ as right adjoint)**.** Let $\vdash \Gamma$ anima and let $\Gamma \vdash A, B$ type. Assume that $\Gamma \vdash A$ anima. Then, every functor $f : A \to B$ factors uniquely through $\gamma_B : B^{\simeq} \to B$. In other words, the functor $(\gamma_B)_* : \mathrm{Map}(A, B^{\simeq}) \to \mathrm{Map}(A, B)$ is an equivalence.

Proof. The reason we cannot do it for any A is related to the fact that we can assign generalised terms of B to those of $B^{\simeq}$ only when these are over animae. In this case, we have $\Gamma.A \vdash \mathrm{ev}_f : B[p]$, and since we have a well formed $\Gamma.A \vdash B^{\simeq}[p] \simeq B[p]^{\simeq}$, we can convert that into $\Gamma.A \vdash \lambda(\mathrm{ev}_f) : B^{\simeq}[p]$. This gives rise to a functor $\lambda a.(\lambda(\mathrm{ev}_f)(a)) : A \to B^{\simeq}$ with the wished property. In particular, by Meta Yoneda Lemma for animae, this lifts to an equivalence between the mapping spaces as in the claim. □

We can interpret Lemma 2.11.1.1 by saying that $(-)^{\simeq}$ is a right adjoint of the inclusion of animae into types, as we wish to have in category theory. In particular, a slight generalisation, or consequence, of this argument is the following:

Construction 2.11.1.2 (Functoriality of $(-)^{\simeq}$)**.** Let $\vdash \Gamma$ anima and $\Gamma \vdash A, B$ type with a functor $f : A \to B$. We want to make the groupoid core functorial. Indeed, note that for any term $\Gamma.A^{\simeq} \vdash b : B$ we can consider $\Gamma.A^{\simeq} \vdash \lambda(b) : B^{\simeq}$ such that $\mathrm{ev}_{\lambda(b)}$ is homotopic to b. Therefore, consider $\Gamma, a : A^{\simeq} \vdash \mathrm{ev}_f(\mathrm{ev}_a) : B$, from which we define define the assignment $\Gamma, a : A^{\simeq} \vdash f^{\simeq}(a) :\equiv \lambda(\mathrm{ev}_f(\mathrm{ev}_a))$. This determines a functor $f^{\simeq} : A^{\simeq} \to B^{\simeq}$, and corresponds to including a into A and then evaluating f followed by the λ operator.

Lemma 2.11.1.3. Let $\vdash \Gamma$ anima and $\Gamma \vdash A, B$ type. Let $f : A \to B$ be a functor that induces a homotopy bijection of terms in context Γ.

(1) The functor $f^{\simeq} : A^{\simeq} \to B^{\simeq}$ is an equivalence.

(2) If $\Gamma \vdash A$ anima, then the induced functor $f : A \to B^{\simeq}$ is an equivalence.

Proof. By the rules of the groupoid core, $f^{\simeq}$ still induces a homotopy bijection of terms. By the animation rules this is between animae, thus by Meta Yoneda Lemma for animae we conclude (1). (2) follows from the same argument, or by (1) combined with Lemma 2.11.0.1. □

The functoriality condition in Construction 2.11.1.2 allows us to make the functors $\gamma_A : A^{\simeq} \to A$, for $\Gamma \vdash A$ type, natural, that is

Lemma 2.11.1.4. Let $\vdash \Gamma$ anima and $\Gamma \vdash A, B$ type with a functor $f : A \to B$. The following square commutes

$$\begin{array}{ccc} A^{\simeq} & \xrightarrow{\gamma_A} & A \\ {\scriptstyle f^{\simeq}}\downarrow & & \downarrow{\scriptstyle f} \\ B^{\simeq} & \xrightarrow[\gamma_B]{} & B \end{array}.$$

Proof. By the computational terms of the functors γ_B and $f^{\simeq}$, the evaluation term of the lower composite is $\Gamma, a : A^{\simeq} \vdash \mathrm{ev}_{\gamma_B}(\mathrm{ev}_{f^{\simeq}})$. We know that, for $b : B^{\simeq}$, $\mathrm{ev}_{\gamma_B}(b)$ is homotopic to ev_b in B, and, for $a : A^{\simeq}$, $\mathrm{ev}_{f^{\simeq}}(a)$ is homotopic in $B^{\simeq}$ to $\lambda(\mathrm{ev}_f(\mathrm{ev}_a))$. By Lemma 2.10.4.15, the composite $\gamma_B \circ f^{\simeq}$ has evaluation $\mathrm{ev}_{\gamma_B \circ f^{\simeq}}(a)$ homotopic in B to $\mathrm{ev}_{\lambda(\mathrm{ev}_f(\mathrm{ev}_a))}$, and thus to $\mathrm{ev}_f(\mathrm{ev}_a)$, for $a : A^{\simeq}$. This is clearly homotopic in B to the evaluation $\mathrm{ev}_{f \circ \gamma_A}(a)$ for $a : A^{\simeq}$ by Lemma 2.10.4.15 and construction of γ_A. We conclude the claim by function extensionality, as the two composite have the same evaluation function. □

2.11.2 Identity in groupoid core

Function extensionality determined the identity type of every dependent mapping space. Therefore, in particular, we inherit a description of the identity type of the groupoid core which we want to look at in detail. First of all, Construction 2.10.4.5 yields, in this setting, a functor

$$\mathrm{pwise}(a, a') : \mathrm{Id}_{A^{\simeq}}(a, b)^{\simeq} \to \mathrm{Id}_A(\mathrm{ev}_a, \mathrm{ev}_b)$$

over $\Gamma, a : A^{\simeq}, b : A^{\simeq}$. Function extensionality claimed this to be an equivalence:

Proposition 2.11.2.1 (Identity in groupoid core)**.** Let $\vdash \Gamma$ anima and let $\Gamma \vdash A$ type. Then, we have an equivalence of types $\mathrm{pwise}(a, a') : \mathrm{Id}_{A^{\simeq}}(a, b)^{\simeq} \xrightarrow{\sim} \mathrm{Id}_A(\mathrm{ev}_a, \mathrm{ev}_b)^{\simeq} \simeq \mathrm{Id}_A(\mathrm{ev}_{\gamma_A}(a), \mathrm{ev}_{\gamma_A}(b))^{\simeq}$ over $\Gamma, a : A^{\simeq}, b : A^{\simeq}$.

This says that an equality between two terms of $A^{\simeq}$ in $A^{\simeq}$ is the same as an equality between the corresponding two terms in A. This is close to saying that the functor $\gamma_A : A^{\simeq} \to A$ is an embedding, but this will be provable within the theory in the future. For the moment, we are happy about having a faithful description of equality of A inside $A^{\simeq}$. This means that we can think of terms of A and of $A^{\simeq}$ in context Γ indifferently. In light of this:

Convention. Let $\vdash \Gamma$ anima and $\Gamma \vdash A$ type. Given $\Gamma \vdash a : A^{\simeq}$, we may denote $\Gamma \vdash a :\equiv \mathrm{ev}_a : A$. In particular, we may write in form of judgement $\Gamma, a : A^{\simeq} \vdash a : A$. In particular, given an assignment to each $a : A$ of $f(a) : B$, we may denote the associated objectwise statement simply $\Gamma, a : A^{\simeq} \vdash f(a) : B$.

Conversely, given $\Gamma \vdash a : A$, we may write $\Gamma \vdash a :\equiv \lambda(a) : A^{\simeq}$. Note that this does *not* give a functorial assignment, as it only holds for generalised terms. However, we can write its relative objectwise statement, which would then appear as $\Gamma, a : A^{\simeq} \vdash a : A^{\simeq}$. This notation is not an abuse, as this assignment is in fact homotopic to the identical one.

2.11.3 Objectwise statements

The first reason why we introduced groupoid cores in the theory was to have access to objectwise statements, to recover one of the key features of category theory in the type theoretical language. In particular, the presence of a groupoid core allows us to recover, in some sense, the idea in homotopy type theory that every meta-theoretical assignment can be written as a judgement. Indeed, by the fundamental principle of animae, these assignments are objectwise statement, that have a judgemental formulation which is functorial *in the groupoid core*. This helped us to formulate function extensionality in the first place. Now, an open question we had from Remark 2.10.3.2 was to prove that the two formulations of objectwise statements, meta-theoretically and internally, are in fact equivalent. Indeed, we showed that starting from a meta-theoretical formulation (exhaustive or minimal), considering the internal form and then again the meta-theoretical formulation associated to it, we get back the starting rule. We are missing to show the converse, which is an easy consequence of function extensionality. Assume we have the internal formulation of an objectwise statement $\Gamma, a : A^{\simeq} \vdash f(a) : B[p]$. The associated meta-theoretical rule consists for every $\Gamma \vdash S$ anima of a family of assignments $\Gamma.S \vdash a : A[p] \mapsto \Gamma.S \vdash f_S(a) :\equiv f(\lambda(a)) : B[p]$, and its internalisation is the judgement $\Gamma, a : A^{\simeq} \vdash \widetilde{f}(a) :\equiv f_{A^{\simeq}}(\mathrm{ev}_a) :\equiv f(\lambda(\mathrm{ev}_a))$. But by function extensionality, we deduced that $\lambda(\mathrm{ev}_a)$ is homotopic to a, so that we get by Lemma 2.4.3.1 a homotopy from $f(\lambda(\mathrm{ev}_a))$ to $f(a)$, that makes the obtained judgement homotopic to the starting one. We conclude that the three given formulations of objectwise statements seen in Remark 2.10.3.2 are all properly equivalent compatibly with homotopy. We will prefer the internal formulation for its functorial properties, although the minimal meta-theoretical formulation can be simpler to control. The exhaustive formulation gives us an idea of all the possible deduction we can make given an objectwise statement.

Remark 2.11.3.1. A functorial statement always implies its objectwise version: given $\Gamma, x : A \vdash f(x) : B$ we can construct $\Gamma, x : A^{\simeq} \vdash f(\mathrm{ev}_{\gamma_A}(x)) : B$, which is extended by the original statement. Vice versa, functoriality is a strictly stronger condition than holding objectwise. The importance of objectwise statements in category theory lies also in the fact that some global properties can surprisingly be proven out of objectwise conditions. The present theory is now able to formalise this idea, overcoming the limits of [RS17]. The ability to express these weaker statements is a feature that will play a key role in the further developments of this theory.

2.11.4 Groupoid core as $\mathbf{Map}(\Delta^0, -)$

There is another very special case of mapping space we did not study yet, which is the mapping space from the point. Because, for every $\vdash \Gamma$ anima with $\vdash \Gamma$ ctx, we have $\Gamma.\Delta^0_\Gamma \simeq \Gamma$, it will not be surprising for the reader that $\mathrm{Map}_\Delta(\Delta^0_\Gamma, A) \simeq A^{\simeq}$ for every $\Gamma \vdash A$ type. This will be interesting to interpret $(-)^{\simeq}$ from a logical perspective. In particular, we can regard $f^{\simeq}$ as a postcomposition functor from this point of view.

Proposition 2.11.4.1. Let $\vdash \Gamma$ anima.

(1) For every $\Gamma \vdash A$ type, we have an equivalence $\varphi_A : \mathrm{Map}(\Delta^0_\Gamma, A) \xrightarrow{\sim} A^{\simeq}$ of types over Γ.

(2) For every $\Gamma \vdash A, B$ type, we have $\Gamma, f : \mathrm{Map}(A, B) \vdash \mathrm{Id}_{\mathrm{Map}(A^{\simeq}, B^{\simeq})}(\varphi_B \circ (f_* \circ \varphi_A^{-1}), f^{\simeq})$. In other words, we have a commutative square

$$\begin{array}{ccc} \mathrm{Map}(\Delta^0_\Gamma, A) & \xrightarrow{f_*} & \mathrm{Map}(\Delta^0_\Gamma, B) \\ \downarrow \wr & & \downarrow \wr \\ A^{\simeq} & \xrightarrow[f^{\simeq}]{} & B \end{array} .$$

Proof. The argument is analogous to Lemma 2.10.6.12 coming from $\Gamma.\Delta^0_\Gamma \simeq \Gamma$ over Γ. □

In other words, we have another way to write the groupoid core up to equivalence. Though, the functorial action on functors of this alternative form coincides with the usual one.

Corollary 2.11.4.2. Let $\vdash \Gamma$ anima and $\Gamma \vdash A$ anima. There is an equivalence $\mathrm{Map}(\Delta^0_\Gamma, A) \to A$.

Proof. Follows by the equivalences Proposition 2.11.4.1 and Lemma 2.11.0.1. □

2.11.5 The logical role of the groupoid core

As we mentioned, in our theory the interpretation of types as statements is not appropriate. Indeed, statements are, by default, thought as collections of their proofs. Because our theory, unlike homotopy type theory, does distinguish collections from arbitrary types, this interpretation works for animae only. However, for those types in anima context, we could think of the groupoid core as a canonical way to turn a type into an underlying statement, i.e. a collection of its "proofs", despite this realising a loss of information. For instance, as we want the types of our theory to represent general structures in which we can organise statements, and in particular theories as in Types as theories, we are associating to a theory its underlying statement. Let us give a naive example in order to have a better intuition: the theory of groupoids will eventually be represented as a type inside our theory. This is clearly not a collection, as it would contain the whole structure of functors between ∞-groupoids (doing what homotopy type, a theory about statements, could not do). Taking its groupoid core would reduce the theory of groupoids to the statement "there exists a groupoid". This coincides with the collection of its proofs, namely the exhibitions of a groupoid. Here we see how the present theory discusses about much richer structures than homotopy type theory does, and the interpretation of types as statements is faithful only concerning their groupoid cores. However, the information concerning the higher structure in the type will be heavily influenced.

2.12 A syntax beyond animae

Summarising, the basic syntax we provided is a framework to develop synthetic category theory. It axiomatised the existence of an intensional identity type, terminal type, dependent sum, and mapping spaces. Out of these, we constructed product types, homotopy pullbacks and groupoid core. In particular, it obeys to the fundamental principle of animae, for which the rules are the same over animae only, and it does not possess dependent products in general.

The most important features of all, though, is the presence of types that are not animae. Indeed, although at first it seemed that the new feature of our theory, compared to homotopy type theory, was that of "adding" the notion of anima among all contexts, it is better to think of animae as homotopy types, and all the non-animae types as the structure we are "adding" to the underlying theory of collections. Indeed, those are the types that behaves differently compared to homotopy type theory: they are not as statements. In this sense, we presented a syntax of a type theory that does not speak only about animae, but about much more general structures. In particular, all other non-animae types could be something much more general than statements, such as theories, recalling the idea of Types as theories. This freedom is crucial if we want, eventually, the theory to describe itself.

