Pós-Graduação em Ciência da Computação

Arthur Freitas Ramos

Explicit Computational Paths in Type Theory

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2018

Arthur Freitas Ramos

Explicit Computational Paths in Type Theory

Esse trabalho foi apresentado à Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Teoria da Computação

Orientadora: Anjolina Grisi de Oliveira
Co-orientador: Ruy José Guerra Barreto de Queiroz

Recife
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

R175e Ramos, Arthur Freitas
Explicit computational paths in type theory / Arthur Freitas Ramos. – 2018.
145 f.: il., tab.

Orientadora: Anjolina Grisi de Oliveira.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2018.
Inclui referências.

1. Ciência da computação. 2. Teoria da computação. I. Oliveira, Anjolina Grisi de (orientador). II. Título.

004 CDD (23. ed.) UFPE- MEI 2018-140

Arthur Freitas Ramos

"Explicit Computational Paths in Type Theory"

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutora em Ciência da Computação.

Aprovado em: 17/08/2018.

To my parents, Neide & Valdez, who are my role models.
To my wife, Láis, for all the love and support.

ACKNOWLEDGEMENTS

This work would have not been possible without the help of many people. First, I can point the never-ending dedication of my parents, Neide and Valdez. Despite facing many hardships in life, they always valued my education, supporting and encouraging my studies. For this, I am eternally grateful. You are my role models and my heroes. I thank my wife, Laís, for her patience, love and support. Without her, it would be impossible to write this work. I also thank my advisor, Professor Ruy de Queiroz. He has been in this journey with me for 5 years, since the end of my undergraduate studies. I have learned a great deal from him and he has shown me true wisdom. Last but not least, I thank God for giving me patience, strength and resilience every time I asked.

ABSTRACT

The current work has three main objectives. The first one is the proposal of computational paths as a new entity of type theory. In this proposal, we point out the fact that computational paths should be seen as the syntax counterpart of the homotopical paths between terms of a type. We also propose a formalization of the identity type using computational paths. The second objective is the proposal of a mathematical structure for a type using computational paths. We show that using categorical semantics it is possible to induce a groupoid structure for a type and also a higher groupoid structure, using computational paths and a rewrite system. We use this groupoid structure to prove that computational paths also refutes the uniqueness of identity proofs. The last objective is to formulate and prove the main concepts and building blocks of homotopy type theory. We end this last objective with a proof of the isomorphism between the fundamental group of the circle and the group of the integers.

Key-words: Computational paths. Homotopy type theory. Identity type. Category theory. Term rewrite system. Uniqueness of identity proofs.

RESUMO

O presente trabalho tem três objetivos principais. O primeiro é propor caminhos computacionais como uma nova entidade da teoria dos tipos. Nessa proposta, indicamos que os caminhos computacionais podem ser vistos como uma contrapartida sintática dos caminhos homotópicos entre termos de um mesmo tipo. Também propomos uma formalização do tipo identidade usando caminhos computacionais. O segundo objetivo é propor uma estrutura matemática para um tipo usando os caminhos computacionais. Mostramos, usando semântica categórica, que é possível induzir uma estrutura de grupóide de alta ordem para um tipo, utilizando os caminhos computacionais e um sistema de reescrita. Usamos o modelo de grupóide para provar que os caminhos computacionais também refutam a unicidade de provas de identidade. O último objetivo é formular e provar os principais conceitos da teoria homotópica dos tipos utilizando caminhos. Finalizamos esse último objetivo com uma prova do isomorfismo entre o grupo fundamental do círculo e o grupo dos inteiros.

Palavras-chaves: Caminhos computacionais. Teoria homotópica dos tipos. Tipo identidade. Teoria das categorias. Sistema de reescrita de termos. Unicidade de provas de identidade.

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

$Id - E$	identity type elimination
$Id^{ext} - E$	extensional identity type elimination
$Id^{ext} - Eq$	extensional identity type equality
$Id^{ext} - F$	extensional identity type formation
$Id^{ext} - I$	extensional identity type introduction
$Id^{int} - F$	identity type formation
$Id^{int} - I$	identity type introduction
$\Pi - C$	dependent function computation
$\Pi - E$	dependent function elimination
$\Pi - F$	dependent function formation
$\Pi - I$	dependent function introduction
$\Sigma - E_1$	first dependent sum elimination
$\Sigma - E_2$	second dependent sum elimination
$\Sigma - F$	dependent sum formation
$\Sigma - I$	dependent sum introduction
$\mathbb{N} - I_0$	zero introduction
$\mathbb{N} - I_s$	successor induction
$\times - E_1$	first product elimination
$\times - E_2$	second product elimination
$\times - F$	product formation
$\times - I$	product introduction
UIP	Uniquiness of Identity Proofs
ZFC	Zermelo-Frankel-Choice

CONTENTS

1 INTRODUCTION

In this chapter, we introduce the main objective of this work. First, we show that the axiom system Zermelo-Frankel-Choice (ZFC), the main theory currently used as foundations of mathematics, is not constructive and thus, modeling it using computers is not practicable. After that, we introduce a theory that can be used as a foundation for computation and mathematics at the same time, homotopy type theory. Then, we introduce the type responsible for this connection, the identity type. Given the importance given to this type, it will be the main entity of this work. The main objective is to show an alternative way of formalizing the identity type, using an entity known as computational path. To do that, we propose a mathematical model to computational paths and prove many results of homotopy type theory using it.

1.1 FOUNDATIONS OF MATHEMATICS AND ZFC

One can easily say that the XIX century caused a mathematical revolution. It is mainly responsible for the modern way of mathematical thinking (AVIGAD, 2007). Before this century, mathematical practice was closely related to algorithmic processes. This century was marked by a sharp increase in abstraction in mathematics (AVIGAD, 2007). For example, it was in this century that non-euclidean geometries were proposed. First, the hyperbolic geometry by Nikolai Lobachevsky in 1832 and then elliptic geometry by Bernhard Riemman in 1851.

Given this high level of abstraction, a natural question has arisen. Where do mathematical objects come from? A mathematical object already exists and is awaiting for someone to discover it, or is it a creation of human minds? This question divided the mathematical community and was responsible for the creation of a whole new area of research, called philosophy of mathematics. In one hand, there were some mathematicians that defended that there is already an abstract and immutable universe containing all mathematical objects. Thus, a mathematician's job was to discover the objects of this work. This vision is known as Platonism and is currently the main vision of most mathematicians. On the other hand, there were some that believed that a mathematical object is created at the exact moment that it is conceived in the mind of a mathematician. Thus, the objective is constructed by the mathematician. This vision is known as constructivism. Since computers uses algorithmic process (programs) to obtain results, it is closely related to this constructive view of mathematics.

This work has been written based on the author's master's thesis (RAMOS, 2015), three journal papers, two published (RAMOS; QUEIROZ; OLIVEIRA, 2017; QUEIROZ; OLIVEIRA; RAMOS, 2016) and a third one still unpublished, and a conference talk (RAMOS, 2017) based on the still unpublished paper.

The way one does mathematics can depend on the view one chooses to follow. We are going to give an classic example of two proofs (DUMMETT, 1977), one that is not accepted by constructivists and one that is. Consider the following proposition:

Proposition 1.1. *There exist two irrational numbers a e b such that a^b is a rational number.*

First, we give a proof that is accepted by platonists but not by constructivists.

Proof. Consider the number $\sqrt{2}$. It is clearly irrational. Thus, $\sqrt{2}^{\sqrt{2}}$ is irrational or rational. If it is rational, then we conclude the proof. If not $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$ (DUMMETT, 1977), also conclude the proof, since $\sqrt{2}^{\sqrt{2}}$ was considered to be irrational. □

This proof shows that one of the two cases solves the problem, but not determine which one does. Thus, this proof is not constructive. But, if one accepts the platonic point of view, this proof is completely acceptable. Consider this alternate proof:

Proof. Consider numbers $\sqrt{2}$ and $\log_{\sqrt{2}} 3$. Since both are irrationals and $\sqrt{2}^{\log_{\sqrt{2}} 3} = 3$, then our proof is complete. □

One can notice that we directly show two irrational numbers that can be used to construct a rational one. Thus, we consider this proof as a constructive one.

Currently, most mathematicians are platonists, thus they accept proofs like the first one. One of the factors that may be responsible for this is the fact that ZFC is widely accepted as a foundation for mathematics. ZFC, a abbreviation for Zermelo-Frankel with choice is an axiomatic theory proposed in 1908 and improved in 1920-1940 (HORSTEN, 2015). that gave a mathematical formalization for set theory. It was born out of necessity, since Russel noticed that naive set theory led to a paradox.

In 1901, the philosopher and mathematician Bertrand Russell discovered a paradox that disrupted the mathematical community. In naive set theory, one can construct the following set: $R = \{x | x \notin x\}$. Russel noticed that it led to a paradox (IRVINE, 2014), if one thinks in the following way: If R is not a member of itself, then by the definition of R it should be a member of itself. But if R is a member of itself, it would contradict directly its own definition. Thus, R cannot exist. Thus, ZFC was proposed to deal with problems like this one.

The main problem is that ZFC is not constructive. Using the axiom of choice, one can show a theorem that states that every set can be well-ordered(HRBACEK; JECH, 1999). Nevertheless, one is not able to construct directly a well-ordering for the reals. Therefore, this clearly shows that ZFC admits results without an explicit construction.