Soon, the syntax will get more and more distant from homotopy type theory, in order to do synthetic category theory. The reader might think it is already a very far framework from homotopy type theory, the theory of ∞-groupoids, due to this asymmetry and the lack of dependent products. Indeed, philosophically speaking, the theory was constructed and motivated with clear differences from homotopy type theory. However, if one imposed that $\vdash \Gamma\,\mathrm{ctx}$ is in fact equivalent to $\vdash \Gamma$ anima, then the main differences we introduced would collapse (apart from the unstrictification process that we could perform in homotopy type theory as well). Indeed, mapping spaces would be defined arbitrarily for all types, and their rules would coincide with those of a dependent product, as the animation rule would be a tautology. Indeed, the distinction between collection of maps type and functor category makes sense in our setting only, also philosophically: in a theory where the rules are the same in every context, these notion would collapse.

Finally, we made this theory *agnostic* in the sense that it tries not to make use of the judgemental strict equality. This is often replaced with an internal, *propositional* equality, as it happens for ∞-categories. The

resulting structure consisting of types and functors of types (over some context) fits into an agnostic setting where composition is non-strict, but associative only up to homotopy. Therefore, the structure might be an ∞-category as well, as we aim.

Now it will be our goal to impose enough rules so that (dependent) types would be (families of) ∞-categories and functors of types would be functors of (families of) ∞-categories. Within it, we expect animae to be subdued to obey to homotopy type theoretical rules, and recover the theory of ∞-groupoids out of them. We plan to construct the theory so that it will describe the whole theory of ∞-groupoids and of ∞-categories internally to a type, doing what homotopy type theory could not do. This is possible only due to the expressive power that allows our theory to talk about theories themselves.

Synthetic Category Theory

We constructed the foundational language of the theory, which borrowed some features and ideas from homotopy type theory but was subdued to profound changes to endow our types with more freedom. Now that this substrate is complete, our goal is to develop theory of synthetic categories in it. Indeed, we have the necessary expressive power to impose rules characterising types as categories. Let us start by introducing appropriate terminology translating the seen concepts into the setting of category theory.

3.0.1 Foundations and terminology

We will talk about *synthetic categories in context* Γ to mean types in context Γ. In particular, we reserve the name of *synthetic categories* to those synthetic categories in an anima context. This because, in light of the fundamental principle of animae, our theory will describe as actual categories all those dependent types over animae: there is no distinction between the rules governing the types over an anima Γ or in empty context. This means that some of the rules and concepts will be introduced for all synthetic categories in context, whereas others will hold for synthetic categories only. For instance, only proper synthetic categories will have a good notion of objects and arrows, and we will use this terminology in those situations.

On the other hand, all synthetic categories in context have a notion of functors into them (and we will introduce a notion of natural transformations between them). This is true also ordinarily, in the sense that $\mathrm{CAT}_{/\Gamma}$ is not the theory of categories for arbitrary Γ, but it still has a notion of arrows (and 2-cells). Indeed, given an anima $\vdash \Gamma$ anima and two synthetic categories $\Gamma.\Delta \vdash A, B$ in context $\Gamma.\Delta$, a functor between them is a term of $\Gamma \vdash \mathrm{Map}_{\Delta}(A, B)$ type. Note that every synthetic category in context should come equipped with an anima of which the context is an extension. However, the choice of this anima does not matter up to homotopy, as the terms of $\mathrm{Map}_{\Delta}(A, B)$ are in homotopy bijection with functions of types from A to B, which are terms of $\Gamma.\Delta.A \vdash B[p]$ type.

Remark 3.0.1.1. In this chapter we will denote the terminal type in some context Γ as $\Gamma \vdash \Delta^0$ type, by leaving the context omitted. Indeed, we will introduce other simplices, and we prefer to choose a coherent, not heavy, notation for all simplices.

In general, we will adopt the convention that whenever we use some terminology coming from the previous chapter (products, dependent sums, contractibility) about synthetic categories, we will mean the same notion for the corresponding types. For instance, note that the language of synthetic categories in context comes equipped with notions of functors, equivalences, commutative triangles of functors and commutative squares of functors.

3.0.2 Terms and objects

Let us now introduce the first distinction in terminology between synthetic categories in context and synthetic categories.

Remark 3.0.2.1. Let $\vdash \Gamma$ ctx, and let $\Gamma \vdash A$ type be a synthetic category in context Γ.

(1) A *generalised term* of A is the datum of a context extension by a type $\Gamma \vdash S$ type and a term $\Gamma.S \vdash a : A[p]$. We say its *context* is S. This is the same as a functor $S \to A$ over Γ.

(2) If $\vdash \Gamma$ anima, an *object of* A is the datum of a context extension by a type $\Gamma \vdash S$ type such that $\vdash \Gamma.S$ anima and a term $\Gamma.S \vdash a : A[p]$. We say its *context* is S. This is the same as a functor $S \to A$. Note that, for synthetic categories, every object is a generalised term but not conversely.

(3) An *absolute term of* A is just a term $\Gamma \vdash a : A$, when we wish to emphasise it lies in context Γ. Note that this is always a generalised term. If $\vdash \Gamma$ anima, then we may call it an *absolute object*.

Note that only proper synthetic categories (those in anima contexts) come equipped with a notion of objects. This is a first instance of a categorical fundamental concept that synthetic categories in context do not have in general. Instead, synthetic categories in context only have a notion of term. Terms will still have a lot of expressive power, but they will be nothing more than functors with values in the category we are studying. On the other hand, objects will have a special role in the theory of categories (over an anima). In particular, they are the key idea of objectwise statements. There will be rules that will talk about synthetic categories in context and their terms in general, and other rules that will be about synthetic categories and their objects. This asymmetry is part of the richness of category theory: we want to be able to prove objectwise properties with more ease, and to prove functorial properties out of them eventually.

Note that, for ordinary categories, objects are just functors from the point, here they are functors from all animae. This makes the concept well-behaved from a type theoretical perspective, in presence of the fundamental principle of animae. Indeed, it should not make any difference to look at functors from the point or from any other anima. In particular, for synthetic categories, every construction which we will give pointwise will automatically objectwise: every construction we will perform will be compatible with base change along animae. Therefore, we will be able to provide the same construction for the weakened types $\Gamma.S \vdash A[p]$ type for any $\Gamma \vdash S$ anima and its absolute terms in the same way, syntactically. Instead, the constructions we will give about synthetic categories in arbitrary context will need to be formed functorially, in order to talk about all generalised terms of a type.

Under this, we can reinterpret Meta Yoneda Lemma for animae by saying that if two synthetic categories $\Gamma \vdash A, B$ type such that $\vdash \Gamma.A$ anima and $\vdash \Gamma.B$ anima have the same objects compatibly with base change, then they are equivalent. This is what allows us to regard them as "collections" as in Remark 2.9.2.3.

Remark 3.0.2.2 (The universal object)**.** Let $\vdash \Gamma$ anima and let $\Gamma \vdash A$ type be a synthetic category. Then, it comes equipped with a *universal object*, given by its groupoid core $\Gamma, a : A^{\simeq} \vdash a : A$, corresponding to the canonical functor $\gamma_A : A^{\simeq} \to A$. The term "universal" has a precise meaning: proving something for such a term will imply proving it for any other object. This is because any object $\Gamma, s : S \vdash a(s) : A$ of A is a base change of $\Gamma, a : A^{\simeq} \vdash a : A$ along $\Gamma, s : S \vdash \lambda(a(s)) : A^{\simeq}$, which is equivalent to $\overline{a} : S \to A$ factoring uniquely through $A^{\simeq}$ as proven in Lemma 2.11.1.1.

The goal is now to enrich the basic syntax we introduced via enough rules and axioms so that synthetic categories can be regarded as ∞-categories. For instance, we still have to determine a structure of internal directed morphisms on types. Furthermore, since we did not declare what an anima context is, the theory is still agnostic about it, and could still be speaking about ∞-groupoids such as in homotopy type theory. However, we strongly want to avoid every context to be an anima, in order for our types to be much more general than statements.

3.1 Internal morphisms

First of all, we want to endow synthetic categories in context with an internal structure, and a first step is to introduce internal *directed* morphisms between terms. This will be a structure for all synthetic categories in context, as it will allow us to define the notion of natural transformation of functors. Furthermore, in the case of synthetic categories (over animae), it will define the notion of arrows between objects. However, in the early stages of the theory, we will discuss more in general about morphisms of terms rather than arrows between objects, as the difference will start arising later. For a matter of intuition, though, we will often think about morphisms between terms as arrows in a category, and discuss them in this regard. Though, the reader should keep in mind the more general interpretation, for all synthetic categories in context. Note that, in homotopy type theory, there is not such a difference in principle, as every dependent type is an ∞-groupoid, and the notions of functors and objects in an ∞-groupoid do not differ.

Let us start by recalling the idea of morphisms in homotopy type theory. There, types have the structure of an ∞-groupoid because morphisms can be considered to be equalities of terms. More explicitly, given $\Gamma \vdash A$ type and two terms $\Gamma \vdash a, b : A$, morphisms in A from a to b are regarded as terms of the equality type $\Gamma \vdash \mathrm{Id}_A(a, b)$ type. These always have a unital and associative composition as we saw. The J-elimination rule, also implies that these morphisms are always invertible, and this is the origin of the groupoidal flavour of types in a homotopy type theory. As a consequence, this notion cannot be enough to represent internal morphisms (i.e. arrows or natural transformations of functors) in a category: we want morphisms in synthetic categories not to be invertible in general. This forces us to consider a directed type theory, i.e. to introduce morphisms as entities of a new nature. It will be our responsibility, then, to make sure that the equalities still provide some special morphisms in synthetic categories; because we want to identify objects in a synthetic categories under isomorphisms, these should exactly be the invertible ones.

3.1.1 Interval type

We introduce a 1-simplex, the interval, which represents the poset $\{0 \leq 1\}$, and represents some sort of free "walking morphism" in our setting. The *formation rule* states the existence of a type

$$\frac{\vdash \Gamma \,\mathrm{ctx}}{\Gamma \vdash \Delta^1 \,\mathrm{type}}$$

In particular, in every context $\vdash \Gamma$ ctx, the 1-simplex is obtained as the weakening of the synthetic category $\vdash \Delta^1$ at Γ by compatibility with base change. The *introduction rule* equips it with two canonical terms

$$\frac{\vdash \Gamma \,\mathrm{ctx}}{\Gamma \vdash 0, 1 : \Delta^1}$$

The interval does not come equipped with any relation: being regarded as a free arrow, it must not be equipped with any elimination rule or induction principle. Let us now define the notion of morphism of terms. Despite the name, this should not be regarded as an arrow in a category, but more in general as a natural transformation between functors valued into the category.

3.1.2 Morphisms and natural transformations

The introduction of the interval type is crucial to develop a theory of categories, as it allows us to talk about arrows in categories and natural transformations between functors.

Definition 3.1.2.1 (Morphisms and arrows)**.** Let $\vdash \Gamma$ ctx and consider a synthetic category $\Gamma \vdash A$ type.

(1) A *morphism of absolute terms in* A is a functor $f : \Delta^1 \to A$.

(2) A *morphism of generalised terms in* A *(in context* S*)*, or *natural transformation of functors* $S \to A$ is a functor $f : \Delta^1 \times S \to A$.

(3) If $\vdash \Gamma$ anima, a *(generalised) arrow* in A is morphism of objects (generalised objects). Explicitly, it is a functor $\Delta^1 \to A$, or more in general $\Delta^1 \times S \to A$ for some $\Gamma \vdash S$ anima.

Note that, in the same way we defined a morphism between absolute terms in a type in (1), we defined morphisms between generalised terms in a type in (2), and by exploiting the fact that generalised terms in A are functors into A, this provides the notion of morphism between two arbitrary functors into A, and thus the reason why we call them *natural transformations*. Once again, we see that for synthetic categories, as in (3), we have a proper notion of arrow. This will be the corresponding concept to that of arrow in an ordinary category. In order to bring on a common discussion about the above notions, our intuition should picture them as natural transformations of functors, in general.

Remark 3.1.2.2. For any $\Gamma \vdash A$ type, and $f : \Delta^1 \to A$ morphism in A. Then, we have evaluation functions $\Gamma, x : \Delta^1 \vdash \mathrm{ev}_f(x) : A$. In particular, a base change along 0 and 1 yields $\Gamma \vdash f(0) :\equiv \mathrm{ev}_f(0) : A$ and $\Gamma \vdash f(1) :\equiv \mathrm{ev}_f(1) : A$. We call these respectively the *source* and the *target* of f. This makes us understand why we regard them as morphisms of terms.

In the same way, given $\Gamma \vdash S$ type and $g : \Delta^1 \times S \to A$ a natural transformation of functors $S \to A$ comes equipped with $\Gamma, x : \Delta^1, a : A \vdash \mathrm{ev}_f(x, a) : B$, which gives rise to $\Gamma, a : A \vdash f(0)(a) :\equiv \mathrm{ev}_f(0, a) : B$ and $\Gamma, a : A \vdash f(1)(a) :\equiv \mathrm{ev}_f(1, a) : B$. We call these the *source* and the *target* of f, and they are generalised terms of B at A. The same clearly holds for arrows between objects.

Now we can say with precision what it means, given two terms, to be a morphism between or natural transformation between them, justifying the terminology we used.

Definition 3.1.2.3. Let $\Gamma \vdash A$ type.

(1) Given $f : \Delta^1 \to A$ morphism of absolute terms, and given $\Gamma \vdash a, b : A$, we say that f is a *morphism from* a *to* b, and write $f : a \to b$, if we have homotopies $f(0) = a$ and $f(1) = b$ in A.

(2) Given $\Gamma \vdash S$ type with $g : \Delta^1 \times S \to A$ and $\Gamma.S \vdash a, b : A$, we say that f is a *morphism from* a *to* b, or a *natural transformation from* a *to* b is we have homotopies $f(0) = a$ and $f(1) = b$ in $A[p]$.

Remark 3.1.2.4. It might look unusual to the reader that we are discussing separately the notion of morphism of absolute terms and morphism of generalised terms (natural transformations) in a type. This is because theory still lacks a good way of unifying them: it is true that a morphism of generalised terms in $\Gamma \vdash A$ type in context S is a morphism of absolute terms in $\Gamma.S \vdash A[p]$ type. However, we want to assemble these to a type, and there is no type yet whose absolute terms are morphisms of absolute terms of A and whose generalised terms are morphisms of generalised terms of A. For instance, even if we consider the case $\vdash \Gamma$ anima, the morphisms of absolute terms in A are exactly the terms of $\Gamma \vdash \mathrm{Map}(\Delta^1, A)$ type. On the other hand, the morphisms of generalised terms in A in context S are exactly the terms of $\Gamma \vdash \mathrm{Map}_S(\Delta^1, A[p])$ type, which is *not* the weakening of $\mathrm{Map}(\Delta^1, A)$ at S. Indeed, they do not even live over the same context. As a consequence, we cannot really formulate functorial statements over all morphisms between (absolute and generalised) terms without fixing their context: a morphism f in a context could either vary in $\mathrm{Map}(\Delta^1, A)$ or $\mathrm{Map}(\Delta^1 \times S, A) \simeq \mathrm{Map}_S(\Delta^1, A)$ for a fixed $\Gamma \vdash S$ type (where the equivalence easily follows from Meta Yoneda Lemma for animae). A functorial statement cannot talk about f any morphism in A, with S varying and possibly empty. This is a discrepancy in the theory we would like to solve, and will turn out to be a motivation to introduce dependent products in our theory.

On the other hand, we do not have such problems for arrows in a synthetic category. Indeed, given $\vdash \Gamma$ anima, and $\Gamma \vdash A$ type, the type $\Gamma \vdash \mathrm{Map}(\Delta^1, A)$ it the type of *arrows in* A. Indeed, a generalised term at $\Gamma \vdash S$ anima is a term of $\Gamma.S \vdash \mathrm{Map}(\Delta^1, A)[p] \equiv \mathrm{Map}(\Delta^1, A[p])$. Still, it does not contain informations about the natural transformations in A between functors whose domain is not an anima.

Now, we introduce the most important morphism in a type, that is, the identity morphism.

Definition 3.1.2.5 (Identity morphism)**.** Let $\Gamma \vdash A$ type.

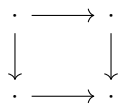
(1) For any absolute term $\Gamma \vdash a : A$, the *identity morphism of* a is the term $\Gamma, x : \Delta^1 \vdash a : A$. This corresponds to the functor $\Delta^1 \to \Delta^0 \xrightarrow{a} A$.

(2) For any generalised term $\Gamma, s : S \vdash a(s) : S$, the *identity natural transformation (morphism) of* a is the term $\Gamma, s : S, x : \Delta^1 \vdash a(s) : A$. This corresponds to the functor $\Delta^1 \times S \xrightarrow{\mathrm{pr}_2} S \xrightarrow{a} A$.

Remark 3.1.2.6. Let $\Gamma \vdash A$ type. For the universal term, we get a universal identity morphism $\Gamma, a : A, x : \Delta^1 \vdash a : A$. This corresponds to the functor $\mathrm{pr}_2 : \Delta^1 \times A \to A$. Its base change along some $a : S \to A$ gives the identity morphism at a. In particular its base change over 0 and 1 are the identical assignments. Therefore, for any (generalised term) $a : A$, we have $1_a : a \to a$.

The introduction of the interval into the theory, allows to talk about a new very important shape of commutative diagrams, namely commutative squares.

Definition 3.1.2.7 (Commutative square)**.** Let $\vdash \Gamma$ ctx and let $\Gamma \vdash A$ type. A *commutative square in* A is a functor $\Delta^1 \times \Delta^1 \to A$ in context Γ. We might denote it as



with the obvious names of the vertices and the arrows.

We could consider the relative generalised version where we look at functors $S \times \Delta^1 \times \Delta^1 \to A$, and call it a commutative square of generalised terms (or of natural transformations). We do not insist on this distinction because soon we will overcome the discrepancy between morphisms of absolute terms and natural transformation from Remark 3.1.2.4.

Remark 3.1.2.8. Let $\vdash \Gamma$ ctx and $\Gamma \vdash A, B$ type with two functors $f, g : A \to B$. For every natural transformation $\alpha : f \Rightarrow g$, every arrow $f : \Delta^1 \to A$ in A provides a *naturality square* $\Delta^1 \times \Delta^1 \xrightarrow{1_{\Delta^1} \times f} \Delta^1 \times A \xrightarrow{\alpha} B$ of the expected form.

As much as studying the arrows in an ordinary category is a fundamental piece of its knowledge, we will be extremely interested into studying arrows into synthetic categories and natural transformations of functors into synthetic categories in context. Given $\vdash \Gamma$ ctx and $\Gamma \vdash A$ type, we refer to its *morphism structure* to mean the data of its morphisms.

Remark 3.1.2.9. Up to now, we have always used groupoid core as a good replacement of a type over anima that has its same objects. Although we never focussed on the loss of information this brings, up to now, the notion of morphism will finally make this loss explicit. We will not be able to prove it now, but because Δ^1 will not be an anima, then the internal morphisms in the groupoid core of $\Gamma \vdash A^{\simeq}$ type will differ from the internal morphisms in $\Gamma \vdash A$ type. Therefore, $A^{\simeq}$ will have a different morphism structure compared to A.

3.2 Dependent product

The language of mapping spaces allows us to state rules that determine the dependent product constructor in our theory. In particular, this will give rise to an internal Hom: the functor category. This is very important because the mapping space, despite being so fundamental for manipulations, does not contain all the information of the higher-dimensional cells, in the same way the groupoid core is a loss of information on the internal structure as seen in Remark 3.1.2.9. Indeed, the mapping space $\mathrm{Map}(A, B)$ was introduced as a replacement for the Hom-set in the homotopy category $\mathrm{Hom}_{\mathrm{Ho}(\mathrm{Type}_{/\Gamma})}(A, B)$ and its manipulations. However, in the same way $\mathrm{Hom}_{\mathrm{Ho}(\mathrm{Type}_{/\Gamma})}(A, B)$ does not capture all the information of the functor category $\mathrm{Fun}(A, B)$ classically, the same happens for $\mathrm{Map}(A, B)$. For instance, we will observe that the the functor category will have a much richer morphism structure compared to the mapping space $\mathrm{Map}(A, B)$, consisting of the morphisms between functors we expect: natural transformations. These are definitely pieces of informations we want to study while developing category theory. As already motivated, though, dependent products will not exist in general. Because we definitely need them in the empty context, they will be introduced at least for all types over animae.

3.2.1 Motivation for the dependent product

Like the dependent sum and the dependent mapping space, the dependent product takes a type $\Gamma \vdash A$ type over $\vdash \Gamma$ anima and a dependent type $\Gamma.A \vdash B$ type and returns a type $\Gamma \vdash \Pi_A B$ type. However, our theory will not only be endowed with dependent products over types that live over animae, but many more: indeed, it will include dependent products indexed by all cocartesian types. Because the current theory cannot formulate such a concept yet, we will limit ourselves to introducing dependent products in the mentioned settings, but also allowing B to live over an arbitrary context extension of $\Gamma.A$. In other words, we are able to exponentiate (i.e. form a dependent product) over all types that are the weakening of a types over an anima. This feature will be crucial in the early developments of the theory.

In particular, when we work over an anima, we can form both dependent mapping spaces and dependent products, thus it is a good setting to compare the two type constructors. Here, the rules of the dependent product will be close to those of the dependent mapping space, and in homotopy type theory they collapse onto the same type constructor. The reader will notice that we will need to split the usual motivations and usage of Π in homotopy type theory between Sec and Π in our theory, where we can see

the difference between the two types. For instance, the dependent mapping space was shown to have a partially dual role with respect to the dependent sum, in the same sense the quantifiers $\forall$ and $\exists$ are dual. The dependent product takes this further, and enjoys the following properties which Sec does not fulfil:

(1) In the same way the dependent sum is a sum operation over the fibres over a dependent type, its dual should be a product operation between the fibres. The dependent mapping space, though, fails to represent a product of the fibres. Indeed, it is some sort of simplified version of it. In order to see this, we may just consider a type $\Gamma.\Delta^0_\Gamma \vdash A$ type with $\vdash \Gamma$ anima. Taking the dependent mapping space over Δ^0_Γ yields $A^\simeq$ by Proposition 2.11.4.1, which differs from the product of the fibres over the terms of Δ^0_Γ, which is simply A, unless A is an anima. The information they share in only about the generalised objects, and for the categorical-oriented reader, this is the essentially the same as saying that two ordinary categories have the same objects and nothing else. On the other hand, the purpose of the dependent product is to introduce a product operation between the fibres of a dependent type. This is the reason why the type $\Gamma \vdash \Pi_A B$ type will also be denoted as $\Gamma \vdash \Pi_{x:A} B(x)$ type.

(2) We want a type of functions which is in fact an internal Hom in the theory, the functor category. Indeed, as the dependent sum is the left adjoint of the weakening by Proposition 2.10.14.1, the dependent product will be introduced as the right adjoint. For this purpose, as seen in Proposition 2.10.14.1 and Lemma 2.11.1.1, the language of mapping spaces gives us access to the language of adjunctions. This will provide the theory with an internal Hom.

(3) As anticipated, we would like types of functors to exist in more general settings than just over animae. This would not even make sense for the dependent mapping space, due to the animation rule. Instead, dependent product will exist much more in general, and this will be essential to talk about a morphism category, for instance.

(4) Because the notion of natural transformation serves as a good notion of morphism of functors in category theory, we would like functors to fit into a type whose morphisms are natural transformations. This will not be the case for ordinary mapping spaces: its rules do not determine generalised terms such as their morphisms. On the other hand, the dependent product will come equipped with a direct characterisation of all its generalised terms, and morphisms in a functor category will correspond to natural transformations.

(5) As pointed out in Remark 3.1.2.4, given a type A, we want to treat morphisms between absolute and generalised terms in A in a unified way. In particular, we want instances $f : \Delta^1 \to A$ in a context to stand for all possible natural transformations of functors into A, and not just the morphisms of absolute terms. For this purpose, we need to have a type whose absolute terms are morphisms between absolute terms of A, and whose generalised terms are natural transformations of functors in A. For the same problems as in (4), the dependent mapping space does not allow us to assemble these to a unique type.

The price of accessing these pieces of information in the internal structure is that the obtained type will not be easily manipulated: it will not be a collection in the sense of Remark 2.9.2.3. For this reason, although the terms of $\Gamma \vdash \Pi_{x:X} A(x)$ will be assignments for every $x : X$ of a term of $A(x)$, the interpretation as $\forall$ quantifier is more suited for the dependent mapping space, as that is indeed a way to form statements.

3.2.2 Rules of the dependent product

Let us start with the rules. First of all, we have a *formation rule*.

$$\frac{\vdash \Gamma \text{ anima} \quad \vdash \Gamma.\Xi, \Gamma.\Delta \text{ ctx} \quad \Gamma.\Xi.\Delta[p^\Xi] \vdash B \text{ type}}{\Gamma.\Xi \vdash \Pi_\Delta B \text{ type}},$$

where p^Ξ clearly means the iterated weakening over the length of Ξ of all the components of Δ. The way this rule is stated with two arbitrary context extensions of Γ should not be confusing, as the two play a different role. The context extension $\vdash \Gamma.\Delta$ ctx is a slight generalisation of the case when $\Delta \equiv A$ type of length one, in the same way we had for dependent mapping spaces. In other words, the rule above will

most commonly be used in the following form

$$\frac{\vdash \Gamma \,\mathrm{anima} \quad \Gamma \vdash A\,\mathrm{type} \quad \vdash \Gamma.\Xi\,\mathrm{ctx} \quad \Gamma.\Xi.A[p^{\Xi}] \vdash B\,\mathrm{type}}{\Gamma.\Xi \vdash \Pi_A B\,\mathrm{type}}.$$

The role of Ξ, however, makes this formation rule possible in more general settings compared to dependent mapping spaces. Indeed, in the future, dependent products will exist over a wider range of dependent types than just types over animae. This rule includes part of this more general type dependences: we have dependent products which are indexed not only by types over animae, but also by their weakenings. This will just be the beginning, as later on we will impose the existence of dependent products indexed by any cocartesian type. However, having access to this piece of information before anything else will be crucial, as we will start to observe in this section already. Note, furthermore, that the notation should include Ξ, but we prefer to omit it for the sake of simplicity.

Remark 3.2.2.1. Alongside the dependent mapping space, the dependent product is a second instance of a type constructor that is not defined in all contexts, which is unusual for the reader experienced in homotopy type theory. Because we care about the generalised terms of these types, we are interested in studying their weakenings. However, the compatibility of dependent mapping space with weakening was well-defined only over animae, and this prevented us from describing the internal structure. On the other hand, this was never truly a problem: by the animation rule, we could weaken the mapping space at itself. This, for instance, was what allowed us to make the evaluation assignment functorial in Remark 2.10.2.4. The dependent product, though, will not have an animation rule, but this will not create any issues. Indeed, the dependent product is fully compatible with weakening, by the way we formulated the rules. Indeed, assume Γ anima, $\vdash \Gamma.\Delta, \Gamma.\Xi$ ctx and $\Gamma.\Xi.\Delta[p^{\Xi}] \vdash B$ type. Then, for any $\Gamma.\Xi \vdash S$ type, we can consider $\Gamma.\Xi.\Delta[p^{\Xi}], s : S \vdash B$ type and form $\Gamma.\Xi, s : S \vdash \Pi_\Delta B$. Compatibility with base change says that this type coincides with the weakening of $\Pi_\Delta B$ at S.

Unlike the previous type constructors, we are going to introduce the dependent product with a different scheme, as an adjoint, as we characterised the dependent sum in Proposition 2.10.14.1. For this purpose, the *elimination rule* will state the existence of a counit map of the adjunction:

$$\frac{\vdash \Gamma \,\mathrm{anima} \quad \vdash \Gamma.\Delta, \Gamma.\Xi\,\mathrm{ctx} \quad \Gamma.\Xi.\Delta[p^{\Xi}] \vdash B\,\mathrm{type}}{\Gamma.\Xi.\Delta[p^{\Xi}].(\Pi_\Delta B)[p] \vdash \mathrm{ev} : B[p]}.$$

This corresponds to a functor $\mathrm{ev} : (\Pi_\Delta B)[p] \to B$ over $\Gamma.\Xi.\Delta[p^{\Xi}]$. In particular, we have a functorial assignment $\Gamma.\Xi.\Delta[p^{\Xi}], f : \Pi_{\delta:\Delta} B(\delta) \vdash \mathrm{ev}_f : B$ analogously to the elimination rule of Sec. In conclusion, we have an evaluation operator also for terms of the dependent product. This notation, equal to that of dependent mapping space, will not be confusing, as we will soon see a deep relation between Sec and Π.

Remark 3.2.2.2. As in the case of the dependent mapping space, but due to a different reason, the evaluation map is functorial, so that we can construct an action on equalities as $\Gamma.\Xi, f, f' : \Pi_A B, \alpha : \mathrm{Id}_{\Pi_A B}(f, f'), a : A \vdash \mathrm{ap}_{\mathrm{ev}}(\alpha)(a) : \mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))$, analogously to Remark 2.10.2.4.

We want the rules to make this evaluation map the counit of an adjunction.

Remark 3.2.2.3. Let $\vdash \Gamma$ anima, with $\vdash \Gamma.\Delta, \Gamma.\Xi$ ctx and $\Gamma.\Xi \vdash C, D$ type. Then we can consider the assignment $\Gamma.\Xi.\Delta[p^{\Xi}], f : \mathrm{Map}_{\Xi}(C, D), y : C \vdash \mathrm{ev}_f(y) : D$ which determines, by the animation rule of the mapping space, a functor $(-)[p] : \mathrm{Map}_{\Xi}(X, D) \to \mathrm{Map}_{\Xi.\Delta[p^{\Xi}]}(C[p], D[p])$, that embodies the functorial action of weakening.