1.2 THE IDENTITY TYPE

In this section, we show that it is possible to connect computer science and mathematics using the identity type. The identity type is arguably the most important entity of a theory known as type theory. Type theory is a construct theory proposed by the mathematician Martin-Löf in 1971(MARTIN-LÖF, 1975; MARTIN-LÖF, 1982; MARTIN-LÖF, 1984). The fundamental concept of this theory is the concept of type. A type is defined by a description on how to construct and eliminate it (BRIDGES; PALMGREN, 2013). We show this theory in detail in **chapter 2**

Given any type A, we write $a : A$ to indicate that a is a term of type A. The identity type captures the following idea: given any terms $a : A$ and $b : A$ and a proof p that establishes that $a = b$, then one can say that p is term of the identity type $Id_A(a, b)$, i.e., the terms of type $Id_A(a, b)$ are proofs that establish that $a = b$. That way, the identity type gives two main facts: that a is equal to b and why this equality holds.

A groundbreaking result turned the identity type in one of the most studied topics of type theory: the direct relation between the identity type and homotopy type theory (VOEVODSKY, 2014). This came from the fact that one can semantically interpret a type A as a topological space, the objects $a, b : A$ are seen as point of this space and a term $p : Id_A(a, b)$ is seen as a homotopical path between points a and b(Univalent Foundations Program, 2013). This interpretation yielded groundbreaking results. One of the most important result is the fact that it connected homotopy theory with type theory, giving rise to homotopy type theory. Moreover, it raised the possibility of type theory as a foundation of mathematics. Also, since type theory is naturally constructive, it can be used as a foundation for computation.

The distinguished mathematician Vladimir Voevodsky made clear the advantages of using a constructive theory as a foundation for mathematics (VOEVODSKY, 2014). Voevodsky, which is a Fields Medal winner, proposes that the increasingly abstraction of mathematics makes the mathematician prone to committing errors when doing mathematics. To illustrate that, he uses his past experiences: he discovered in 2013 that a paper published by him in 1989 contained errors (VOEVODSKY, 2014). Thus, he argues that if one did mathematics using the help of an automatic theorem checker, errors like this one would not occur anymore. Thus, one practical advantage of homotopy type theory is the fact that all proofs can be modeled and checked by computers.

One of the disadvantages of the identity type is that it can be hard to understand. It is based on the fact that the only canonical proof of equality is the reflexivity, i.e., given a term $a : A$, we have a canonical proof $r(a) : a = a$. This leads to a complex elimination rule that gives rise to an induction known as path induction (Univalent Foundations Program, 2013). Although beautifully defined, we have noticed that proofs that uses the identity type can be sometimes a little too complex. The elimination rule of the intensional identity type encapsulates lots of information, sometimes making too troublesome the process of finding the reason that builds the correct type.

Inspired by the path-based approach of the homotopy interpretation, we believe that a similar approach can be used to define the identity type in type theory. Our main idea is to add computational paths to the formal syntax of type theory. That way, this new entity would be the syntax counterpart of semantical paths. In the sequel, we shall define formally the concept of a computational path. The main idea, i.e. proofs of equality statements as (reversible) sequences of rewrites, is not new, as it goes back to a paper entitled "Equality in labeled deductive systems and the functional interpretation of propositional equality ", presented in December 1993 at the *9th Amsterdam Colloquium*, and published in the proceedings in 1994(QUEIROZ; GABBAY, 1994).

One of the most interesting aspects of the identity type is the fact that it can be used to construct higher structures. This is a rather natural consequence of the fact that it is possible to construct higher identities. For any $a, b : A$, we have type $Id_A(a, b)$. If this type is inhabited by any $p, q : Id_A(a, b)$, then we have type $Id_{Id_A(a,b)}(p, q)$. If the latter type is inhabited, we have a higher equality between p and q(HARPER, 2012). This concept is also present in computational paths. One can show the equality between two computational paths s and t by constructing a third one between s and t. We show in the sequel a system of rules used to establish equalities between computational paths(OLIVEIRA, 1995). Then, we show that these higher equalities go up to the infinity, forming a ∞-globular-set. We also show that computational paths naturally induce a structure known as groupoid. We also go a step further, showing that computational paths are capable of inducing a higher groupoid structure.

After constructing this mathematical model, we also need to show that it is possible to use computational paths to construct concepts of homotopy type theory. To do that, we investigate well established properties and concepts of the foundations of homotopy type theory. We are interested in the ones connected to the identity type. Our main objective is to show that these properties and theorems are valid in our approach for the identity type based on computational paths. In this sense, we show that one can use computational paths to define and develop concepts of homotopy type theory. We end this work with a proof using computational paths that the fundamental group of a circle is isomorphic to the integers.

1.3 OBJECTIVES

In the previous sections, we pointed out the importance of type theory to mathematics and computation. We have also said that the identity type is one of the main concepts of this theory and perhaps the most interesting one. With that in mind, we have said that we want to develop an alternative approach to the identity type, based on the fact that an equality proof can be seen as a sequence of rewrites between two computational objects. In this sense, this work has 3 main objectives.

The first objective is to formally introduce to type theory an entity known as computational paths and, based on this entity, propose a formulation for the identity type. To do this, we revisit the main concepts of type theory and some important concepts of λ-calculus. Then, we introduce the notion of computational paths. We present this new entity in the traditional way of defining a type in type theory: we define formation, introduction, elimination and computation rules. We also establish a rewrite system that will work as an algebra of computational paths.

The second objective is to give mathematical meaning to the structure of computational paths. We do this using categorical semantics. Specifically, we are talking about the groupoid structure of a type. We use our computational path entity and the associated rewrite system to show that every type has an induced groupoid associated to it. We also go a step further, showing that it is possible to induce higher structures such as bicategories.

Our final objective is to establish the connection between computational paths and homotopy type theory. We use the theory developed in this work to show that many concepts and proofs of homotopy type theory can be achieved without the use of path-induction, using computational paths instead. In this sense, we show that our approach is capable of producing the main building blocks of homotopy theory. We end this objective with an important proof: we use computational paths to show that the fundamental groupoid of the circle is isomorphic to the group of the integers.

Thus, to achieve the first objective, we have mainly used a computational approach. The second one is mainly a mathematical approach. The third objective is a mix between the previous two.

1.4 STRUCTURE

This first chapter was meant as an introduction for this work. We have highlighted the importance of the importance of the identity type in type theory. We have also exposed the main objectives of this work.

The second chapter will be focused on type theory. In this chapter, we will introduce the basic concepts and the difference of definitional and propositional equality. We also show the classic approach for the identity type, showing the formation, introduction, elimination and computation rules. We also show how to use this approach in practice, showing how some basic types can be constructed.

The third chapter will be focused on category theory. We show the basic concepts of this theory and also some concepts of higher category. This chapter is important to understand the results of chapter 4.

The fourth chapter is of great importance, since it is responsible for objectives 1 and 2. In this chapter, we introduce the concept of computational paths and establishes the connection with the identity type. Moreover, we introduce an extremely important rewrite system, responsible for establishing the equalities between computational paths. Thus, we use this system and categorical semantics to show that computational paths induce a mathematical structure known as groupoid. We finish this chapter establishing one fundamental result, the refutation of the uniqueness of identity proofs using computational paths.

The fifth chapter is responsible for objective 3. In this chapter. we use the theory developed in chapter 4 to define, construct and prove the main building blocks of homotopy theory. In the process, we prove dozens of lemmas and theorems. We end this chapter with one of the most classic proofs of algebraic topology: we use computational paths and the rewrite system to show that the fundamental group of the circle is isomorphic to the group of the integers.

The sixth chapter is the conclusion of this work. It is a short chapter in which we review and point out all results obtained in this work.

2 TYPE THEORY

Among the theories used as a foundation for computation, one of the most famous and that is closely related to a general purpose computer are the theory of Turing machines, proposed by Alan Turing in 1937. Turing machines had great success in giving a mathematical formalization for the concept of computer. Moreover, it is essential in the investigation of the complexity of algorithms and the limits of computations, proving the existence of problems that are not bound to be solved by computers (BARKER-PLUMMER, 2013).

Even before Turing's groundbreaking work, one theory that has the same power of Turing's machine had already been proposed by Alonso Church. Known as λ-calculus, it can be seen as a very basic programming language, with only two operations: function abstraction and function application. Despite this fact, λ-calculus plays a very important role in computation, logic and mathematics.

Initially, λ-calculus was proposed to simplify the notation of functions (ALAMA, 2015). Take $f(x) = x\check{s}$ as a very simple example. How do we calculate the value of $f(3)$? It is very simple. We just need to plug the value 3 in place of x in the expression x^2, resulting in $3^2 = 9$. The idea that x^2 is an expression that awaits one term to be plugged into it is given by the abstraction operation. We denote this fact by the expression $\lambda.x^2$. Thus, $f = \lambda.x^2$. Function is the reverse process. Given $\lambda.x^2$, one can use an application to substitute the value of input x for an arbitrary value y. When we apply y, we wound up with $(\lambda.x^2).y$. Then, we say that this term β-reduce to $[y/x]x^2 = y^2$. The notation $[y/x]$ indicates that all occurrences of x in the term will be substituted for y. We are also going to use this notation in type theory. Therefore, we had in our previous example the following case: $[3/x]x^2 = 3^2 = 9$.

One of the most counterintuitive facts is that it is possible to prove that this theory with only two simple operations is powerful enough to formalize all computable functions (HINDLEY; SELDIN, 2008). That way, λ-calculus, together with the theory of Turing's machines are the two main theories used to formalize computability and that serve as a foundation for computation.

Type theory is a theory that is in many aspects similar to λ-calculus. Despite this, it was originally proposed with for a different purpose. Instead of being a foundation for computation, the main objective of type theory was to function as a foundation for constructive mathematics. Specifically, an attempt based on Erret Bishop's constructivism. This work will focus on a specific kind of type theory: Martin-Löf's intensional type the-

Parts of this chapter are based on previous research done by the author and appears in his Master's Thesis (RAMOS, 2015) ory, originally proposed in 1971. Since this theory is intrinsically constructive, it has also been used as a foundation for computation. This fact can be seen in practice: Coq, Agda and Epigram are all examples of programming languages based on the concepts of type theory.