Now, applied to our setting, where $\vdash \Gamma$ anima, $\vdash \Gamma.\Delta, \Gamma.\Xi$ ctx and $\Gamma.\Xi.\Delta[p^{\Xi}] \vdash B$ type, for any $\Gamma.\Xi \vdash C$ type we have an induced functor

$$\mathrm{ev}_*((-)[p]) : \mathrm{Map}_{\Xi}(C, \Pi_A B) \xrightarrow{(-)[p]} \mathrm{Map}_{\Xi.\Delta[p^{\Xi}]}(C[p], (\Pi_A B)[p]) \xrightarrow{\mathrm{ev}_*} \mathrm{Map}_{\Xi.\Delta[p^{\Xi}]}(C[p], B)$$

in context Γ. The *adjunction rule* realises this functors as an equivalence of types:

$$\frac{\vdash \Gamma \,\mathrm{anima} \quad \Gamma \vdash A\,\mathrm{type} \quad \Gamma.A \vdash B\,\mathrm{type} \quad \Gamma \vdash C\,\mathrm{type}}{\Gamma \vdash \Pi\mathrm{radj}_{A;B} : \text{is-}\simeq\text{-Equiv}(\mathrm{ev}_*((-)[p]))}.$$

This realises the functor

$$\mathrm{ev}_*((-)[p]) : \mathrm{Map}_\Xi(C, \Pi_\Delta B) \xrightarrow{\sim} \mathrm{Map}_{\Xi.\Delta[p^\Xi]}(C[p], B)$$

as an equivalence of synthetic categories, actually of animae, in the same spirit of Proposition 2.10.14.1 for Σ_A. In particular, the dependent product Π_A is defined as a right adjoint to the weakening, dually to the dependent sum Σ_A. This almost concludes the rules of the dependent product. Our next goals will be, in the first place, to understand better this definition, to construct the internal Hom $\mathrm{Fun}(-,-)$, and to see the similarities and the differences with the dependent mapping space.

3.2.3 Dependent product and dependent mapping space

The given rules that exhibit the dependent product as a right adjoint to the weakening allow us to deduce various properties right away. As we anticipated, the dependent product is a richer version of the mapping space, with the wished morphism structure, mimicking what happens for a type over anima and its groupoid core. This is emphasised by the following

Lemma 3.2.3.1. Let $\vdash \Gamma$ anima and let $\Gamma \vdash A$ type. Then, we have an equivalence of types $\Pi_\emptyset A \simeq A$ over Γ.

Indeed, we regarded the groupoid core $\mathrm{Sec}_\emptyset A$ as a loss of information from $A \simeq \Pi_\emptyset A$. More in general, we expect to recover the rules we had for the mapping spaces determining their terms, with function extensionality replaced by Lemma 2.10.6.4.

Remark 3.2.3.2 (Π and Sec have the same terms)**.** We saw that the dependent product shares with the dependent mapping space an evaluation operator, translating any term of the dependent product into an appropriate dependent function. The adjunction rule allows us to reverse the direction, in the same way we have for the dependent mapping space. Indeed, for $C \equiv \Delta^0$ in the adjunction rule, the functor $\mathrm{ev}_*((-)[p]) : \mathrm{Map}_\Xi(\Delta^0, \Pi_A B) \to \mathrm{Map}_{\Xi.\Delta[p^\Xi]}(\Delta^0, B)$ is an equivalence. By Proposition 2.5.0.2, this determines a homotopy bijection of terms between $\Gamma.\Xi \vdash \Pi_\Delta B$ type and $\Gamma.\Xi.\Delta[p^\Xi] \vdash B$ type, realised by the ev operator in one direction. The converse assignment, in the case $\Gamma \vdash \Delta \equiv A$ type will again be denoted as $\Gamma.\Xi, x : A \vdash f(x) : B(x) \mapsto \Gamma.\Xi \vdash \lambda x.f(x) : \Pi_\Delta B$ following the notation of the dependent mapping space. Note that this is fixed by the term $\Pi\mathrm{radj}_{A;B}$, but the choice will not really matter by Proposition 2.10.10.5.

In particular, for $\Xi \equiv \emptyset$, we get that that terms of $\Pi_\Delta B$ in context Γ are in homotopy bijection with terms of $\mathrm{Sec}_\Delta B$, as both correspond to dependent functions $\Gamma.\Delta \vdash f : B$. This pointwise correspondence, together with the animation rule, automatically gives rise to a functor $\mathrm{Sec}_\Delta B \to \Pi_\Delta B$ over Γ, assigning to each term $\Gamma.\Delta \vdash f : B$ the corresponding term of $\Pi_\Delta B$, which we will still denote with f. This functor commutes with the evaluation, in the sense that given $f : \mathrm{Sec}_\Delta B$, we can map it into $f : \Pi_\Delta B$ and consequently we have an evaluation function $\Gamma.\Delta \vdash \mathrm{ev}_f : B$ which is homotopic to the evaluation of f as term of the dependent mapping space. Conversely, a function $\Gamma.\Delta \vdash g : B$ gives rise to $\Gamma \vdash \lambda(g) : \mathrm{Sec}_\Delta B$, which mapped into $\Pi_\Delta B$ yields the λ-construction of $\Pi_\Delta B$, $\Gamma \vdash \lambda(g) : \Pi_\Delta B$. All this looks like an abuse of notation, but we will now see it is actually coherent with the conventions we fixed. In order to see this, we must go more in-depth with the relation between the dependent mapping space and the dependent product.

Lemma 3.2.3.3. Let $\vdash \Gamma$ anima, $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash B$. The canonical functor $\mathrm{Sec}_\Delta B \to \Pi_\Delta B$ over Γ induces an equivalence $\mathrm{Sec}_\Delta B \to (\Pi_\Delta B)^\simeq$.

Proof. As the functor determines a homotopy bijection of objects, the claim follows by Lemma 2.11.1.3. □

This justifies why we call the image of $f : \mathrm{Sec}_\Delta B$ under $\mathrm{Sec}_\Delta B \to \Pi_\Delta B$ still f, and why we gave the same names to the assignments for Sec and for Π: a term $\Gamma.\Delta \vdash f : B$ can be translated into $\Gamma \vdash \lambda(f) : \mathrm{Sec}_\Delta B$ or into $\Gamma \vdash \lambda(f) : \Pi_\Delta B$, and the same for the evaluation. This is coherent with the convention of not distinguishing the objects of a type and its groupoid core notationwise, and removes the ambiguity.

This means, in the above notation, that the types $\Pi_\Delta B$ and $\mathrm{Sec}_\Delta B$ are similar on objects, yet widely different for what concerns the more general internal structure. Indeed, $\mathrm{Sec}_\Delta B$ forgets about all the structure of generalised terms that $\Pi_\Delta B$ has. This makes us understand the reason of such a deep difference between

the rules of the two types: the dependent product is *not* an anima, thus we could not even use Meta Yoneda Lemma for animae to fully determine it from its generalised objects. Instead, we had to make explicit all its generalised terms:

Proposition 3.2.3.4 (Generalised terms of the dependent product)**.** Let $\vdash \Gamma$ anima and let $\vdash \Gamma.\Delta, \Gamma.\Xi$ ctx with $\Gamma.\Xi.\Delta[p^{\Xi}] \vdash B$ type. For any $\Gamma.\Xi \vdash S$ type, the terms of $\Gamma.\Xi.S \vdash (\Pi_{\Delta} B)[p]$ are in homotopy bijection with the terms of $\Gamma.\Xi.\Delta[p^{\Xi}].S[p] \vdash B[p]$ type. In particular, if $\Gamma \vdash \Delta \equiv A$ type, the terms of $\Gamma.\Xi \vdash \Pi_A B$ type are in homotopy bijection with dependent functions from $S \times A$ to B.

Proof. By Remark 3.2.2.1 we can rewrite weakenings of dependent products as dependent products, of which we know terms by Remark 3.2.3.2. □

This is all the amount of information the dependent product contains, as opposed to its groupoid core: the dependent mapping space by Lemma 3.2.3.3. The price we pay in order to access this information, is that the obtain type Π will be harder to manipulate, not being a collection. We now wish to focus our attention onto some special kinds of dependent products, which are the ones corresponding to ordinary mapping spaces among all dependent mapping spaces: functor categories.

3.2.4 Functor categories

The introduction of the dependent product, i.e. the the right adjoint to the base change, allows, in the non-dependent cases, the construction of internal Homs, the functor categories. Indeed, we want to be able to form a functor category between any two synthetic categories at least, and even more in general. In some sense, these will take further what dependent mapping spaces could not do.

Definition 3.2.4.1 (Functor category)**.** Let $\vdash \Gamma$ anima with $\Gamma \vdash A$ type and $\vdash \Gamma.\Xi$ ctx with $\Gamma.\Xi \vdash B$ type. Consider the dependent type $\Gamma.\Delta.A[p^{\Xi}] \vdash B[p]$ type. The *functor category from* A *to* B is $\Gamma.\Xi \vdash \mathrm{Fun}(A, B) :\equiv \Pi_A(B[p])$ type.

We note immediately that functor categories are allowed to be defined in more general premises than the ordinary mapping spaces. For instance, they might not live over animae. This will be crucial in some examples we will see soon. First of all, let us give a trivial instance of this. Since in every context Γ the terminal Δ^0 is given by weakening $\vdash \Delta^0$ type at Γ, then we can always form dependent products over the terminal type:

Lemma 3.2.4.2. Let $\Gamma \vdash A$ type. We have an equivalence of types $\mathrm{Fun}(\Delta^0, A) \simeq A$ over Γ.

Proof. The rules realise a homotopy bijection of generalised terms, by means of Lemma 2.7.0.5. □

Remark 3.2.4.3. Despite the name, the functor category will not always be a synthetic category in the proper sense, as it does not live over an anima in general. Indeed, we can form a functor category from any synthetic category to a synthetic category in some context Γ, and the resulting functor category will only be a synthetic category in context Γ. This is what happens in Lemma 3.2.4.2. For this reason, a more appropriate name would be *functor type* or *functor category in context*. Apart from these cases, what is surely true, in general, is that for any two synthetic categories we can form both the functor category and the mapping space. In particular, this functor category will be a proper synthetic category.

Now we observe that, as in the case of dependent mapping spaces, when we have arbitrary synthetic categories in context we can still form the local functor categories.

Definition 3.2.4.4 (Local functor category)**.** Let $\vdash \Gamma$ anima with $\vdash \Gamma.\Xi$ ctx and $\Gamma.\Xi \vdash C, D$ type, we can consider $\Gamma.\Xi.C \vdash D[p]$ type. The *local functor category over* Ξ *from* C *to* D is $\Gamma \vdash \mathrm{Fun}_{\Xi}(C, D) :\equiv \Pi_{\Xi.C}(D[p])$ type.

Remark 3.2.4.5. By Remark 3.2.3.2, it immediately follows that for $\vdash \Gamma$ anima with $\Gamma \vdash A$ type, and $\vdash \Gamma\Xi$ ctx, with $\Gamma.\Xi \vdash B$ type, the terms of $\Gamma.\Xi \vdash \mathrm{Fun}(A, B)$ type are in homotopy bijection with functors $A[p^{\Xi}] \to B$ over $\Gamma.\Xi$. In particular, by Proposition 2.5.0.2 and Remark 2.9.2.3, for any $\Gamma.\Xi \vdash S$ type we get an equivalence of

animae $\mathrm{Map}_{\Xi}(S, \mathrm{Fun}(A,B)) \simeq \mathrm{Map}_{\Xi}(S \times A[p^{\Xi}], B)$ over Γ, that means that the functor category is a right adjoint to the product with the weakening of A. In particular, for $\Xi \equiv \emptyset$, we have

$$\mathrm{Map}(S, \mathrm{Fun}(A,B)) \simeq \mathrm{Map}(S \times A, B),$$

making $\mathrm{Fun}(A,-)$ a right adjoint of $A \times -$.

For the local case, consider $\vdash \Gamma.\Delta$ ctx and $\Gamma.\Delta \vdash C, D$ type, the terms of $\Gamma \vdash \mathrm{Fun}_{\Delta}(C,D)$ type are in homotopy bijection with those of $\Gamma \vdash \mathrm{Map}_{\Delta}(C,D)$ type, i.e. functors $C \to D$. However, this does not lift to their generalised terms: we only have $\mathrm{Fun}_{\Delta}(C,D)^{\simeq} \simeq \mathrm{Map}_{\Delta}(C,D)$ over Γ. The generalised terms of $\mathrm{Fun}_{\Delta}(C,D)$ over S are actually in homotopy bijection with functors $S \times C \to D$ over $\Gamma.\Delta$ by Proposition 3.2.3.4.

To sum up, we constructed a type whose absolute terms are functors of types, as for the dependent mapping space, but with a fully determined, richer, internal structure of generalised terms.

3.2.5 Evaluation maps

The next step is to study one of the most important functors we will see: the evaluation map.

Construction 3.2.5.1 (Evaluation and composition functors)**.** Let $\vdash \Gamma$ anima with $\Gamma \vdash A$ type and $\vdash \Gamma.\Xi$ ctx and $\Gamma.\Xi \vdash B$ type. The counit $\Gamma.\Xi.\mathrm{Fun}(A,B)[p] \vdash \mathrm{ev} : B[p]$ determines an *evaluation functor*

$$\mathrm{ev} : A[p^{\Xi}] \times \mathrm{Fun}(A,B) \to B$$

over $\Gamma.\Xi$. In particular, under $\mathrm{Map}(A[p^{\Xi}] \times \mathrm{Fun}(A,B), B) \simeq \mathrm{Map}(\mathrm{Fun}(A,B), \mathrm{Fun}(A,B))$ from Remark 3.2.4.5, it corresponds to $1_{\mathrm{Fun}(A,B)}$, as always happens for the counit of an adjunction. Now, assume that B is the weakening of some $\Gamma' \vdash B'$ type with $\vdash \Gamma'$ anima as well. In other words, we assume that $\Gamma.\Xi \equiv \Gamma'.\Xi'$ together with $\Gamma.\Xi \vdash B \equiv B'[p^{\Xi'}]$ type. In this setting, we consider a further type $\Gamma.\Xi \vdash C$ type in the same context as B. In particular, we can form functor categories $\Gamma.\Xi \vdash \mathrm{Fun}(A,C)$ type and $\Gamma.\Xi \vdash \mathrm{Fun}(B,C)$ type.

(1) The functor $A[p^{\Xi}] \times \mathrm{Fun}(A,B) \times \mathrm{Fun}(B,C) \xrightarrow{\mathrm{ev} \times 1_{\mathrm{Fun}(B,C)}} B \times \mathrm{Fun}(B,C) \xrightarrow{\mathrm{ev}} C$ over $\Gamma.\Xi$ determines, by adjunction, a *composition functor* $\mathrm{comp} : \mathrm{Fun}(A,B) \times \mathrm{Fun}(B,C) \to \mathrm{Fun}(A,C)$ over $\Gamma.\Xi$, that maps $(f : A \to B, g : B \to C)$ into $g \circ f : A \to C$ up to homotopy, where this stands for $\lambda(\mathrm{ev}_g \circ \mathrm{ev}_f)$ coherently with the notation for the mapping space.

(2) For every functor $g : B \to C$ over $\Gamma.\Xi$, the composite $A[p^{\Xi}] \times \mathrm{Fun}(A,B) \xrightarrow{\mathrm{ev}} B \xrightarrow{g} C$ over $\Gamma.\Xi$ determines, by adjunction, a *postcomposition with* g *functor* $g_* : \mathrm{Fun}(A,B) \to \mathrm{Fun}(A,C)$, that maps $f : A \to B$ into $g \circ f : A \to C$ up to homotopy.