In the next section, we describe and develop the main basic concepts and types of Martin-Löf's intensional type theory. The correct understanding of this concepts will be essential to the development of latter sections of this work.

2.1 BASIC CONCEPTS

The main and most basic concept of type theory is the concept of type. A type must be understood as a fundamental concept that work as the basis of type theory. One can think of a type as similar to the concept of set in set theory. In fact, types are more powerful in type theory than sets in set theory (Univalent Foundations Program, 2013). The reason for that is that one cannot derive all results of ZFC using sets only. The use of first-order logic is necessary to state the axioms. In ZFC, the concept of set and propositions are completely distinct. In contrast, we are going to see that propositions can be seen as types in type theory.

Let A be a type and a a term of type A. We say that A is inhabited by a and denote this by $a : A$. This is the basic judgment of type theory. Another judgment is $a = b : A$, indicating that a and b are intensionally equal elements of type A.

The syntax of type theory is independent of the nature of type A. A can be a set, a proposition or even a topological space. Nevertheless, for all those cases, if a has type A, then we always denote $a : A$. What differs is the semantical interpretation of $a : A$ (Univalent Foundations Program, 2013). If A is a set, then a can be understood as an element of set A. If it is a space, then a is a point of space A. A very interesting case is when A is a proposition. In that case, a can be seen as a witness of the veracity of A. In other words, a is a proof that A is true. One should keep in mind that it is only a semantical interpretation. Even when A is a set and $a : A$, one cannot write $a \in A$.

In the next subsection, we investigate the existence of two different kinds of equalities in type theory.

2.1.1 Definitional Equality vs Propositional Equality

One of the main things that one should grasp about type theory is the existence and the difference between two types of equality. The first kind of equality is originated by the fact that two terms are equal simply by definition, without the need of some result establishing the equality. This equality is called definitional equality. The other kind of equality is when equality is seen as a type. In that case, it is called propositional equality.

Between the two kinds of equality, the most interesting one is the propositional, since it originates the identity type. Given terms $a, b : A$, one could derive the judgment $a = b : A$. Thus, taken this judgment as a starting point, one can naturally conceive a premise establishing the equality between terms a and b of type A. This equality gives rise to a proposition, i.e., a type known as identity type, usually written as $Id_A(a, b)$. Using the aforementioned interpretation, $Id_A(a, b)$ should be understood as a proposition that establishes the equality between a and b. We say that a is propositionally equal to b. If we have a proof p of this equality, then $p : Id_A(a, b)$. That way, p is a witness of $Id_A(a, b)$, establishing the veracity of the proposition that a is propositionally equal to b.

The definitional equality is much simpler than the propositional one. In this equality, the existence of a witness that establishes the equality is not necessary. We denote definitional equality using the symbol $\equiv$. For example, take the function $f : \mathbb{N} \to \mathbb{N}$ as $f(x) \equiv x^2$ (Univalent Foundations Program, 2013). Suppose one wants to compute $f(3)$. From the definition of f, we can conclude that $f(3) \equiv 3^2$. Nevertheless, one cannot conclude by definition that $f(3) = 9$. To conclude that, one needs to conceive a proof p such that $p : Id_{\mathbb{N}}(3, 9)$. Thus, one could establish that 3 is propositionally equal to 9. To denote that two terms of type A are definitionally equal we use the notation $a \equiv b : A$.

To better grasp the difference between these two kinds of equalities, consider the following example (HARPER, 2012):

Example 2.1. Define addition as following:

$$add : \begin{cases} a + 0 = a \\ a + s(b) = s(a + b) \end{cases}$$

Consider $s(x)$ as the successor function. We can analyze the following cases:

- $2 + 2 \equiv 4 : \mathbb{N}$ (True): First, we have $2 \equiv s(1)$. From the definition, we have $2 + s(1) \equiv s(2 + 1) \equiv s(s(2 + 0)) \equiv s(s(2)) \equiv s(3) \equiv 4$.

- $0 + x \equiv x : \mathbb{N}$ (False): Since by definition we only have that $x + 0 \equiv x$, we cannot conclude that $0 + x \equiv x$. One can easily prove that, but since we need a proof to establish this, this is a propositional equality.

- $s(x) \equiv (1 + x) : \mathbb{N}$ (False): We know that $s(x) \equiv (1 + x)$, but we need to prove the commutativity of addition to conclude that $x + 1 = 1 + x$. Thus, this equality is not definitional.

- $(x + y) \equiv (y + x) : \mathbb{N}$ (False): Same case as before.

This example shows the subtle differences between definitional equality and propositional equality. Every time one needs an external evidence, it is a propositional equality. If the result follows directly from a definition, then it is a definitional equality. That way, even a simple property as commutativity of addition is established by a propositional equality.

2.1.2 Type Families

In this subsection, our objective is to show that it is possible to simulate the concept of predicate in type theory. To understand that, it is useful to consider a simple example. Given $x : \mathbb{N}$, one wants to decide if x is even. The problem is that if you consider a type that $even(x)$, that is inhabited if x is even, the type would depend of the value of x. Since for every different value of x one creates a new type $even(x)$, we say that it is a type family indexed by the values of x, i.e., indexed by the natural numbers. Thus, in this type family, we would have $0 : even(0)$. Since 1 is odd, $even(1)$ is not inhabited. Thus, every $even(2n)$ is inhabited and every $even(2n+1)$ is not.

As one can see, the concept of type families is equivalent to the concept of predicate. One important property of type families is the following: (HARPER, 2012):

$$\frac{m : A \qquad \begin{matrix}[x : A] \\ B(x) \text{ type}\end{matrix}}{[m/x]B(x) \text{ type}}$$

Given a type family $B(x)$ indexed by a type A, then if one has a term $m : A$ then one can obtain the type $[m/x]B(x) \equiv B(m)$. We have another important property: (HARPER, 2012):

$$\textit{functionality} \qquad \frac{m \equiv n : A \qquad \begin{matrix}[x : A] \\ B(x) \text{ type}\end{matrix}}{[m/x]B(x) \equiv [n/x]B(x) \text{ type}}$$

This property establishes that if we have a type family indexed by A and two terms $m \equiv n : A$, then it is intuitive to think that $B(m) \equiv B(n)$. For example, we know that $2+2 \equiv 1+3 \equiv 4 : \mathbb{N}$. Thus, by functionality, we have $even(1+3) \equiv even(2+2) \equiv even(4)$.

2.1.3 Function Type

In this subsection we are going to introduce the type that represents functions. Given any two types A, B, one can build the function type $A \to B$. The first type, A, is called domain of the function and the type B is called codomain. Of course it is also possible to construct this type using only A, obtaining $A \to A$. In that case, A is at the same time the domain and the codomain of the function. In set theory, a function is built as a relation, i.e., pairs of input and output, whereas in type theory a function is understood as a primitive element of the theory (Univalent Foundations Program, 2013). Since it is a primitive element, a function must be understood and interpreted by the way of how it is constructed and utilized.

In a function type $f : A \to B$, one can apply f in a term $a : A$ to obtain a term $b : B$. In that case, we write the usual notation $f(a) = b : B$. The notation $fa = b : B$ can also be used sometimes. This behavior defines how the function type can be used. We also need to define how a function can be constructed.

There are two main ways of constructing $A \to B$ (Univalent Foundations Program, 2013). The first one is to use directly the definitional equality of $f(x)$. In that case, $A \to B$ is constructed from an expression $f(x) \equiv \theta$. The necessary conditions are that x must a term of type A, i.e., $x : A$ and that $\theta : B$. The second way is defining a function using λ notation. In that case, a function is defined using an abstraction λx, obtaining $(\lambda(x : A).\theta) : A \to B$. Using this notation, a function application is similar to how is done in λ-calculus: $(\lambda x.\theta)(a) \equiv [a/x]\theta$.

As a simple example, one can take a $f : \mathbb{N} \to \mathbb{N}$ that has a input $x : \mathbb{N}$ and gives $x + 2 : \mathbb{N}$ as output. One can construct this function using the two aforementioned approaches. First, take the one using definitional equality. One obtain $f(x) \equiv x + 2 : \mathbb{N} \to \mathbb{N}$, with $f(a) \equiv a + 2, a : \mathbb{N}$. The other way is using λ notation. One can define $f = \lambda x.(x + 2) : \mathbb{N} \to \mathbb{N}$. If one applies $a : \mathbb{N}$, one winds up with the same result: $(\lambda x.(x + 2))(a) \equiv [a/x](x + 2) \equiv a + 2$.

It also should be possible to define multivariable functions. In the previous examples, the function received only one input. Nonetheless, it is possible to define functions receiving 2 or more inputs. Take the case of 2 inputs as example: $f(x, y : \mathbb{N}) \equiv \theta : N$. This function has type $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. We have not formally defined the symbol $\times$ yet, but take it as the usual cartesian product, which can also be defined in the framework of type theory. Using λ notation, one defines $f = (\lambda x.\lambda y.\theta) : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. To define functions with more than 2 variables, one can use an analogous process.

One important concept of multivariable functions is the existence of a process known as currying. Currying is a process to transform a function in n variables into n functions that receives only one variable, such that the output of those functions is another function (Univalent Foundations Program, 2013). Let's explain this process in a function of two variables, since one can analogously extend this process to an arbitrary number of variables. Given a function $f(a, b) \equiv c$, it is possible to transform f into a function that receives only a and has as output another function that receives only b, giving c as the final output. Doing this, the type $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is transformed in an equivalent type $f : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$. Since parenthesis is right-associative, one can write this type as $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$.

To a better understanding of this process, let's take an example. Let's take a function $f(x, y) \equiv x + y + 5$. Consider the specific case of an application $x = 3$ and $y = 5$. Without using currying, one could do $f(3, 5) \equiv 3 + 5 + 5 \equiv 8 + 5 \equiv 13$. With currying, $f(x)$ outputs $g : \mathbb{N} \to \mathbb{N}$ that receives y as input. In that case, one would have $f(3) \equiv \lambda y.(3 + y + 5)$.

This function receives $y = 5 : f(3)(5) \equiv 3+5+5 \equiv 13$. As one could see, the final results of these two approaches is exactly the same.

2.1.4 Dependent Function Type(Π)

Sometimes the type of the output of a function depends on the value of the input. Using only the function type, it is impossible to do this kind of construction. That way, we need to introduce a new function type called dependent function type. To better understand that, consider a type $Seq(x)$ indexed by the natural numbers. A term of type $Seq(x)$ can be seen as a sequence of naturals from 0 to x. Now, a problem arises when we try to construct a function that receives $n : \mathbb{N}$ as input and $\{0...n\} : Seq(n)$ as output. Since the output depends on the value of the input, the previous function type cannot define a function for this case. To indicate that a type is a dependent function, one can use the following notation: $\Pi_{(x:A)}B(x)$, where x is the input and $B(x)$ a type family indexed by the type A. In the previous example of $Seq(x)$, one can construct the dependent function $\Pi_{(x:\mathbb{N})}Seq(x)$. We also have the following inference rules (HARPER, 2012).

$$\frac{A \text{ type} \qquad \begin{matrix}[x:A] \\ B(x) \text{ type}\end{matrix}}{\Pi_{(x:A)}B(x) \text{ type}} \,(\Pi - F)$$

$$\frac{\begin{matrix}[x:A] \\ m(x) : B(x)\end{matrix}}{\lambda(x:A).m(x) : \Pi_{(x:A)}B(x)} \,(\Pi - I)$$

In the case of dependent function formation $(\Pi - F)$, given a type A and a type family $B(x)$ indexed by this type, it is formed the type $\Pi_{(x:A)}B(x)$. The case of dependent function introduction $(\Pi - I)$ is also very simple. Given a term $m(x) : B(x)$, with $x : A$, it is possible to obtain a dependent function that receives type x and gives a term of type $m(x)$ as output. The elimination rule is the following:

$$\frac{n : A \qquad m : \Pi_{(x:A)}B(x)}{m.n : [n/x]B(x)} \,(\Pi - E)$$

The dependent function elimination $(\Pi - E)$ boils down to an application of a element n in the dependent function. The output is very similar to the non-dependent case, the sole difference is the fact that the type of the output depends on the value of the input. We also have a computation rule:

$$\frac{n : A \qquad \begin{matrix}[x:A] \\ m(x) : B(x)\end{matrix}}{(\lambda(x:A).m).n = [n/x]m(x) : [n/x]B(x)} \,(\Pi - C)$$

If one looks closely, the dependent function computation $(\Pi - C)$ establishes function application the exactly way that was defined in non-dependent functions. The sole difference is that the output depends on x. With that, we have all necessary rules to define and compute dependent functions. Also, when working directly with types that represent propositions, one should can semantically interpret a type $\Pi_{(x:A)}B(x)$ as $\forall(x : A)B(x)$.

2.1.5 Product type ($\times$)

Previously we have mentioned the existence of multivariable functions, that receives two or more inputs. But how can we represent a function that, for instance, receives two natural numbers? In set theory, one could use the cross product to do that. That way, this function receives $\mathbb{N} \times \mathbb{N}$ as input. In this sense, it is essential to define cross product in the framework of type theory.

Intuitively, for any types A and B one should be able to construct the type $A \times B$. A term of $A \times B$ is a pair $(a, b) : A \times B$, with $a : A$ and $b : B$. This is formalized in the following rules (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\frac{A \text{ type} \qquad B \text{ type}}{A \times B \text{ type}}\,(\times - F)$$

$$\frac{a : A \qquad b : B}{(a, b) : A \times B}\,(\times - I)$$

If one looks closely, if A and B are propositions, $A \times B$ can be interpreted as $A \wedge B$. That way, one should be able to infer A and B from $A \times B$. To do that, we introduce functions FST and SND (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\frac{\langle a, b\rangle : A \times B}{FST((a, b)) : A}\,(\times - E_1)$$

$$\frac{(a, b) : A \times B}{SND((a, b)) : B}\,(\times - E_2)$$

Using FST and SND functions, we want to be able to retrieve the first and second elements of a pair $\langle a, b\rangle$ respectively. Thus, we have the following computation rules (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\frac{(a, b) : A \times B}{FST((a, b)) : A}\,\times - E_1 \quad \rhd_\beta \quad a : A$$

$$\frac{(a, b) : A \times B}{SND((a, b)) : B}\,\times - E_2 \quad \rhd_\beta \quad b : A$$

The above rules are called $\times$-reductions. We also have another computation rule called $\times$-induction (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\cfrac{\cfrac{c : A \times B}{FST(c) : A}\times - E_1 \qquad \cfrac{c : A \times B}{SND(c) : B}\times - E_2}{(FST(c), SND(c)) : A \times B}\times - I \quad \rhd_\eta \quad c : A \times B$$

2.1.6 Coproduct $(+)$

In the previous subsection we have introduced the product type. We saw that $\times$ can be seen semantically as a cross product in set theory and the connector $\wedge$ when dealing with propositions. Thus, it is natural to think that the dual of the product should also be able to be formalized in type theory. That is what we do in this section. In set theory, the coproduct represents the disjoint union. If one is working with propositions, the coproduct can be semantically interpreted as the connector $\vee$.

In the product, we have seen that from a term of $A \times B$ one should be able to obtain terms of A and B by separate processes. Since the coproduct work as the dual of the product, then it is natural to think that one should be able to obtain a term of $A + B$ from a term of A or a term of B. That is exactly what happens (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}}(+ - F)$$

$$\frac{a : A}{inl(a) : A + B}(+ - I_1) \qquad\qquad \frac{b : B}{inr(b) : A + B}(+ - I_2)$$

The elimination rule follows directly from the fact that if one from $a : A$ is able to construct a term $r(a) : C$ and from $b : B$ we also construct $l(b) : C$, then we should be able to construct one term of type C directly from a term $c : A + B$ (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\frac{c : A + B \qquad \begin{matrix}[a : A] \\ r(a) : C\end{matrix} \qquad \begin{matrix}[b : B] \\ l(b) : C\end{matrix}}{CASE(c, \acute{a}r(a), \acute{b}l(b)) : C}(+ - E)$$

One should see $d(a)$ and $e(b)$ as functional expressions dependent on a and b respectively. One should also notice the use of '$\acute{}$' in $\acute{a}$ and $\acute{b}$. One should see '$\acute{}$' as an abstractor that binds the occurrences of the variable a and b both introduced in the local assumptions. We have the following reduction rules (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\cfrac{\cfrac{c : A}{inl(c) : A + B} \qquad \begin{matrix}[a : A] \\ r(a) : C\end{matrix} \qquad \begin{matrix}[b : B] \\ l(b) : C\end{matrix}}{CASE(inl(c), \acute{a}r(a), \acute{b}l(b)) : C}(+ - E) \quad \rhd_\beta \quad \begin{matrix}[c : A] \\ [c/x]r(a) : C\end{matrix}$$

$$\dfrac{\dfrac{c : B}{inr(c) : A + B} \qquad \begin{matrix}[a : A] \\ r(a) : C\end{matrix} \qquad \begin{matrix}[b : B] \\ l(b) : C\end{matrix}}{CASE(inr(c), \acute{a}r(a), \acute{b}l(b)) : C}\,(+ - E) \quad \rhd_\beta \quad \begin{matrix}[c : B] \\ [c/x]l(b) : C\end{matrix}$$

We also have an induction rule (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\dfrac{c : A + B \qquad \dfrac{[a : A]}{inl(a) : A + B}\, + - I_1 \qquad \dfrac{[b : B]}{inr(y) : A + B}\, + - I_2}{CASE(c, (\acute{a})inl(a), \acute{b}inr(b)) : A + B}\, + - E \quad \rhd_\eta \quad c : A + B$$

2.1.7 Dependent Pair Type (Σ)

When working with functions, it is sometimes useful to work with ordered pairs (a, b) such that a is a input for the function and b is the respective output. But what happens if one is working with a dependent function? In that case, the type of the output b would be bounded to the value of the input a. To represent this, one needs to use a type called dependent pair. The notation for this type is $\Sigma_{(x:A)}B(x)$ and it has the following rules(HARPER, 2012):

$$\dfrac{A \text{ type} \qquad \begin{matrix}[x : A] \\ B(x) \text{ type}\end{matrix}}{\Sigma_{(x:A)}B(x) \text{ type}}\,(\Sigma - F)$$

$$\dfrac{m : A \qquad n : B(m)}{\langle m, n\rangle : \Sigma_{(x:A)}B(x)}\,(\Sigma - I)$$

The dependent sum formation ($\Sigma - F$) is similar to $\Pi - F$. For dependent sum introduction ($\Sigma - I$), an element $m : A$ and an element $n : B(m)$ are enough to introduce a dependent pair $\langle m, n\rangle : \Sigma_{(x:A)}B(x)$. We have two elimination rules:

$$\dfrac{m : \Sigma_{(x:A)}B(x)}{\pi_1(m) : A}\,(\Sigma - E_1)$$

$$\dfrac{m : \Sigma_{(x:A)}B(x)}{\pi_2(m) : [\pi_1(m)/x]B(x)}\,(\Sigma - E_2)$$

In traditional type theory, the dependent pair has two eliminations. This is explained by the fact that since the pair is composed by two terms, it is possible to eliminate Σ obtaining the first or the second term. From a dependent pair, one can use the first dependent sum elimination ($\Sigma - E_1$) and extract the first element. From the second dependent sum elimination ($\Sigma - E_2$) one can extract the second element, which type depends on the value of the first element. Thus, we have two computation rules:

$$\pi_1(\langle m, n\rangle) = m \text{ e } \pi_2(\langle m, n\rangle) = n.$$

The operators π_1 and π_2 extract the necessary information from the dependent pair. One obtains m and n respectively. The dependent pair should be understood as the dual of the dependent product.