(3) For every functor $f : A \to B$ and $\Gamma \vdash$, the composite $A[p^{\Xi}] \times \mathrm{Fun}(B,C) \xrightarrow{f \times 1_{\mathrm{Fun}(B,C)}} B \times \mathrm{Fun}(B,C) \xrightarrow{\mathrm{ev}} C$ determines, by adjunction, a *precomposition with* f *functor* $f^* : \mathrm{Fun}(B,C) \to \mathrm{Fun}(A,C)$ mapping $g : B \to C$ into $g \circ f : A \to C$ up to homotopy.

In particular, evaluation of terms of the functor category is compatible with composition.

Remark 3.2.5.2. When $\Gamma.\Xi$ is an anima, we may consider the postcomposition and precomposition maps induced on the groupoid cores. By construction, under the identification in Lemma 3.2.3.3, these maps are the usual postcomposition and precomposition functors between mapping spaces.

3.2.6 The morphism structure of functor categories

We finally observe explicit examples where the functor category carries more information than the mapping space, that is, an example where we care about generalised terms that are not objects.

Remark 3.2.6.1. Let $\vdash \Gamma$ anima with $\Gamma \vdash A$ type and $\vdash \Gamma.\Xi$ ctx. Let $\Gamma.\Xi \vdash B$ type. Then, $\mathrm{Map}_{\Xi}(\Delta^1, \mathrm{Fun}(A,B)) \simeq \mathrm{Map}(\Delta^1 \times A, B)$. This exactly means that the morphisms in $\mathrm{Fun}(A,B)$ over $\Gamma.\Xi$ are in homotopy bijection with natural transformations of functors $A \to B$. In other words, functor categories contain not only the functors as their terms, but also the piece of information consisting of natural transformations between

them. Because natural transformation should indeed embody the idea of being "morphisms of functors", the existence of the functor category allows us to formalise this idea: a natural transformation is a functor $\Delta^1 \to \mathrm{Fun}(A, B)$ inside $\mathrm{Fun}(A, B)$. On the other hand, even for $\Xi \equiv \emptyset$, the mapping space $\mathrm{Map}(A, B)$ does not have such a morphism structure, and natural transformations of functors $A \to B$ can only be seen as objects of the mapping space $\mathrm{Map}(\Delta^1 \times A, B)$.

The above responds to (4) in Motivation for the dependent product. In other words, functor categories contain more information than mapping spaces, including all the information about natural transformations of functors. Imposing a condition on the level of functor categories will be a stronger condition than requiring the same condition but on the mapping spaces, namely their groupoid cores. This would correspond to the associated objectwise statement. On the other hand, it is way less easy to manipulate functor categories or to show that they satisfy a certain condition:

Remark 3.2.6.2. Due to the absence of the animation rule, we cannot rely on Meta Yoneda Lemma for animae to prove equivalences between functor categories. For example, it is true that $\mathrm{Map}(A, B)$ and $\mathrm{Fun}(A, B)$ over some anima Γ have the same terms, but this yields a syntactic proof only about the generalised objects. Indeed, they are far from being equivalent types.

3.2.7 Identity in the dependent product

When we introduced the dependent mapping space, we had to face the issue of understanding equality between its terms. Indeed, in absence of an induction principle, we had to impose a rule making it coincide with pointwise equality of the underlying evaluation functions. Now, we have to face a similar problem for the dependent product: we introduced a new type whose terms are dependent functions of types, and this will come equipped with its notion of equality. In light of Lemma 3.2.3.3 and Proposition 2.11.2.1, the reader might hope that we can deduce function extensionality for dependent product from the one of its groupoid core, the dependent mapping space. This works fine pointwise:

Remark 3.2.7.1. Let $\vdash \Gamma$ anima with $\Gamma \vdash A$ type and $\Gamma.A \vdash B$ type. Then, for any two terms $\Gamma \vdash f, f' : \Pi_A B$, we have an equivalence

$$\mathrm{Id}_{\Pi_A B}(f, f')^{\simeq} \simeq \mathrm{Id}_{(\Pi_A B)^{\simeq}}(f, f')^{\simeq} \simeq \mathrm{Id}_{\mathrm{Sec}_A B}(f, f')^{\simeq} \simeq \mathrm{Sec}_{a:A}(\mathrm{Id}_B(\mathrm{ev}_f(a)), \mathrm{ev}_{f'}(a)).$$

This says that, identifying two functors f and f' as terms of $\Pi_A B$ is the same as identifying them as terms of $\mathrm{Sec}_A B$, and thus as identifying the underlying evaluation functions of types pointwise. Note that the evaluation functions of f and f' as terms of $\Pi_A B$ or of $\mathrm{Sec}_A B$ coincide. A more general argument should work also when Γ is not an anima and A is the weakening of a type that originally lives an anima.

Even recovering the pointwise version is not enough for our purposes, as we would like a functorial statement in f and f'. Furthermore, in this functorial formulation, we would like to remove the groupoid core on the left (as it does not make sense to take the groupoid core over a non-anima context such as $\Gamma, f : \Pi_A B, f' : \Pi_A B$) and replace Sec with Π on the right for a more general formulation that resembles the usual homotopy type theoretical function extensionality. In other words, we will need to impose an extra function extensionality axiom that will be compatible with Remark 3.2.7.1. This does not sound friendly since, in order to formulate function extensionality for the dependent mapping space, we relied on the animation rule. However, the possibility to form the dependent product in more general assumptions will remove these issues.

Construction 3.2.7.2. Let $\vdash \Gamma$ anima with $\Gamma \vdash A$ type and consider $\vdash \Gamma.\Xi$ ctx with $\Gamma.\Xi.A[p^{\Xi}] \vdash B$ type. In Remark 3.2.2.2, we constructed an assignment $\Gamma.\Xi, f, f' : \Pi_A B, \alpha : \mathrm{Id}_{\Pi_A B}(f, f'), a : A \vdash \mathrm{ap}_{\mathrm{ev}}(\alpha)(a) : \mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))$ thanks to the functoriality of evaluation. Now, in Construction 2.10.4.5, we used the fact that on the left of $\mathrm{Id}_{\Pi_A B}(f, f')$ we had an anima; hence, by replacing the identity type with its groupoid core, we could take the dependent mapping space over A. This is not true anymore in this situation, however we can exploit the fact that A is a weakened type that originally lives over an anima Γ. In other words, we can form the dependent product over A without even having to consider the groupoid core of the identity type. In particular we construct a type $\Gamma.\Xi, f, f' : \Pi_A B \vdash f \simeq^{\Pi} f' :\equiv \Pi_{a:A}(\mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a)))$ type in

line with the analogus from Remark 2.10.2.4. Then, we get an assignment $\Gamma.\Xi, f, f' : \Pi_A B, \alpha : \mathrm{Id}_{\Pi_A B}(f, f') \vdash \lambda a.\mathrm{ap}_{ev}(\alpha)(a) : f \simeq^{\Pi} f'$, which determines a functor

$$\mathrm{pwise}^{\Pi}(f, f') :\equiv \lambda\alpha.(\lambda a.\, \mathrm{ap}_{ev}(\alpha)(a)) : \mathrm{Id}_{\Pi_A B}(f, f') \to f \simeq^{\Pi} f'$$

over $\Gamma.\Xi, f, f' : \Pi_A B$, that evaluates at $\alpha : \mathrm{Id}_{\Pi_A B}(f, f')$ as $a : A \mapsto \mathrm{ap}_{ev}(a) : \mathrm{Id}_{B(a)}(\mathrm{ev}_f(a), \mathrm{ev}_{f'}(a))$.

Analogously to the case of the dependent mapping space, the *function extensionality rule for* Π realises this as an equivalence:

$$\frac{\vdash \Gamma\,\mathrm{anima} \quad \Gamma \vdash A\,\mathrm{type} \quad \vdash \Gamma.\Xi\,\mathrm{ctx} \quad \Gamma.\Xi.A[p^{\Xi}] \vdash B\,\mathrm{type}}{\Gamma \vdash \mathrm{funext}^{\Pi}_{A;B} : \text{is-}\simeq\text{-Equiv}(\mathrm{pwise}(q[p], q))}.$$

We will omit the various discussions on the consequences of function extensionality for the dependent product, as they will be the most intuitive generalisation of those we had about the dependent mapping space. Let us sum up some of its implications:

(1) We can check equality of terms in a functor category on the pointwise evaluation functions.

(2) Dependent product is compatible with equivalences.

(3) Equality in the functor category respects composition of functors.

When $\Xi \equiv \emptyset$, we can compare Π and Sec via Lemma 3.2.3.3, and from the combination of this rule with function extensionality of dependent mapping space and Proposition 2.11.2.1 we recover Remark 3.2.7.1.

Remark 3.2.7.3. A future version of the present work will include a more extended discussion on equality in the dependent product following function extensionality. Furthermore, the way function extensionality for dependent products was introduced may be subdued to future changes, as we expect to be able to refine it. What is surely true, though, is that we want the given rule to hold. Thus the reason why we assume it for the current version of the project.

3.2.8 Morphism category

We now focus on the most important example of functor category: the morphism category of a synthetic category in context. In particular, this will not have a corresponding mapping space in general, as it will exist for all types in all contexts. Indeed, the very reason why we formulated the existence of dependent products more in general than that of dependent mapping spaces was to have access to these types.

Construction 3.2.8.1 (The category of morphisms)**.** Let $\Gamma \vdash A\,\mathrm{type}$. Because we have $\vdash \Delta^1\,\mathrm{type}$, we can exponentiate along any weakening of it. In particular, there is $\Gamma \vdash \mathrm{Fun}(\Delta^1, A)\,\mathrm{type}$, whose absolute terms are in homotopy bijection with functors $\Delta^1 \to A$, i.e. morphisms of absolute terms. Its generalised terms in context $\Gamma \vdash S\,\mathrm{type}$ are natural transformations of functors $S \to A$, following from $\mathrm{Fun}(\Delta^1, A[p]) \equiv \mathrm{Fun}(\Delta^1, A)[p]$ as types over $\Gamma.S$. Note that the morphism category is a proper synthetic category (over an anima) if and only if A is.

As a consequence, we can indeed regard morphisms of generalised terms in A as generalised terms in $\mathrm{Fun}(\Delta^1, A)$ and solve the discrepancy the two concepts had a priori, discussed in Remark 3.1.2.4. Up to now they were in fact only absolute terms of distinct types, and there was no way of formulating a functorial statement talking about all possible morphisms of generalised terms in A without fixing a context. In particular, now, whenever we have $\Gamma, f : \mathrm{Fun}(\Delta^1, A)$ in the context, f will stand automatically for all morphisms of absolute and generalised objects. This gives us the ability to talk about functorial statements over all kinds of morphisms in a type. Therefore, we solved the issue addressed in (5) in Motivation for the dependent product. As a consequence, from now on we may call morphisms of generalised terms in A *generalised morphisms of terms in* A, or just *morphisms of terms*. Note that analogous considerations hold for $\mathrm{Fun}(\Delta^1 \times \Delta^1, A)$, meaning that its generalised terms are commutative squares of natural transformations. In particular, we may talk about commutative squares in A to mean commutative squares of natural transformations of functors into A.

Another important feature about the morphism category of a type A is that, even when we are over an anima, the morphism structure of $\mathrm{Map}(\Delta^1, A)$ does not contain the information we wish for. On the other hand the corresponding functor category has an interesting morphism structure as every functor category:

Remark 3.2.8.2. Let $\Gamma \vdash A$ type. By adjunction, as discussed for functor categories in general, there is equivalence of animae $\mathrm{Map}(\Delta^1, \mathrm{Fun}(\Delta^1, A)) \simeq \mathrm{Map}(\Delta^1 \times \Delta^1, A)$. This means that morphisms in $\Gamma \vdash \mathrm{Fun}(\Delta^1, A)$, are commutative squares in A. Intuitively, in ordinary category theory, it is the category of arrows of a category whose morphisms are commutative squares between them.

Now we focus on the construction of very important functors involving the morphism category. The first will lift the identity morphism construction, assigning to $\Gamma, s : S \vdash a(s) : A$ the morphism $\Gamma, s : S \vdash 1_{a(s)} : \mathrm{Map}_S(\Delta^1, A)$, to a functorial assignment over all terms of A, as now we can assemble a common target.

Construction 3.2.8.3 (The identity morphism functor)**.** Let $\Gamma \vdash A$ type. By the adjunction rule, out of the universal identity morphism $\mathrm{pr}_2 : \Delta^1 \times A \to A$, we induce a functor

$$1_{(-)} : A \to \mathrm{Fun}(\Delta^1, A)$$

such that, for any $a : S \to A$ generalised term, the composite $S \xrightarrow{a} A \xrightarrow{1_{(-)}} \mathrm{Fun}(\Delta^1, A)$ corresponds to $1_a : S \times \Delta^1 \to A$. Thus, it maps every (generalised) term of A into its identity morphism in A. In the form of judgement, this realises the assignment $\Gamma, a : A \vdash 1_a : \mathrm{Fun}(\Delta^1, A)$, which can be finally represented functorially thanks to the existence of $\mathrm{Fun}(\Delta^1, A)$ with the desired generalised terms. Applying base change along a generalised $\Gamma.S \vdash a : A[p]$ yields the identity morphism of a.

Now, because Δ^1 has two canonical terms 0 and 1, and evaluating a morphism at them yields respectively the source and the target, we want to have evaluation functors that return the source and the target of a morphism. In particular, we can make these evaluations functorial in $\mathrm{Fun}(\Delta^1, A)$, so that they involve morphisms in A in all contexts.

Construction 3.2.8.4 (Evaluation functors and dependent morhism category)**.** Let $\Gamma \vdash A$ type. Thanks to the existence of the morphism category, we can construct functorial assignments $\Gamma, f : \mathrm{Fun}(\Delta^1, A) \vdash f(0) : A$ and $\Gamma, f : \mathrm{Fun}(\Delta^1, A) \vdash f(1) : A$. This correspond to the functors

$$\mathrm{ev}_0 : \mathrm{Fun}(\Delta^1, A) \xrightarrow{0^*} \mathrm{Fun}(\Delta^0, A) \simeq A \qquad \mathrm{ev}_1 : \mathrm{Fun}(\Delta^1, A) \xrightarrow{1^*} \mathrm{Fun}(\Delta^0, A) \simeq A$$

by using the functors precomposition functor together with Lemma 3.2.4.2, and we call them *evaluation functors*. ev_0 and ev_1 map a morphism $f : x \to y$ in A into x and y respectively, up to homotopy. In particular, they induce a functor $(\mathrm{ev}_0, \mathrm{ev}_1) : \mathrm{Fun}(\Delta^1, A) \to A \times A$ over Γ that embodies to the assignment $\Gamma, f : \mathrm{Fun}(\Delta^1, A) \vdash (f(0), f(1)) : A \times A$.