2.1.8 Alternative Formulation for the Dependent Pair

If one is working within the paradigm of *formulae-as-types*(HOWARD, 1980), the dependent product should be semantically interpreted as the $\exists$ quantifier. Nevertheless, it has been pointed out by(QUEIROZ; GABBAY, 1995) that the previous formulation for the dependent pair leads to problems. The main issue is the fact that the previous formulation does not 'hide' the witness in an existential formula. One can apply directly π_1 to a dependent pair, obtaining the witness and ignoring the fact that it should be hidden. Thus,(QUEIROZ; GABBAY, 1995) argues that this 'availability' does not match with the true spirit of indefiniteness of the existential qualifier. To better see this, one could consider the fact that the duality between the existential and universal quantifiers are the first-order counterpart of the duality between the disjunction and the conjunction. In the language of type theory, when one talks about conjunction one is really talking about $\times$-product. In the $\times$-product, the terms of a pair are readily available:

$$\dfrac{\dfrac{a : A \qquad b : B}{(a, b) : A \times B}\times - I}{FST((a, b)) : A}\times - E_1 \qquad \dfrac{\dfrac{a : A \qquad b : B}{(a, b) : A \times B}\times - I}{SND((a, b)) : B}\times - E_2$$

In the case of the disjunction, i.e., the coproduct in type theory, once one of the disjuncts is used to construct a term of the coproduct, it becomes 'hidden', the elimination rule has to proceed by Skolem-like introductions of new local assumptions(QUEIROZ; GABBAY, 1995):

$$\dfrac{\dfrac{a : A}{inl(a) : A + B} + - I \qquad \begin{matrix}[x : A] \\ r(x) : C\end{matrix} \qquad \begin{matrix}[y : B] \\ l(y) : C\end{matrix}}{CASE(inl(a), \acute{x}r(x), \acute{y}l(y))} + - E_1$$

$$\dfrac{\dfrac{b : B}{inr(b) : A + B} + - I \qquad \begin{matrix}[x : A] \\ r(x) : C\end{matrix} \qquad \begin{matrix}[y : B] \\ l(y) : C\end{matrix}}{CASE(inr(b), \acute{x}r(x), \acute{y}l(y))} + - E_2$$

In the first case, one has $a : A$ in the start, but after introducing $inl(a) : A + B$, a becomes hidden, i.e., one loses direct access to it. The same thing happens to $b : B$ and $inr(b) : A + B$ in the second case. Thus, one proceeds adding local assumptions $x : A$ and $y : B$. With that in mind, (QUEIROZ; GABBAY, 1995) showed that the existential should mirror this aspect. The witness should be hidden and one should proceed by introducing Skolem-like local assumptions. Thus, the elimination rule for the dependent pair is reformulated (QUEIROZ; GABBAY, 1995):

$$\frac{n : \Sigma_{(x:A)}B(x) \qquad \begin{matrix}[t : A,\ f(t) : B(t)]\\ h(t,f) : C\end{matrix}}{E(n, \acute{t}\acute{f}h(t,f)) : C}\ (\Sigma - E)$$

This elimination rule eliminates the dependent pair without giving direct access to the witness. Of course, it generates new computation rules. Here follows the reduction rule(QUEIROZ; GABBAY, 1995):

$$\frac{\dfrac{a : D \qquad f(a) : F(a)}{\langle a, f(a)\rangle : \Sigma_{(x:D)}F(x)}\ \Sigma - I \qquad \begin{matrix} t : D, g(t) : F(t)\\ d(g,t) : C\end{matrix}}{E(\langle a, f(a)\rangle, \acute{g}\acute{t}d(g,t)) : C}\ \Sigma - E \qquad \rhd_{\beta}$$

$$\begin{matrix}[A : D, f(a) : F(a)]\\ [f/g, a/t]d(g,t)\end{matrix}$$

The last rule is the induction(QUEIROZ; GABBAY, 1995):

$$\frac{c : \Sigma_{(x:D)}P(x) \qquad \dfrac{[t : D] \qquad [g(t) : P(t)]}{\langle t, g(t)\rangle : \Sigma_{(y:D)}P(y)}\ \Sigma - I}{E(c, \acute{g}\acute{t}\,\langle t, g(t)\rangle) : \Sigma_{(y:D)}P(y)}\ \Sigma - E \qquad \rhd_{\eta} \qquad c : \Sigma_{(x:D)}P(x)$$

2.1.9 The Natural Numbers

It has been said that one of the main objectives that motivated type theory was the fact that it worked as a foundation for constructive mathematics. Thus, a theory that proposes to be a foundation for mathematics should at least be able to formalize the natural numbers. Set theory, for example, is capable of doing that. Thus, the objective of this subsection is to show how the natural numbers can be constructed in type theory and to show some basic properties of the naturals. As always, we start with basic constructions: (HARPER, 2012):

$$\frac{}{0 : \mathbb{N}}\ (\mathbb{N} - I_0) \qquad\qquad \frac{m : \mathbb{N}}{succ(m) : \mathbb{N}}\ (\mathbb{N} - I_s)$$

The natural are constructed in a simple way, from the two rules above. The zero introduction $(\mathbb{N} - I_0)$ should be understood as the base of the construction. It states the existence of at least one term called 0 that is a term of $\mathbb{N}$. From a term $m : \mathbb{N}$, one can apply the successor induction $(\mathbb{N} - I_s)$ to obtain another term of type $succ(m) : \mathbb{N}$. Thus, if one starts from 0 and inductively construct numbers applying *succ*, one wind up with the entire set of natural numbers. Nevertheless, those two rules does not give us the tools to work with the natural numbers. Ideally, one wants to construct functions on the naturals and also apply the inductive principle to construct proofs. Those tools are given by the computation rules. Let's start with the non-dependent case (HARPER, 2012):

$$\dfrac{m : \mathbb{N} \qquad C \text{ type} \qquad N_0 : C \qquad \begin{array}{c}[x : C]\\ N_s : C\end{array}}{rec(m, N_0, \lambda x.N_s) : C}$$

The inference rule above can be rather confusing. The idea is to use a recursor *rec* to define basic functions on the naturals, such as addition, multiplication and exponentiation. The recursor receives a base case N_0, that states how *rec* should act when it iterates 0 times. Another piece of information is that given x iterations, the recursor needs to know how to compute the next iteration. This next iteration is represented by N_s. Thus, based on the inductive definition of the naturals, this is enough to define how any function on the naturals works. To better illustrate that, let's use *rec* to define addition. Given $m, n : \mathbb{N}$, we want $m + n : \mathbb{N}$. Starting from m, one can define the base case to be $m + 0 = m$. Thus, one just needs to iterate n times the function *succ* starting from m. This is expressed as following:

$$\dfrac{\dfrac{m : \mathbb{N} \qquad \mathbb{N} \text{ type} \qquad m : \mathbb{N} \qquad \begin{array}{c}[n : \mathbb{N}]\\ succ(n) : \mathbb{N}\end{array}}{rec(m, m, \lambda n.succ(n)) : \mathbb{N}}}{\lambda m.rec(m, m, \lambda n.succ(n)) : \mathbb{N} \to \mathbb{N}}$$

This rule used to define recursive functions on the naturals is called the non-dependent case, since it originates terms of a non-dependent type C. To use the induction principle, one needs to define the dependent case (HARPER, 2012):

$$\dfrac{m : \mathbb{N} \qquad \begin{array}{c}[x : \mathbb{N}]\\ C(x) \text{ type}\end{array} \qquad N_0 : [0/x]C(x) \qquad \begin{array}{c}[x : \mathbb{N}, y : C(x)]\\ N_s : [suc(x)/x]C(x)\end{array}}{rec(m, N_0, (x, \lambda y.Ns)) : [m/x]C(x)}$$

If one looks closely, one should notice that the non-dependent case is the induction principle of the natural numbers. Basically, it states that given a proof of the base case

$N_0 : C(0)$ and the inductive step, i.e., that a proof $y : C(x)$ implies a proof of $N_s : C(x{+}1)$, then one should wind up with a proof $C(m)$ for any $m : \mathbb{N}$. With that, we conclude the construction and main properties used to work with the naturals in type theory.

2.1.10 Additional Information

We have said that a type can be semantically interpreted in different ways. It can be a proposition, a topological space, a set, among others semantical interpretation. The main ones are summed up in the table below:

Table 1 – Multiple semantical interpretations of a type

Types	**Logic**	**Sets**	**Homotopy**
A	proposition	set	space
$a : A$	proof	element	point
$B(x)$	predicate	family of sets	fibration
$b(x) : B(x)$	conditional proof	family of elements	section
0,1	$\bot, \top$	$\varnothing$, $\{\varnothing\}$	$\varnothing, *$
$A + B$	$A \vee B$	disjoint union	coproduct
$A \times B$	$A \wedge B$	set of pairs	product space
$A \to B$	$A \Rightarrow B$	set of functions	function space
$\Sigma_{x:A} B(x)$	$\exists_{(x:A)} B(x)$	disjoint sum	total space
$\Pi_{x:A} B(x)$	$\forall_{(x:A)} B(x)$	product	space of sections
Id_a	equality $=$	$\{(x, x) \mid x \in A\}$	path space A^I

Source: Homotopy Type Theory Book (Univalent Foundations Program, 2013).

As one can see, type theory is capable of representing many diverse objects of mathematics and logic. In the previous subsections, we have shown the basic constructions that everyone interested in type theory should be familiar with. Nevertheless, we have not talked about the identity type yet. Since it is a type of central to the main contribution of this work, the next section is completely focused on explaining it.

2.2 IDENTITY TYPE

After briefly introducing the main concepts and types of type theory, the objective of this section is to introduce the main entity of this work, the identity type. Here we are going to introduce the classic approach of type theory. One can check our proposed formulation for the identity type in **chapter 4** of this work.

The identity type is arguably the most interesting concept of type theory. This claim is based on the fact that many results have been achieved using it. One of these was the discovery of the Univalent Models in 2005 by Vladimir Voevodsky (VOEVODSKY, 2014). From this work, a groundbreaking result has arisen: the connection between type theory and homotopy theory. The intuitive connection is simple: a term $a : A$ can be considered as a point of the space A and $p : Id_A(a, b)$ is a homotopy path between points $a, b \in A$(Univalent Foundations Program, 2013). This has given rise to a whole new area of research, known as Homotopy Type Theory. It leads to a new perspective on the study of equality, as expressed by Voevodsky in a recent talk in *The Paul Bernays Lectures* (Sept 2014, Zürich): equality (for abstract sets) should be looked at as a *structure* rather than as a *relation*.

2.2.1 Formal Definition of The Identity Type

We start with the formation and introduction rules (QUEIROZ; OLIVEIRA, 2014b):

$$\frac{A \text{ type} \qquad a : A \qquad b : A}{Id_A^{int}(a, b) \text{ type}} \; Id^{int} - F$$

$$\frac{a : A}{r(a) : Id_A^{int}(a, a)} \; Id^{int} - I$$

First, we need to clarify the meaning of *int* that appears in Id^{int}. This *int* indicates that we are working with the intensional version of the identity type. The difference between the intensional and extensional version will be explained in detail in the next subsection. For now, let's postulate that every time *int* or *ext* do not appear above Id_A, one should assume that the version being used is the intensional one.

The identity type formation ($Id^{int} - F$) is pretty straightforward. Given any terms $a, b : A$, it is possible to create a type that will only be inhabited if there exits an witness that testifies the propositional equality between a and b. Nevertheless, the existence of type $Id_A(a, b)$ does not dependent on whether it is inhabited or not.

The rule identity type introduction ($Id^{int} - I$) states that there is only one canonical proof of equality, the reflexivity. It states that any term $a : A$ is always propositional equal to itself. This introduces a canonical proof of reflexivity $r(a) : Id_A^{int}(a, a)$. Thus, the main idea is that every proof of equality can be constructed inductively from a reflexive proof of equality (QUEIROZ; OLIVEIRA, 2014b):

$$\frac{a : A \qquad b : A \qquad c : Id_A^{int}(a, b) \qquad \begin{array}{c}[x : A] \\ d(x) : C(x, x, r(x))\end{array} \qquad \begin{array}{c}[x : A, y : A, z : Id_A(x, y)] \\ C(x, y, z) \text{ type}\end{array}}{J(c, d) : C(a, b, c)} \; Id - E$$

$$\frac{a : A \qquad \begin{array}{c}[x : A] \\ d(x) : C(x, x, r(x))\end{array} \qquad \begin{array}{c}[x : A, y : A, z : Id_A(x, y)] \\ C(x, y, z) \text{ type}\end{array}}{J(r(a), d(x)) = d(a/x) : C(a, a, r(a))} \; Id - Eq$$

The identity type elimination ($Id - E$) introduces the recursor J and eliminates the identity type in the process. The inner details of the recursor J are pretty complicated to understand and uses complex concepts of category theory. Nevertheless, one can try to intuitively understand J. Basically, $J(c, d) : C(a, b, c)$ works constructing c from a reflexive proof d. In this case, as any recursor of type theory, c is constructed based on multiple iterations that starts from the base d. Since this elimination rule has many terms, constructing some types using J can be a cumbersome process, as we are going to see in the next subsection. In this work, we are going to introduce a way of formalizing the identity type using a new entity known as computational paths. We believe that this way is simpler and more straightforward to use than the one introduced here. Nevertheless, we leave this discussion to a further chapter of this work.

2.2.2 Basic Constructions

The objective of this subsection is to show the use of the elimination rule of the identity type in practice. The constructions that we have chosen to build are the reflexive, transitive and symmetric type of the identity type. Those were not random choices. The main reason is the fact that reflexive, transitive and symmetric types are essential to the process of building a groupoid model for the identity type (HOFMANN; STREICHER, 1994).

Before we start the constructions, we think that it is essential to understand how to use the eliminations rules. The process of building a term of some type is a matter of finding the right reason. In the case of J, the reason is the correct $x, y : A$ and $z : Id_A(a, b)$ that generates the adequate $C(x, y, z)$.

We start proving the reflexivity. We want a witness for $\Pi_{(a:A)} Id_A(a, a)$. This case is trivial, since we have the canonical proof of identity:

$$\frac{a : A}{r(a) : Id_A(a, a)}\ Id - I$$

The second construction is the symmetry. This one does not follow trivially from the introduction rule. Thus, we need to use the elimination rule in some way. Our objective is to construct a term for the type $\Pi_{(a:A)}\Pi_{(b:A)}(Id_A(a, b) \to Id_A(b, a))$.

Looking at the elimination rule of the constructor J, it is clear that our main objective is to find a suitable $C(x, y, z)$, with $x : A$, $y : A$ and $z : Id_A(a, b)$. The main problem is the fact that, to find the correct reason, we do not have a fixed process or a fixed set of rules to follow. One needs to rely on one's intuition. In this case, one could conclude that the correct reason is to look at $C(x, y, , z)$ as the type $Id_A(y, x)$, with $x : A$, $y : A$ and z as any term (in the deduction, z will be represented by $-$, to indicate that it can be any term). Thus, we obtain the following deduction:

$$
\dfrac{\dfrac{\dfrac{\dfrac{a : A \qquad b : A \qquad c : Id_A(a,b) \qquad \begin{matrix}[x : A]\\ r(x) : Id_A(x,x)\end{matrix} \qquad \begin{matrix}[x : A, y : A, - : Id_A(a,b)]\\ Id_A(y,x) \text{ type}\end{matrix}}{J(c, r(x)) : Id_A(b,a)}\,Id-E}{\lambda c.J(c,r(x)) : Id_A(a,b) \to Id_A(b,a)}\to -I}{\lambda b.\lambda c.J(c,r(x)) : \Pi_{(b:A)}(Id_A(a,b) \to Id_A(b,a))}\,\Pi - I}{\lambda a.\lambda b.\lambda c.J(c,r(x)) : \Pi_{(a:A)}\Pi_{(b:A)}(Id_A(a,b) \to Id_A(b,a))}\,\Pi - I
$$

The third and last construction will be the transitivity. Our objective is to construct a term for the type $\Pi_{(a:A)}\Pi_{(b:A)}\Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))$.

Let's now use J to build a term for the transitivity. The proof will be based on the one found in (Univalent Foundations Program, 2013). The difference is that instead of defining induction principles for J based on the elimination rules, we will use the rule directly. The complexity is the same, since the proofs are two forms of presenting the same thing and they share the same reasons. As one should expect, the first and main step is to find a suitable reason. In other words, we need to find suitable $x, y : A$ and $z : Id_A(a,b)$ to construct an adequate $C(x,y,z)$. Similar to the case of symmetry, this first step is already problematic. Different from our approach, in which one starts from a path and applies intuitive equality axioms to find a suitable reason, there is no clear point of how one should proceed to find a suitable reason for the construction based on J. In this case, one should rely on intuition and make attempts until one finds out the correct reason. As one can check in (Univalent Foundations Program, 2013), a suitable reason would be $x : A, y : A, - : Id_A(x,y)$ and $C(x,y,z) \equiv Id_A(y,c) \to Id_A(x,c)$. Looking closely, the proof is not over yet. The problem is the type of $C(x,x,r(x))$. With this reason, we have that $C(x,x,r(x)) \equiv Id_A(x,c) \to Id_A(x,c)$. Therefore, we cannot assume that $q(x) : Id_A(x,c) \to Id_A(x,c)$ is the term $r(x)$. The only way to proceed is to apply again the constructor J to build the term $q(x)$. It means, of course, that we will need to find yet another reason to build this type. This second reason is given by $x : A, y : A, - : Id_A(x,y)$ and $C'(x,y,z) \equiv Id_A(x,y)$. In that case, $C'(x,x,r(x)) = Id_A(x,x)$. We will not need to use J again, since now we have that $r(x) : Id_A(x,x)$. Then, we can construct $q(x)$:

$$
\dfrac{\dfrac{x : A \qquad c : A \qquad q : Id_A(x,c) \qquad \begin{matrix}[x : A]\\ r(x) : Id_A(x,x)\end{matrix} \qquad \begin{matrix}[x : A, y : A, - : Id_A(x,y)]\\ Id_A(x,y) \text{ type}\end{matrix}}{J(q, r(x)) : Id_A(x,c)}\,Id-E}{\lambda q.J(q,r(x)) : Id_A(x,c) \to Id_A(x,c)}\to -I
$$

Finally, we obtain the desired term:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a : A \qquad b : A \qquad p : Id_A(a,b) \qquad \begin{matrix}[x : A] \\ \lambda q.J(q, r(x)) : Id_A(x,c) \to Id_A(x,c)\end{matrix} \qquad \begin{matrix}[x : A, y : A, - : Id_A(x,y)] \\ Id_A(y,c) \to Id_A(x,c) \text{ type}\end{matrix}}{J(p, \lambda q.J(q, r(x))) : Id_A(b,c) \to Id_A(a,c)}\, Id - E}{\lambda p.J(p, \lambda q.J(q, r(x))) : Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c)}\, \to -I}{\lambda c.\lambda p.J(p, \lambda q.J(q, r(x))) : \Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))}\, \Pi - I}{\lambda b.\lambda c.\lambda p.J(p, \lambda q.J(q, r(x))) : \Pi_{(b:A)}\Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))}\, \Pi - I}{\lambda a.\lambda b.\lambda c.\lambda p.J(p, \lambda q.J(q, r(x))) : \Pi_{(a:A)}\Pi_{(b:A)}\Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))}\, \Pi - I$$

This construction is an example that makes clear the difficulties of working with this approach for the identity type. We had to find two different reasons and use two applications of the elimination rule. Another problem is the fact that the reasons were not obtained by a fixed process, like the applications of axioms in some entity of type theory. They were obtained purely by the intuition that a certain $C(x, y, z)$ should be capable of constructing the desired term. For that reason, obtaining these reasons can be troublesome.