We now see why the above functor is extremely important. We will often like to consider the category of morphisms $\mathrm{Fun}(\Delta^1, A)$ as dependent over $A \times A$, with a parametrisation over the source and the target. In other words, fixed two terms x and y in A, we would like to consider the type of morphisms from x to y in A. For this purpose, we will follow the fibre type construction for the functor $(\mathrm{ev}_0, \mathrm{ev}_1)$: we have a dependent type

$$\Gamma, x : A, y : A \vdash x \to y :\equiv \mathrm{fib}_{(\mathrm{ev}_0, \mathrm{ev}_1)}(x, y)\,\mathrm{type}\,.$$

We call this type the *dependent morphism category of* A, and we may regard it as a type in context $\Gamma.A.A[p]$ or $\Gamma.A \times A$ indifferently, being them equivalent contexts. In particular, this dependent type is such that $\Sigma_{x:A, y:A} x \to y \simeq \mathrm{Fun}(\Delta^1, A)$ over Γ by Proposition 2.6.9.3. Furthermore, given $\Gamma \vdash a, b : A$, a term of $\Gamma \vdash a \to b$ type is a morphism $f : a \to b$, i.e. $f : \Delta^1 \to A$ with $f(0)$ homotopic to a and $f(1)$ homotopic to b. This enforces to the chosen notation for $f : a \to b$.

Remark 3.2.8.5. The reader might expect to denote such a type $\Gamma, x : A, y : A \vdash \mathrm{Hom}_A(x, y)$ type, however we prefer to avoid this notation as the fibres over terms $x, y : A$ that are not objects will not be animae. We will reserve the notation $\mathrm{Hom}_A(x, y)$ when we will consider fibres over objects of a synthetic category.

We constructed a type of morphisms with a fixed source and target up to homotopy. Not only that: the reader familiar with category theory can already anticipate the next construction.

Construction 3.2.8.6 (The slice category)**.** Let $\Gamma \vdash A$ type, and let $\Gamma \vdash x : A$.

(1) We construct the *slice type of* A *over* x as the base change $\Gamma, y : A \vdash A_{/x}(y) :\equiv y \to x$ type. In particular, $\Gamma \vdash \Sigma_{y:A} A_{/x}(y)$ type has terms in homotopy bijection with a pair of $y : A$ and $f : y \to x$.

(2) We construct the *coslice type of* A *over* x as the base change $\Gamma, y : A \vdash A_{x/}(y) :\equiv x \to y$ type. In particular, $\Gamma \vdash \Sigma_{y:A} A_{/x}(y)$ type has terms in homotopy bijection with a pair of $y : A$ and $f : x \to y$.

These are what we would expect from a (co)slice construction in synthetic category theory (in context).

3.3 Composition of morphisms

The theory finally endowed types with an internal morphism structure, for which we have identity morphisms, dependent morphism categories and (co)slice categories. What is still missing, though, is a good notion of composition of morphisms, which should be associative and unital with respect to identities.

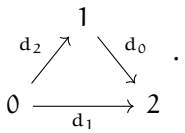
3.3.1 The 2-simplex

In the spirit of ∞-category theory, we need a higher simplex in order for composition of morphisms to be the provision of some data, rather than a property of a morphism. In order to do this, the reader might be surprised in seeing that we already have all the ingredients to construct a 2-simplex, that are the 1-simplex and dependent products. For this purpose, we will keep the focus on morphism categories.

Construction 3.3.1.1 (The 2-simplex)**.** Let $\vdash \Gamma$ ctx. We define the *2-simplex type* (or the *walking commutative triangle*) as the type $\Gamma \vdash \Delta^2 :\equiv \mathrm{Fun}(\Delta^1, \Delta^1)$. Intuitively, it is the poset $\Delta^2 = \{0 \leq 1 \leq 2\}$. This is well-defined in every context and is compatible with base change since the functor category and Δ^1 are. In particular, it falls itself in the class of those types that are, in every context, the weakening of some absolute type $\vdash \Delta^2$ type. Therefore, we can always exponentiate over it. The given definition of the 2-simplex is quite peculiar, hence let us explore it further. First of all, we have the two evaluation maps

$$s_0 :\equiv \mathrm{ev}_0 : \Delta^2 \to \Delta^1 \qquad s_1 :\equiv \mathrm{ev}_1 : \Delta^2 \to \Delta^1.$$

We refer to [CCNW25] for the constructions of functors $d_0, d_1, d_2 : \Delta^1 \to \Delta^2$ via geometric considerations. These are morphisms in Δ^2 such that we have homotopies $d_1(0) = d_2(0)$, $d_0(0) = d_2(1)$ and $d_0(1) = d_1(1)$. These give rise to three distinct terms of Δ^2 which we name 0, 1 and 2 respectively. In other words, the three morphisms are directed as $d_0 : 1 \to 2$, $d_1 : 0 \to 2$ and $d_2 : 0 \to 1$. Thus, we may display Δ^2 as



Moreover, they satisfy simplicial identities, that are equalities as functors $\Delta^1 \to \Delta^1$, thus as terms of Δ^2:

$$s_0 \circ d_0 = \mathrm{id}_{\Delta^1} = s_0 \circ d_1, \qquad s_0 \circ d_2 = \mathrm{const}_0 \qquad s_1 \circ d_0 = \mathrm{const}_1, \qquad s_1 \circ d_1 = \mathrm{id}_{\Delta^1} = s_1 \circ d_2,$$

where const obviously denotes the constant assignment.

3.3.2 Commutative triangles

Now that we have a "walking commutative triangle" Δ^2 in the theory, we can finally define commutative triangles in a synthetic category in context.

Definition 3.3.2.1 (Commutative triangle)**.** Let $\Gamma \vdash A$ type. The *type of commutative triangles* is defined as $\Gamma \vdash \mathrm{Fun}(\Delta^2, A)$ type. A *commutative triangle in* A is a term of this type.

Commutative triangles are then functors $\Delta^2 \to A$ (absolute terms) or more in general functors $S \times \Delta^2 \to A$ (generalised terms in contexts S). We can give a unified treatment between these two since they assemble to a type $\mathrm{Fun}(\Delta^2, A)$. Its compatibility with weakening means that an absolute commutative triangle in A in context S is an absolute commutative triangle in $\Gamma.S \vdash A[p]$ type. Once again, commutative triangles in A should be thought as the notion of commutative triangle of natural transformations of functors into A. In the particular case Γ is an anima, i.e. A is a synthetic category, the objects of $\Gamma \vdash \mathrm{Fun}(\Delta^2, A)$ should be pictured as actual commutative triangles of arrows in a category.

Remark 3.3.2.2. Let $\Gamma \vdash A$ type. The functors d_0, d_1 and d_2 induce the precomposition functors

$$d_0^* : \mathrm{Fun}(\Delta^2, A) \to \mathrm{Fun}(\Delta^1, A) \qquad d_1^* : \mathrm{Fun}(\Delta^2, A) \to \mathrm{Fun}(\Delta^1, A) \qquad d_2^* : \mathrm{Fun}(\Delta^2, A) \to \mathrm{Fun}(\Delta^1, A)$$

from the type of commutative triangles in A to the morphism category of A. These assign to each commutative triangle $t : \Delta^2 \to A$ the morphism $t \mapsto t \circ d_i$ up to homotopy. In particular, for any commutative triangle t, we get three terms, $t(0)$, $t(1)$ and $t(2)$, and three morphisms, $f :\equiv t \circ d_2$, $g :\equiv t \circ d_0$ and $h :\equiv t \circ d_1$, of A. By Construction 3.3.1.1 we know that their sources and targets satisfy relations to fit into a picture

$$\begin{array}{ccc} & t(1) & \\ {\scriptstyle f}\nearrow & & \searrow{\scriptstyle g} \\ t(0) & \xrightarrow[h]{} & t(2) \end{array} .$$

We will always denote a commutative triangle with a picture of this form to make explicit the morphisms it "restricts" to.

3.3.3 Composition of morphisms

Through the notion of commutative triangles in a synthetic category in context, we can define a composition operation between its morphisms of terms. In particular, we will define a dependent type of composition over a pair of two consecutive morphisms, and terms of this types will be the exhibition of a composition.

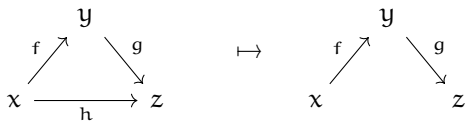
Construction 3.3.3.1 (The type of compositions)**.** Let $\Gamma \vdash A$ type. We consider the type $\Gamma, x : A, y : A, t : A, z : A \vdash (x \to y) \times (y \to z)$ type. By Lemma 2.8.6.2 and Theorem 2.8.3.1, we get a homotopy pullback square

$$\begin{array}{ccc} \Sigma_{x:A,y:A,z:A}(x \to y) \times (y \to z) & \longrightarrow & \mathrm{Fun}(\Delta^1, A) \\ \downarrow & & \downarrow{\scriptstyle ev_1} \\ \mathrm{Fun}(\Delta^1, A) & \xrightarrow[ev_0]{} & A \end{array} .$$

In particular, the terms of $\Gamma \vdash \Sigma_{x:A,y:A,z:A}(x \to y) \times (y \to z)$ type are the datum of $x : A, y : A, z : A$ with two morphisms $f, g : \Delta^1 \to A$ of the form $f : x \to y$ and $g : y \to z$ in A. In other words, it is the type of pairs of consecutive morphisms in A. The reason why we are interested in these, is that they will be composable. Now, we get an induced functor

$$\begin{array}{ccccc} & & {\scriptstyle d_2^*}\frown & & \\ \mathrm{Fun}(\Delta^2, A) & \overset{(d_2^*, d_0^*)}{\dashrightarrow} & \Sigma_{x:A,y:A,z:A}(x \to y) \times (y \to z) & \longrightarrow & \mathrm{Fun}(\Delta^1, A) \\ & {\scriptstyle d_0^*}\searrow & \downarrow & & \downarrow{\scriptstyle ev_1} \\ & & \mathrm{Fun}(\Delta^1, A) & \xrightarrow[ev_0]{} & A \end{array}$$

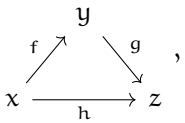
by means of the simplicial identities in Construction 3.3.1.1, together with the description of ev_i as precomposition with $i : \Delta^0 \to \Delta^1$. This maps acts via



by construction. Furthermore, we can equip $\mathrm{Fun}(\Delta^2, A)$ with a functor $(0^*, 1^*, 2^*) : \mathrm{Fun}(\Delta^2, A) \to A \times A \times A$. The fibre construction gives rise to a dependent type over $\Gamma, x : A, y : A, z : A$ of commutative triangles between x, y and z, whose underlying isofibration is $(0^*, 1^*, 2^*)$. Because the induced functor (d_2^*, d_0^*) commutes with the isofibrations from $\mathrm{Fun}(\Delta^2, A)$ and $\Sigma_{x:A,y:A,z:A}(x \to y) \times (y \to z)$ to $A \times A \times A$, the functor (d_2^*, d_0^*) lifts to a functor over $x : A, y : A, z : A$ with values into $(x \to y) \times (y \to z)$ by Proposition 2.6.9.7. In particular, the fibre construction, together with the induction principle of the product, determines a dependent type

$$\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z \vdash \mathrm{comp}_{f,g}(x, y, z) :\equiv \mathrm{fib}_{(d_2^*, d_1^*)}(x, y, z, f, g)\ \mathrm{type}$$

which we call the *type of compositions*. Given $x, y, z : A$, and given two morphisms $f : x \to y$ and $g : y \to z$, a *composition of* f *an* g is a term of $\mathrm{comp}_{f,g}(x, y, z)$. In particular, it is the datum of a commutative triangle $t : \Delta^2 \to A$ that gets mapped into (f, g) under (d_2^*, d_0^*), i.e. of the form



,

as we would expect. In such setting we also call h a *composite morphism* of f and g. This type is such that $\Sigma_{x:A,y:A,z:A,f:x\to y,g:y\to z}\mathrm{comp}_{f,g}(x, y, z) \simeq \mathrm{Fun}(\Delta^2, A)$ over Γ. In other words, we promoted $\Gamma \vdash \mathrm{Fun}(\Delta^2, A)$ type to a dependent type over $\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z$. Furthermore, fixed a choice of $a : A, b : A, c : A$ and of $\bar{f} : a \to b, \bar{g} : b \to c$, we refer to $\mathrm{comp}_{\bar{f},\bar{g}}(a, b, c)$ as the *type of compositions of* $\bar{f}$ *and* $\bar{g}$, and it consist of compositions of $\bar{f}$ and $\bar{g}$. Note that it is still unknown whether this type is always inhabited, and if it is so, whether it is contractible, as we would like for a good notion of composition of morphisms in ∞-categories. The next subsection will be devoted to axiomatising the contractibility of this type.

As we expect from a type of compositions, we may construct an underlying composite map.

Construction 3.3.3.2 (The underlying map)**.** Let $\Gamma \vdash A$ type. We want to make the composition type $\mathrm{comp}_{f,g}(x, y, z)$ dependent over $x \to z$ in the most obvious way, given by "projecting on the side $0 \to 2$". For this, we need to exhibit the fibre construction for a functor $\mathrm{comp}_{f,g}(x, y, z) \to (x \to z)$ over $\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z$. In light of Proposition 2.6.9.7, we can induce this out of a functor between the dependent sums that commutes with the isofibrations. Because $\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z \vdash x \to z$ type is the weakening of $\Gamma, x : A, z : A \vdash x \to z$ type, which sums to $\mathrm{Fun}(\Delta^1, A)$, it suffices to construct a functor $\mathrm{Fun}(\Delta^2, A) \to \mathrm{Fun}(\Delta^1, A)$ that postcomposed with $(\mathrm{ev}_0, \mathrm{ev}_1)$ gives $(0^*, 2^*)$. The functor $d_1^* : \mathrm{Fun}(\Delta^2, A) \to \mathrm{Fun}(\Delta^1, A)$ satisfies this by Construction 3.3.1.1. The induced functor $\mathrm{und}_A(x, y, z, f, g) : \mathrm{comp}_{f,g}(x, y, z) \to (x \to z)$ can then be promoted to a dependent type $\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z, h : x \to z \vdash \mathrm{isComp}_{f,g}(h)(x, y, z) :\equiv \mathrm{fib}_{\mathrm{und}_A(x,y,z,f,g)}(h)$. In particular, its dependent sum over h is equivalent to $\mathrm{comp}_{f,g}(x, y, z)$. Furthermore, for any choice of terms in the context, a term of $\mathrm{isComp}_{f,g}(h)(x, y, z)$ is the exhibition of h as a composition of f and g.

Note that, in line with the spirit of homotopy theory, being a composition of two morphisms is not a property of the morphism h. Instead, it is a datum t, the triangle, that provides h as a composite morphism.

Proposition 3.3.3.3. Composition of morphisms is associative and unital with respect to the identity morphism.