Further in this work we are going to construct those types using computational paths. Thus, we will have some base on which we can compare the two approaches.

2.2.3 Extensionality vs Intensionality

We have seen that functions are extensional in set theory. Functions are considered as pairs of input and output. The way of how those pairs were obtained is ignored completely. This does not occurs on the aforementioned formulation of the identity type. In that case, the proof of how the equality has been obtained matters. That is the reason way we said that this approach is intensional. In the original formulation of type theory, Martin-Löf proposed an intensional approach (MARTIN-LÖF, 1998) and an extensional one (MARTIN-LÖF, 1982; MARTIN-LÖF, 1984). Since we have already shown the construction of the intensional approach, we show the construction of the extensional one in this subsection. We have the following rules (QUEIROZ; OLIVEIRA; RAMOS, 2016):

$$\frac{A \text{ type} \qquad a : A \qquad b : A}{Id_A^{ext}(a, b) \text{ type}}\; Id^{ext} - F$$

$$\frac{a = b : A}{r : Id_A^{ext}(a, b)}\; Id^{ext} - I$$

The extensional identity type formation ($Id^{ext} - F$) is completely equal to the intensional case. The extensional identity type introduction ($Id^{ext} - I$) is also very simple. If we have that $a = b : A$, then one has for sure at least one proof r that establishes this equality. Naturally, r has type $Id_A^{ext}(a, b)$. We have now the elimination and equality rules (QUEIROZ; OLIVEIRA; RAMOS, 2016):

$$\frac{c : Id_A^{ext}(a, b)}{a = b : A}\; Id^{ext} - E$$

$$\frac{c : Id_A^{ext}(a, b)}{c = r : Id_A^{ext}(a, b)}\; Id^{ext} - Eq$$

The first thing one should notice is that extensional identity type elimination ($Id^{ext} - E$) is much simpler than the intensional version. One does not need to define over-complicated structures like J. Given any element $Id_A^{ext}(a, b)$, since we do not need to keep track of the equality proof, one can extract directly that $a = b : C$. At this point, the information that c is responsible for establishing is irrelevant. As one can see, the existence of the term c is completely excluded of the final inference. The fact that we do not need to keep track of an equality proof is also seen in the extensional identity type equality $(Id^{ext} - Eq)$. This rule shows that any two equality proofs c and r are equal. Thus, how c and r are constructed does not matter. The only relevant fact is that they are both proofs of the equality $a = b : A$.

The approach that we are going to propose based on computational paths will be clearly intensional. This is due to the fact that computation is intrinsically intensional. Two algorithms can solve the same problem, but they can be implemented in completely different ways and have difference space and time complexities. Also, the intensional version is much more interesting, since it gave rise to homotopy type theory. Thus, this work will introduce computational paths as an alternative way of formalizing the intensional type theory.

2.3 CONCLUSION

In this chapter, we have introduced the basic types of type theory. We showed how to construct and compute using those types. We have also highlighted the essential difference between definitional equality and propositional equality. Based on this difference, we showed how to construct a type based on propositional equality, called identity type.

A good understanding of the basic concepts of this chapter is essential to understand the main results of this work. In further chapters, we are going to propose a different way of formalizing intensional identity type. We are going to compare the two approaches using the basic constructions shown in this chapter. In the next chapter, we are going to introduce the basic concepts of another theory essential to the results of this work: category theory.

3 CATEGORY THEORY

In this chapter, we introduce the theory that we are going to use to develop the mathematical aspect of computational paths. The theory used to construct this mathematical model will be category theory. We are going to see that this theory is extremely powerful, but it is abstract and complex. With that in mind, we start this chapter defining and exposing the main basic concepts of this theory. We then use those basic concepts to arrive at more complicated results, which will be necessary in the construction of our mathematical model for computational paths. Since this model has many details and peculiarities, we leave it to the next chapter of this work.

Category theory appeared for the first time in the works of Eilenberg & Mac Lane in 1945. It was originally proposed as just a tool to study natural transformations and functors(MARQUIS, 2014). Functors and natural transformations are basically entities that transform one mathematical structure into another. For instance, one can transform a function between sets into a homotopy between topological spaces. Despite being proposed as only a tool, category theory grew quickly and became a well-established theory. Today, it is widely used in mathematics and computer science. Even logic and λ-calculus can be modeled by category theory. It also has been used as foundation of mathematics in place of ZFC (MARQUIS, 2014).

Despite its power of abstraction, category theory does not hinder the importance of type theory. It can be used as a foundation for mathematics, but it shares some problems that appears in ZFC, such as the fact that it is hard to be modeled by computers. Thus, the importance of category theory to this work is limited to the fact that it is closely related to type theory. This fact is clear when one realizes that many type theoretical constructions can be mathematically modeled using category theory.

It is also important to note that this chapter is not limited to basic category theory. To obtain the results that we want about computational paths, one also needs to define some basic concepts of higher category theory. Thus, we are going to introduce those concepts in the end of this chapter.

3.1 BASIC CONCEPTS

Category theory has a rather interesting aspect when compared directly with set theory or type theory. It is the fact that set theory and type theory have a fundamental concept in the core of those theories. One gains nothing asking what is the exact definition of sets or types, but one can ask what results can be obtained from those concepts. In category

Parts of this chapter are based on previous research done by the author and appears in his Master's Thesis (RAMOS, 2015) theory, this does not happen. It is possible to prove from rules and well-established entities whether some structure is a category or not. Perhaps the best way of introducing category theory is to analyze some aspects of functions. The reason for that is that one can affirm that a category is constructed based on structures called morphisms that are very similar to functions.

Consider that we have sets A, B and C and functions f and g such that $f : A \to B$ and $g : B \to C$. Thus, it is possible to construct a composite function $(g \circ f) : A \to C$ defined by $(g \circ f) = g(f(x))$. We have the following diagram:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g\circ f"'] & B \arrow[d, "g"] \\
 & C
\end{tikzcd}$$

One should become acquainted with those kinds of diagrams, since they commonly appear in category theory. Categories are based on objects (in the above diagrams, the objects are sets) and arrows or morphisms (in the above diagram, the functions). Thus, it is natural to think that those structures are represented by diagrams. They are not just a graphic representation of some structure, since they are frequently used as the main argument in proofs of important properties.

One interesting property rises from function composition: it is associative. In other words, if there is some $h : C \to D$, then $(h \circ g) \circ f = h \circ (g \circ f)$. This property is represented by the following diagram (AWODEY, 2010):

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g\circ f"'] & B \arrow[d, "g"] \arrow[dr, "h\circ g"] & \\
 & C \arrow[r, "h"] & D
\end{tikzcd}$$

Functions have another interesting property. For any set A, there is a identity function $1_A : A \to A$ defined by $1_A(a) = a$. This function can be considered as the neutral element of the composition operation, i.e. $1_B \circ f = f = f \circ 1_A$. In the next subsection we give a formal definition for the concept of category, which will give general versions of those properties.

3.1.1 Categories

Category is the main concept of category theory. Its definition follows:

Definition 3.1 (Categories (AWODEY, 2010)). *A category $\boldsymbol{C}$ is a structure with the following elements and rules:*

- *Objects: Objects are usually represented by capital letters $A, B, C....$ The class of all objects is called C_0.*
- *Arrows(morphisms): Arrows or morphisms are commonly represented by letters which are used to represent functions, such as $f, g, h....$ It is importance to notice that arrows always appear between two objects. The class of all arrows is called C_1.*
- *Domain and range: Given any arrow $f : A \to B$, the domain of f is A and is written as $dom(f) = A$. The range of f is B and is written as $cod(f) = B$.*
- *Composition: Given any arrows $f : A \to B$ and $g : B \to C$, there is always an arrow $h = (g \circ f) : A \to C$. This arrow is the composition of f and g. Moreover, h has a universal property, i.e., given any $m = (g \circ f)$, then $m = h$.*
- *Identity Arrow: For any object A there is an arrow $1_A : A \to A$ called identity arrow.*
- *Associativity rule: Given any arrows $f : A \to B$, $g : B \to C$ and $h : C \to D$, then $(h \circ g) \circ f = h \circ (g \circ f)$ always holds.*
- *Identity rule: Given any $f : A \to B$, then $1_B \circ f = f = f \circ 1_A$.*

From the above definition, it is clear that a function has a categorical structure. In fact, one of the most common category is **Sets**. In this category, objects are sets and arrows are functions between sets. As we have just seen, a function follows associative and identity rules and thus, **Sets** is clearly a category. Before we define further concepts, it is important to show the use of the above definition in another example. An interesting one is the structure **Pos**. As we have seen in *chapter 1*, a set can be ordered by an order relation $\leq$. Those ordered sets are known as *posets* (partially ordered sets). That way, an object in this structure is a poset and an arrow is a monotone function (AWODEY, 2010). A function is monotone iff $a \leq b \Rightarrow f(a) \leq f(b)$.

Proposition 3.1. ***Pos*** *is a category.*

Proof. A proof that some structure is a category follows a well-defined set of steps. Initially, it is necessary to define what is an arrow, the compositions and the identity arrow. Then, one just needs to check if the rules of associativity and identity hold.