Proof. The proof is omitted and, up to translating the formalism properly, can be found in [CCNW25]. □

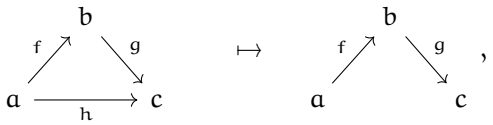
3.3.4 Existence and uniqueness of composition

In order to make composition of morphism unique, we need to state the contractibility of the type of compositions $\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z \vdash \mathrm{comp}_{f,g}(x, y, z)$ type. The *Segal axiom* is stated as

$$\frac{\Gamma \vdash A \text{ type}}{\vdash \mathrm{Segal}_A : \text{is-}\simeq\text{-}\mathrm{Contr}_{\Gamma, A.A[p].A[p^2].q[p^2]\to q[p].q[p^2]\to q[p]}(\mathrm{comp}_{f,g}(x, y, z))}.$$

This imposes all the fibres of $\mathrm{comp}_{f,g}(x, y, z)$ to be contractible, that means that fixed $x, y, z : A$ and $f : x \to y$ and $g : y \to z$, there is one unique inhabitant, up to homotopy, of the fibre $\mathrm{comp}_{f,g}(x, y, z)$. Its terms are in homotopy bijection with compositions of f and g, which thus says that the composition of two consecutive morphisms f and g always exists and is unique up to homotopy.

Remark 3.3.4.1. By Proposition 2.6.10.1, we know that contractibility of the fibre type of a functor, in this case $(d_2^*, d_0^*) : \mathrm{Fun}(\Delta^2, A) \to \Sigma_{x:A,y:A,z:A}(x \to y) \times (y \to z) \simeq \mathrm{Fun}(\Delta^1, A) \times_A \mathrm{Fun}(\Delta^1, A)$, is equivalent to requiring this functor to be an equivalence. Recall that this functor acts as



by forgetting the composite. Thus this functor being an equivalence exactly means that the whole information of the commutative triangle on the left is contained in the spine on the right, i.e. the datum of the two composable morphisms. This is a rephrasing of existence and uniqueness of composition of morphisms.

Remark 3.3.4.2. Given $\Gamma \vdash A$ type, by Lemma 2.5.2.1 the term Segal_A determines a centre of contraction which we call $\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z \vdash g \circ f : \mathrm{comp}_{f,g}(x, y, z)$. In particular, this fixes a choice of composition fro f and g which we denote with $g \circ f$, knowing that, by contractibility, this centre of contraction is unique up to higher-order equality anyway. In particular, by Construction 3.3.3.2, this induces an underlying morphism $\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z \vdash g \circ f :\equiv \mathrm{und}_A(g \circ f)$ which we denote with a slight abuse of notation, for the sake of simplicity.

Corollary 3.3.4.3. Let $\Gamma \vdash A$ type. In context $\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z, h : x \to z$, we have an equivalence of types $\mathrm{isComp}_{f,g}(h)(x, y, z) \simeq \mathrm{Id}_{x\to z}(h, g \circ f)$.

Proof. By definition, $\mathrm{isComp}_{f,g}(h)(x, y, z)$ is defined as the isofibration associated to the function und_A from $\mathrm{comp}_{f,g}(x, y, z)$ to $x \to z$. Because the Segal axiom realises a contraction of $\mathrm{comp}_{f,g}(x, y, z)$ at $g \circ f$, then the corresponding isofibration is equivalent to the isofibration associated to the map detecting the term $g \circ f$ inside $x \to z$, which is by definition the type $\mathrm{Id}_{x\to z}(h, g \circ f)$. □

Corollary 3.3.4.3 says something we may have expected from uniqueness of composition. Indeed, it says that for a map h, it is equivalent to be a composite of f and g or to be homotopic to the canonical composite $g \circ f$. Because the type $\mathrm{Id}_{x\to z}(h, g \circ f)$ has a simpler intuition compared to $\mathrm{isComp}_{f,g}(h)(x, y, z)$, we will often choose to work with the former in the constructions we will see. This choice will usually be a matter of simplicity, as the same constructions or definitions may be given using the latter without relying on the Segal axiom. This, indeed, will just provide us with a cleaner rewriting: by imposing the contractibility of a type, we can replace such a type with the point and work with this instead.

This axiom concludes the discussion about composition of morphisms of terms in a type, as we were able to determine rules realising it as an associative, unital operation, which always exists unique. These are, in fact, the key properties of composition of natural transformations of functors, or of composition of arrows in a category.

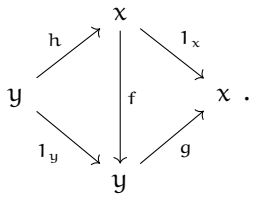
3.4 Isomorphisms

Since we defined composition of internal morphisms in a type and identity morphisms, we can immediately define the concept of isomorphism in a type by following our categorical intuition. First of all, note that the Segal axiom allows us to consider the dependent term $\Gamma, x : A, y : A, z : A, f : x \to y, g : y \to z \vdash g \circ f : x \to z$ by Remark 3.3.4.2, which is canonically a composition of f and g, and furthermore, it is the only one up to homotopy. This allows us to define the notion of a one-sided inverse of f as a morphism g such that, equivalently, the identity is a composition of f with g, or is homotopic to the canonical composite $g \circ f$ by Corollary 3.3.4.3, where the latter reduction makes use of the Segal axiom.

Construction 3.4.0.1 (The isomorphism type)**.** Let $\Gamma \vdash A$ type. We construct $\Gamma, x : A, y : A, f : x \to y \vdash \mathrm{isIso}_A(f)(x,y) :\equiv \Sigma_{g:y\to x}(\mathrm{Id}_{x\to x}(g \circ f, 1_x)) \times \Sigma_{h:y\to x}(f \circ h, 1_y))$ type. Fixed a choice of $x : A, y : A, f : x \to y$, the terms of $\mathrm{isIso}_A(f)(x,y)$ are in homotopy bijection with a pair of morphisms $g, h : y \to x$ equipped with homotopies from $g \circ f$ and $f \circ h$ to 1_x and 1_y inside $x \to x$ and $y \to y$ respectively. In particular, summing over f yields the *isomorphism type of* A *from* x *to* y

$$\Gamma, x : A, y : A \vdash \mathrm{Iso}_A(x,y) :\equiv \Sigma_{f:x\to y}\mathrm{isIso}_A(f)(x,y)\,\mathrm{type}\,.$$

For any two $x, y : A$, an *isomorphism from* x *to* y is a term of the fibre $\mathrm{Iso}_A(x,y)$. This is the datum of $f : x \to y$ equipped with $g, h : y \to x$ such that $g \circ f$ and $f \circ h$ are homotopic to 1_x and 1_y respectively. In other words, by Corollary 3.3.4.3, it is the datum of two commutative triangles



Note that we have a functor $\pi_{\mathrm{Iso}}(x,y) : \mathrm{Iso}_A(x,y) \to (x \to y)$ given by the associated isofibration. This maps any isomorphism (f, g, h) from x to y into the *underlying morphism* f from x to y. Summing over x and y yields the *isomorphism type of* A: $\Gamma \vdash \mathrm{Iso}(A) :\equiv \Sigma_{x:A,y:A}\,\mathrm{Iso}_A(x,y)$ type. An *isomorphism in* A is a term of $\mathrm{Iso}(A)$, i.e. the datum of $x, y : A$ and an isomorphism from x to y. This comes equipped with two functors:

(1) Summing $\pi_{\mathrm{Iso}}(x,y)$ over $x, y : A$ yields $\pi_{\mathrm{Iso}} : \mathrm{Iso}(A) \to \mathrm{Fun}(\Delta^1, A)$, that maps any isomorphism into the underlying morphism.

(2) Its associated isofibration is a functor $(\mathrm{ev}_0, \mathrm{ev}_1) : \mathrm{Iso}(A) \to A \times A$, sends an isomorphism into the source and the target of the underlying morphisms by Proposition 2.6.3.3.

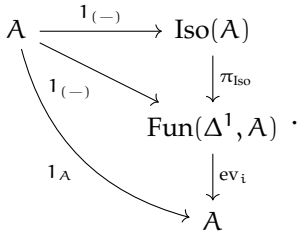
Again, we observe that the exhibition of a morphism $f : x \to y$ as an isomorphism, i.e. a term of $\mathrm{isIso}_A(f)(x,y)$, is a provision of data, that is a left and a right inverse with the respective homotopies.

Remark 3.4.0.2. We could have formed an isomorphism type without relying on the Segal axiom, i.e. without relying on the centre of contraction $g \circ f$ of the composition type. Indeed, we could make use of the appropriate replacement from Corollary 3.3.4.3. One could also give a geometric construction with the correct maps between simplices analogously to Construction 3.3.3.1, but with different choices, to construct a type of "pasting of two triangles" with the right dependence and then taking the right fibres over the identities, and this would turn out to be equivalent to the already discussed ones. The given definition is the most concrete that we choose for the sake of simplicity: since we know that composition is unique, there is no reason to pretend it is not.

Remark 3.4.0.3. The equality in the fibre type $x \to y$ was described in Proposition 2.6.9.10. In particular, consider $\Gamma \vdash l : \mathrm{Id}_{x\to x}(g \circ f, 1_x)$. Here, we obviously consider $g \circ f$ equipped with $(g \circ f)(0) = f(0) = x$ and $(g \circ f)(1) = g(1) = x$ by assumption, and 1_x equipped with the homotopies $1_x(0) = 0$ and $1_x(1) = x$ provided in Remark 3.1.2.6. The term l is the datum of a homotopy between $g \circ f$ and 1_x in $\mathrm{Fun}(\Delta^1, A)$ that is compatible with these given homotopies.

Let us now give the most important example of isomorphism: the identity.

Construction 3.4.0.4 (The identity isomorphism functor)**.** We can construct a functor $1_{(-)} : A \to \mathrm{Iso}(A)$. Consider the functor $\mathrm{pr}_2 : \Delta^2 \to A \to A$. Precomposed with some $1_{\Delta^2} \times a : \Delta^2 \times S \to \Delta^2 \times A$ this yields a commutative triangle $S \times \Delta^2 \to A$ exhibiting 1_a as composite of 1_a and 1_a, in particular inducing a homotopy $\text{id-iso}(a) : \mathrm{Id}_{a \to a}(1_a \circ 1_a, 1_a)$. This, we can consider the assignment $\Gamma, a : A \vdash (1_a, 1_a, \text{id-iso}(a), \text{id-iso}(a)) : \mathrm{isIso}_A(1_a, 1_a)$. In particular, this determines a functor $1_{(-)} : A \to \mathrm{Iso}(A)$ that sends a to 1_a equipped with 1_a as left and right inverse. Therefore, the identity morphism is exhibited as an isomorphism by means of this functor. Note that, for $i \in \{0, 1\}$, the following diagram commutes



3.4.1 Two ways of identifying terms

A crucial question now arises. The introduction of directed morphisms in types, that have a composition operation and an identity morphism, intrinsically gives rise to a notion of isomorphism between terms in a type. Since we are doing category theory, this shall be thought as an isomorphism between objects in a synthetic category, or more in general as a natural isomorphism of functors into a synthetic category in context. In particular, this is the correct relation under which we want to identify objects (functors) in a category (in context). However, as in every type theory, we already had a notion of identification of terms of a type to begin with: propositional equality. We now wish to compare the two notions.

Remark 3.4.1.1. Let $\Gamma \vdash A$ type be a synthetic category in context, and consider terms $a, b : A$.

(1) The notion of morphism from a to b is directed, in the sense that we have a functor $\Delta^1 \to A$ from a to b (possibly in some context). In homotopy type theory, types also have an internal structure, where morphisms, or paths, are terms of the identity type $\mathrm{Id}_A(a, b)$. We saw that an intensional type theory endows types a groupoidal structure with respect to these equalities, which are always invertible due to the path induction. Synthetic category theory also has an underlying intensional type theory where all these relations are still satisfied by equalities: the unstrictification procedure did not alter them as we carefully proved. Therefore, in order for directed morphisms to fit into our theory, we needed to axiomatise them via a deeply different foundation compared to equalities, which led to the introduction of the 1-simplex Δ^1. However, the reader might find unusual that our theory, which proposes to extends the theory of animae to that of categories, has such a deep different nature for the morphisms. In particular, for our theory to extend homotopy type theory, there should still be an interpretation of equalities in a type as (some) morphisms in a category. Moreover, types still have a groupoidal structure with respect to equalities, and it is not clear yet what this should mean if types were categories with their internal notion of morphisms.

(2) We have two different ways of identifying the two terms a and b, i.e. functors into A or possibly objects when A is a proper synthetic category. The first is of purely type-theoretical nature, coming from the fact we have a type of equalities $\mathrm{Id}_A(a, b)$, whose terms are the so-called homotopies between a and b. Semantically, these are homotopies between the functors $a, b : * \to A$ in the tribe, i.e. factorisations through the path object. The second, instead, is of category-theoretical nature, as the definition of internal morphisms intrinsically gave us a notion of isomorphism of terms. Since, in category theory, the correct way of identifying two objects (functors) is that of (natural) isomorphism, we want to dig into the distinction between the two identifications we have, and solve it. Indeed, the theory behaves well with respect to homotopy by construction. However, if we want it to represent category theory, the identification under which we want the theory to be invariant is that of isomorphism (for objects) and of natural isomorphism (for functors).

The next rule is the Rezk axiom, which overcomes these two issues by imposing an equivalence between homotopies and isomorphisms in a type. As a result, we will be able to interpret the path equalities, which are the internal morphisms in homotopy type theory, as *some* of the internal morphisms in a type, namely the invertible ones.

Remark 3.4.1.2. The groupoidal structure of types with respect to equalities will translate into saying that categories have a groupoidal structure with respect to their internal isomorphisms. As an upshot, this is something we want to be true: later on, we will interpret it by saying that, given a category, the subcategory of its isomorphisms, which will be the groupoid core, has a groupoidal structure.

3.4.2 Homotopy implies isomorphism

Before stating the axiom, let us study how we can compare the notion of homotopy and isomorphism to begin with. We show that we can construct a functor from Id_A to Iso_A over $A \times A$, which will associate to any equality of terms an isomorphism between them. As the reader might expect, since inducing functors out of Id_A is a simple process we met various times, this will rely on path induction.

Construction 3.4.2.1. Let $\Gamma \vdash A$ type. By Construction 3.4.0.4, we can construct an assignment $\Gamma, x : A \vdash \mathrm{ev}_{1_{(-)}}(x) : \mathrm{Iso}_A(x,x)$, which assigns to x the identity 1_x exhibited as isomorphism. By path induction, this induces a term $\Gamma, x : A, y : A, \alpha : \mathrm{Id}_A(x,y) \vdash \text{id-to-iso}(x,y,\alpha) : \mathrm{Iso}_A(x,y)$, that maps any equality between x and y into an isomorphism between them. In particular, it maps (x, x, refl_x) in the identity 1_x. In particular, this gives rise to a local functor $\lambda\alpha.\text{id-to-iso}(x,y,\alpha) : \mathrm{Id}_A(x,y) \to \mathrm{Iso}_A(x,y)$ over $\Gamma, x : A, y : A$. We can interpret this by saying that homotopy of terms implies isomorphisms.

However, our theory is agnostic about further relations between the two notions, in particular it cannot reverse the implication. This means that, a priori, homotopy is a stronger condition than isomorphism. This is coherent with the spirit of our theory, which was designed not to see any differences between two equal terms, and every concept should be compatible with it. Therefore, it is reasonable that isomorphism of terms is not finer than, and is actually implied by, homotopy.

3.4.3 The Rezk axiom

We are finally ready to impose the Rezk axiom, which aims to reverse the implication between homotopy and isomorphism. Even more, it realises this as an equivalence between types. Its goal, then, is to solve the issues we pointed out in Remark 3.4.1.1. The *Rezk axiom* is formulated as follows:

$$\frac{\Gamma \vdash A \text{ type}}{\vdash \text{is-}\simeq\text{-Equiv}_{\Gamma.A.A[p]}(\lambda\alpha.\text{id-to-iso}(q[p], q, \alpha))}.$$

This realises a local equivalence of types

$$\lambda\alpha.\text{id-to-iso}(x,y,\alpha) : \mathrm{Id}_A(x,y) \xrightarrow{\sim} \mathrm{Iso}_A(x,y)$$

over $\Gamma, x : A, y : A$, as we wished. In particular, it comes equipped with a quasi-inverse $\mathrm{Iso}_A(x,y) \to \mathrm{Id}_A(x,y)$ that realises an implication from isomorphism to homotopy. In particular, they are two equivalent conditions.

Remark 3.4.3.1. Let $\Gamma \vdash A$ type. We have a commutative square

$$\begin{array}{ccc} A & \xrightarrow{\sim} & \Sigma_{x:A,y:A}\, \mathrm{Id}_A(x,y) \\ {\scriptstyle 1_{(-)}}\downarrow & & \downarrow{\scriptstyle \Sigma_{x:A,y:A}\lambda\alpha.\text{id-to-iso}(x,y,\alpha)} \\ \mathrm{Iso}(A) & \xrightarrow{\sim} & \Sigma_{x:A,y:A}\, \mathrm{Iso}_A(x,y) \end{array}$$

By Proposition 2.6.3.2, the Rezk axiom realises the right functor as an equivalence. By Lemma 2.4.11.5, we conclude that $1_{(-)} : A \to \mathrm{Iso}(A)$ is an equivalence.

Remark 3.4.3.2. Let us go through Remark 3.4.1.1 to see how the Rezk axiom fixes the issues we mentioned.

(1) The ability to reinterpret equalities as isomorphisms, and in particular as some internal morphisms, makes us see that we did not completely overwrite homotopy type theory. More explicitly, what were defined to be morphisms in homotopy types in that setting are exactly the natural isomorphisms of functors into the types of our theory. Our theory largely enriches this structure by adding all the non-invertible morphisms, that also link objects (functors) that are not isomorphic (naturally isomorphic). In this sense, a common interpretation of equalities in types is that of "invertible morphisms". In order to do that, we had to introduce an interval type and define morphisms as functors out of it. This is not innatural as an extension of homotopy type theory, as in the theory of ∞-groupoids one could axiomatise the existence of an interval type as well (check [BGL+17]). However, its rules determine all functors out of it as equalities, whereas in our setting we only determine isomorphisms in this way via the Rezk axiom.

(2) The two ways we could use to identify two objects of a synthetic category, or more in general two functors into a type, now coincide, as (natural) isomorphism coincide with homotopy. This translates all the considerations we had about equalities to isomorphisms. In particular, because our theory was built to be invariant under equality, we inherit the invariance of the theory under isomorphisms between objects and functors. This perfectly suits our desires, as isomorphism is the proper identification we want to consider in category theory.

As a consequence of the Rezk axiom, we see that π_{Iso} always has a retraction. Because π_{Iso} is the isofibration associated to $\Gamma, x : A, y : A.f : x \to y \vdash \mathrm{isIso}_A(f)(x,y)$ type by Proposition 2.6.9.8 this means that $\mathrm{isIso}_A(f)(x,y)$ is inhabited if and only if it is contractible, which means that the isomorphism data of a morphisms are a contractible choice, resembling the fact that the two inverses, if they exist, are unique.

Proposition 3.4.3.3. Let $\Gamma \vdash A$ type. Then, $\pi_{\mathrm{Iso}} : \mathrm{Iso}(A) \to \mathrm{Fun}(\Delta^1, A)$ has a retraction, which is given by $\mathrm{Fun}(\Delta^1, A) \xrightarrow{\mathrm{ev}_i} A \xrightarrow{1_{(-)}} \mathrm{Iso}(A)$ for any $i \in \{0,1\}$.

Proof. By Construction 3.2.8.3, the composite $A \xrightarrow{1_{(-)}} \mathrm{Iso}_A \xrightarrow{\pi_{\mathrm{Iso}}} \mathrm{Fun}(\Delta^1, A) \xrightarrow{\mathrm{ev}_i} A$ is the identical assignment. By Remark 3.4.3.1, $1_{(-)} : A \to \mathrm{Iso}(A)$ is an equivalence, and thus $\mathrm{ev}_i \circ \pi_{\mathrm{Iso}} : \mathrm{Iso}(A) \to A$ is an inverse for it. This implies that the other composite $\mathrm{Iso}(A) \xrightarrow{\pi_{\mathrm{Iso}}} \mathrm{Fun}(\Delta^1, A) \xrightarrow{\mathrm{ev}_i} A \xrightarrow{1_{(-)}} \mathrm{Iso}(A)$ is homotopic to the identity. □

This concludes the discussion about isomorphisms and their role as equalities of terms. More in general, it concludes the section of this work dedicated to the internal structure of types.

3.5 Groupoids

It is now a good point to tell the theory what animae are, so that it will no longer be agnostic about it. In particular, we want to impose a characterisation of animae by means of a checkable property. Now, it was widely argued that the interpretation of what animae, those well-behaved contexts, should be in the theory of categories are groupoids, i.e. those categories with only invertible morphisms. Now that the theory has a notion of morphisms and morphism categories, we can specify what a groupoid, among all synthetic categories, is.

Definition 3.5.0.1 (Groupoid)**.** Let $\vdash \Gamma$ anima and let $\Gamma \vdash A$ type. We say that A is a *groupoid* if one of the following equivalent conditions hold:

(1) The type $\Gamma, x : A, y : A, f : x \to y \vdash \mathrm{isIso}_A(f)(x,y)$ is inhabited.

(2) The type $\Gamma, x : A, y : A, f : x \to y \vdash \mathrm{isIso}_A(f)(x,y)$ is contractible.

(3) The functor $1_{(-)} : A \to \mathrm{Fun}(\Delta^1, A)$ is an equivalence.

(4) The functor $\mathrm{ev}_0 : \mathrm{Fun}(\Delta^1, A) \to A$ is an equivalence.

(5) The functor $\mathrm{ev}_1 : \mathrm{Fun}(\Delta^1, A) \to A$ is an equivalence.

Lemma 3.5.0.2. The conditions above are all equivalent.

Proof. (1) $\Leftrightarrow$ (2): By Proposition 2.6.9.8 it suffices to show that the underlying isofibration of $\Gamma, x : A, y : A, f : x \to y \vdash \mathrm{isIso}_A(f)(x,y)$ has a retraction. This follows by Proposition 3.4.3.3.

(2) $\Leftrightarrow$ (3): The functor $1_{(-)} : A \to \mathrm{Fun}(\Delta^1, A)$ factors through the equivalence $1_{(-)} : A \xrightarrow{\sim} \mathrm{Iso}_A$ by Construction 3.4.0.4. In particular, it is an equivalence if and only if $\pi_{\mathrm{Iso}} \to \mathrm{Fun}(\Delta^1, A)$ is, if and only if the dependent type $\Gamma, x : A, y : A, f : \mathrm{fib}_{(\mathrm{ev}_0, \mathrm{ev}_1)}(x,y) \vdash \mathrm{isIso}_A(f)(x,y)$ type is contractible by Proposition 2.6.10.1, which shows (3) $\Leftrightarrow$ (2).

(4) $\Leftrightarrow$ (5) $\Leftrightarrow$ (3): By Remark 2.4.11.4, (4) and (5) are equivalent to (3) because ev_i is a retraction of $1_{(-)}$ by Construction 3.4.0.4 □

The next step is to construct a type that exhibits a type as a groupoid.

Definition 3.5.0.3 (The groupoid type)**.** Let $\vdash \Gamma$ anima and let $\Gamma \vdash A$ type. We define $\Gamma \vdash \mathrm{isGpd}(A) :\equiv \mathrm{Sec}_{x:A, y:A, f:x\to y} \mathrm{isIso}_A(f)(x,y)$ type.

Lemma 3.5.0.4. Groupoids are stable under equivalences.

Proof. Consider $\vdash \Gamma$ anima with $\Gamma \vdash A, B$ type, and consider a functor $A \to B$ of synthetic categories. If A is an anima, then we have a commutative square

$$\begin{array}{ccc} \mathrm{Fun}(\Delta^1, A) & \xrightarrow[f_*]{\sim} & \mathrm{Fun}(\Delta^1, B) \\ {\scriptstyle \mathrm{ev}_i}\downarrow & & \downarrow{\scriptstyle \mathrm{ev}_i} \\ A & \xrightarrow[f_*]{\sim} & B \end{array}$$

by construction, since $(f \circ g)(i) = \mathrm{ev}_f(\mathrm{ev}_g(i))$ by compatibility of evaluation with composition. In particular, by Lemma 2.4.11.5, A is a groupoid if and only if the left functor is an equivalence, if and only if the right functor is an equivalence, if and only if B is a groupoid. □

3.5.1 Animae and groupoids

We can finally state the rule that will force animae to be groupoids and our theory not to be agnostic anymore about it. Indeed, they should be a manifestation of the same concept under two different perspectives. The *anima-groupoid rules* are stated as

$$\frac{\vdash \Gamma \text{ anima} \quad \Gamma \vdash A \text{ type} \quad \vdash \Gamma.A \text{ anima}}{\Gamma \vdash \text{anima-groupoid}_A : \mathrm{isGpd}(A)} \qquad \frac{\vdash \Gamma \text{ anima} \quad \Gamma \vdash A \text{ type} \quad \Gamma \vdash a : \mathrm{isGpd}(A)}{\vdash \Gamma.A \text{ anima}}.$$

Note that the rule on the right finally gives us a criterion to prove that some synthetic category is in fact an anima. This gives us access to multiple key properties of animae. For instance, we can finally prove the following basic, yet fundamental, property of animae, namely their stability under equivalences.

Corollary 3.5.1.1. Animae are stable under equivalences.

Upshot. In the current state, the theory is not complete yet. However, the provided foundations are a good framework to develop the full theory of ∞-categories as types. In particular, future versions of this project will discuss the following aspects.

(1) Cocartesian fibrations. These will be some special dependent types whose fibres vary functorially. Morally, we want to see them as the fibres of a functor into a universe of categories. Their importance will be crucial in any stage of the theory.

(2) Subcategories and full subcategories. After introducing a synthetic definition of these notions, we need to make sure that the theory allows the construction of subcategories out of a collection of maps which is closed under compositions and identities.

(3) Localisations of categories. The ability of constructing categories via a localisation process is essential in category theory, and needs to be introduced synthetically. A particular attention will be devoted to the most important localisation: the fundamental groupoid of a synthetic category. This will mimic a left adjoint of the inclusion of groupoids into all categories, in the same way the groupoid core mimics the right adjoint.

(4) Functorial statements out of objectwise properties. A crucial component of category theory, which led us to the introduction of animae as distinguished contexts, is the idea that some properties can be checked objectwise, i.e. over groupoids rather than over all generalised terms. In homotopy type theory we do not see this aspect, and this is not due to some lack of "special" contexts; instead, it is due to the lack of "non-well-behaved" contexts, meaning that, in some sense, every statement is objectwise. On the other hand, in our setting, arbitrary generalised terms are much more complicated than they are in homotopy type theory, as a syntactic proof in a fixed context does not automatically lift to a syntactic proof in every context extension of it. It is crucial to teach our theory that, for some statements, this lifting is possible. For instance, we will deduce that natural isomorphisms are exactly those natural transformations that are objectwise isomorphisms.

(5) Fully faithful functors and essentially surjective functors. These concepts will allow us to prove a synthetic formulation of the fundamental theorem of category theory, making use of the methods introduced in (4). In particular, this will be a further instance of a global statement deduced from an objectwise one.

(6) The directed univalence axiom. This will translate the central idea of univalence from homotopy type theory into our setting. In particular, it will introduce universes of categories into the theory, and we will inherit the ability of internalising constructions on types as constructions about terms of the universe. These universes will be richer compared to those of homotopy type theory, not just for describing categories rather than groupoids, but also because they will not be mere statements. Indeed, they will fully describe the theory, by containing not just categories as terms, but all functors between them as well. Furthermore, within the theory, universes of groupoids will be constructed, and these will contain the whole theory of ∞-groupoids, overcoming something that homotopy type theoretical universes could not do, being them mere collections by nature. By considering their groupoid cores, i.e. their underlying statements, we will recover the universes from homotopy type theory and classical univalence.

Bibliography

[AG24] Carlo Angiuli and Daniel Gratzer. Principles of dependent type theory. *Online at https://carloangiuli. com/courses/b619-sp24/notes. pdf. Version*, pages 11–26, 2024.

[Awo18] Steve Awodey. Natural models of homotopy type theory. *Mathematical Structures in Computer Science*, 28(2):241–286, 2018.

[BGL+17] Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters. The hott library: a formalization of homotopy type theory in coq. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 164–172, 2017.

[Car86] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of pure and applied logic*, 32:209–243, 1986.

[CCD21] Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. In *Joachim Lambek: The Interplay of Mathematics, Logic, and Linguistics*, pages 135–180. Springer, 2021.

[CCNW25] Bastiaan Cnossen, Denis-Charles Cisinski, Kim Nguyen, and Tashi Walde. Formalization of higher categories. in-progress book project, 2025. Accessed via the project page of Bastiaan Cnossen at University of Regensburg.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(2):56–68, 1940.

[Esc19] Martín Hötzel Escardó. Introduction to univalent foundations of mathematics with agda. *arXiv preprint arXiv:1911.00580*, 2019.

[HS98] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. *Twenty-five years of constructive type theory (Venice, 1995)*, 36:83–111, 1998.

[Joy17] André Joyal. Notes on clans and tribes. *arXiv preprint arXiv:1710.10238*, 2017.

[KvR19] Nicolai Kraus and Jakob von Raumer. Path spaces of higher inductive types in homotopy type theory. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2019.

[ML75] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In *Studies in Logic and the Foundations of Mathematics*, volume 80, pages 73–118. Elsevier, 1975.

[MLS84] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory*, volume 9. Bibliopolis Naples, 1984.

[RS17] Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories. *arXiv preprint arXiv:1705.07442*, 2017.

[Rus08] Bertrand Russell. Mathematical logic as based on the theory of types. *American journal of mathematics*, 30(3):222–262, 1908.

[vdBdB21] Benno van den Berg and Martijn den Besten. Quadratic type checking for objective type theory. *arXiv e-prints*, pages arXiv–2102, 2021.