- Composition: The composition is similar to the one given in **Sets**, i.e., given any $f : A \to B$ and $g : B \to C$, then one can define $(g \circ f) = g(f(x))$. In the case of **Pos**, $(g \circ f)$ needs to be monotone. Suppose that $a \leq b$. Since f is monotone, then $f(a) \leq f(b)$. Since g is monotone, then $g(f(a)) \leq g(f(b))$ and thus, $g \circ f$ is monotone.

- Identity: The identity arrow will also be similar to the one in **Sets**, i.e., the identity arrow will be an $1_A : A \to A$ defined by $1_A(a) = a$. 1_A is clearly monotone, since $a \leq b$ then $1_A(a) = a \leq b = 1_A(b)$.
- Associativity rule: Since arrows are functions, the associative rule clearly holds.
- Identity rule: Analogous to the associative rule. It has already proved for functions that 1_A respects the identity rule.

$\square$

Therefore **Pos** is a category. Our choice of **Pos** as an example was intentional, since it is a category of great importance to computer science. The reason for that is the fact that **Pos** is used on the formalization of boolean algebra. Nevertheless, this formalization is out of the scope of this work. After introducing the concept of category, we can proceed with other interesting concepts.

3.1.2 Isomorphism

In many areas of mathematics it is rather common to work with two different structures that have similar properties. Sometimes one wants to establish a direction connection between those structures. To do this, one establishes an isomorphism. The main problem is that the definition of isomorphism is intrinsically dependent to the structure being used. For example, the concept of isomorphism between sets is different to isomorphism between posets, which is different from an isomorphism between groups. Thus, the unification of those different definitions for the same concept would be a great feat. Fortunately, this can be easily done in category theory. It also showcases how powerful is this theory. We have the following definition:

Definition 3.2 (Isomorphism (AWODEY, 2010))**.** *: In a category **C**, an arrow $f : A \to B$ is called isomorphism if there exists an arrow $g : B \to C$ such that $g \circ f = 1_A$ and $f \circ g = 1_B$. When this happens, g is called inverse of f. We say that A is isomorphic to b and the write $A \cong B$. The arrow that generated this isomorphism is written as: $f : A \xrightarrow{\sim} B$.*

Proposition 3.2. *Given any isomorphism f and a inverse g of f, then g is unique.*

Proof. Let h be an inverse of f To show that g is unique, one needs to show that $h = g$. The following equations establish this result:

$$(g \circ f) \circ h = g \circ (f \circ h)$$
$$1_A \circ h = g \circ 1_B$$
$$h = g.$$

One can see that the first equation comes directly from the associativity of the composition. The second comes from the fact that g and h are inverses of f. The last equation comes from the fact that composition respects the identity rule. $\square$

Using the above result we can create a notation for an inverse of an isomorphism f. Since an inverse g is unique, we can write it as f^{-1}.

Using this definition, what would constitute an isomorphism in **Sets**? One can notice that sets A and B are isomorphic iff there is a $f : A \to B$ and a $g : B \to A$ such that $g(f(x)) = x$. Thus, $g = f^{-1}$, i.e., g is the inverse of f. In set theory, a function has an inverse iff f is bijective. Thus, A and B are isomorphic in **Sets** iff there is an injective function f from A onto B. Some interesting results arise from this definition of isomorphism:

Proposition 3.3. *A and B are isomorphic in **Sets** iff they have the same cardinality (this result can be informally interpreted as both sets having the same size).*

Proof. One can prove directly both sides of this proof. In one hand, One knows that if A and B are isomorphic, then there is a bijection $f : A \to B$. Since two sets by definition have the same cardinality if there is a bijection between them, then A and B have the same cardinality. On the other hand, if two sets have the same cardinality then there exists a bijection f between them, Thus, they are isomorphic. $\square$

In set theory, it is well known that one can prove that $\mathbb{N}$, $\mathbb{Z}$ e $\mathbb{Q}$ have the same cardinality (HRBACEK; JECH, 1999). Thus, by the above proposition, the naturals are isomorphic to the integers and to the rationals. At a first glance, it can be completely counterintuitive. The naturals have the same structure as the integers? To answer this, one should remember that in **Sets** the object of the category is only a set. Thus, this category does not consider a set together with a order relation. If one considers the naturals and integers with their usual orderings, then it is obvious that they are not isomorphic. One direct argument is that in their usual orderings, the naturals are well-ordered, but the integers are not.

What would constitute an isomorphism in **Pos**? Given two orderings $(A, \leq')$ and $(B, \leq'')$, one can think of an isomorphism as a bijection f between A and B that respects the ordering, i.e., if $a, b \in A$ and $a \leq' b$, then $f(a) \leq'' f(b)$.

From this definition of isomorphism, one can show that one can pick a suitable ordering to make the naturals isomorphic to the integers:

Proof. Consider the naturals together with its traditional ordering, i.e, $(\mathbb{N}, \leq) = 0, 1, 2, ...$ For the integers, consider an ordering $\leq''$ such that $(\mathbb{Z}, \leq'') = 0, -1, 1, -2, 2, -3, 3.....$ One needs to find a bijection f such that $f(0) = 0$, $f(1) = -1$, $f(2) = 1$, $f(3) = -2$, etc. To do that, one can define $f(x) = (-1)^x. \left\lceil \frac{x}{2} \right\rceil$. f is clearly monotone, i.e., $a \leq b \Rightarrow f(a) \leq'' f(b)$ and it is also onto. Thus, $(\mathbb{N}, \leq) \cong (\mathbb{Z}, \leq'')$. $\square$

In this subsection, we have seen a way of using category theory to give a general definition for a important concept of mathematics. We have also shown practical examples using **Sets** and **Pos**. Isomorphism will be an essential concept in the mathematical model of computational paths.

3.1.3 Groupoid

We can use the concept of isomorphism to define a structure known as groupoid. It has the following definition:

Definition 3.3 (Groupoid (AWODEY, 2010))**.** *Given any category **C**, we say that **C** is a groupoid iff every arrow $f \in C_1$ is an isomorphism. In other words, every pair of objects are mutually isomorphic.*

The concept of groupoid plays a main role in the semantical definition of the identity type. If a category **C** is a groupoid, then every object of **C** has a similar structure. In fact, this concept has achieved great importance due to the discovery of the fact that types in Martin-Löf's type theory have a groupoid structure (HOFMANN; STREICHER, 1994). In fact, one can prove that types have a weak groupoid structure. Weak here means that the equalities do not hold on the nose (i.e., the equalities are not definitional), but only up to propositional equality. One will have a better grasp of the concept of a weak structure in the next chapter, when we introduce the weak groupoid model of computational paths. In fact, one of the objectives of our next chapter is to show that one can use computational paths to build a weak structure similar to the one proposed by (HOFMANN; STREICHER, 1994), which uses the usual formulation of the intensional identity type to build it. With that in mind, one should notice that groupoid plays an important rule in this work.

We leave further details of this topic for the next chapter. Nevertheless, to better illustrate this concept, we are going to give simple examples. Take a category whose objects are sets and the arrows are bijections between these sets. Since bijections are isomorphisms, this category constitutes a groupoid structure. Another example of groupoid is constructed using **Pos**. One can take a category whose objects are ordered sets $(A_n, \leq_n)$ and arrows are bijections that respect the orderings. Thus, every arrow is an isomorphism.

3.1.4 Functors

The notion of functor is one of the most interesting and important concepts of category theory. Functors could be understood as a function that receives a category as input. Therefore, a functor transforms one category into another For example, one could think of the transformation of a category **C** into a category **D**. To do that, one needs to transform every object of **C** into a object of **D** and the arrows of **C** into arrows of **D**. Moreover, the associativity and identity rule must be preserved. We have the following formal definition:

Definition 3.4 (Functor (AWODEY, 2010))**.** *A functor $F : \boldsymbol{C} \to \boldsymbol{D}$ between categories $\boldsymbol{C}$ and $\boldsymbol{D}$ is a mapping of objects and arrows such that:*

- *$A \in C_0$ is mapped to $F(A) \in D_0$,*
- *$f : A \to B \in C_1$ is mapped to $F(f) : F(A) \to F(B) \in D_1$,*
- *$F(g \circ f) = F(g) \circ F(f)$,*
- *$F(1_A) = 1_{F(A)}$.*

Perhaps a better way of understanding the above definition is to visualize it through the help of diagrams (AWODEY, 2012):

Example 3.1. This example is going to show the diagram representation of the action of a generic functor $F : \mathbf{C} \to \mathbf{D}$. Let $\mathbf{C}$ be a category represented by the following diagram:

$$\begin{tikzcd}
A \arrow[loop left, "1_A"] \arrow[r, "f"] \arrow[dr, "g\circ f"'] & B \arrow[d, "g"] \\
 & C
\end{tikzcd}$$

After applying a functor F, one ends up with $F(\mathbf{C}) = \mathbf{D}$. This is represented by the following diagram:

$$\begin{tikzcd}
F(A) \arrow[loop left, "F(1_A)=1_{F(A)}"] \arrow[r, "F(f)"] \arrow[dr, "F(g\circ f)=F(g)\circ F(f)"'] & F(B) \arrow[d, "F(g)"] \\
 & F(C)
\end{tikzcd}$$

Functors are common in practice. A simple example is the functor $For : \mathbf{Pos} \to \mathbf{Sets}$. One can define For explaining its action on the objects and arrows of $\mathbf{Pos}$. As we have seen, an object of $\mathbf{Pos}$ is a pair $(A, \leq)$, in which A is a set. For has the following behavior: $For(A, \leq) = A$. In the arrows, we have that $For(f : (A, \leq) \to (B, \leq')) = f : A \to B$. Thus, For can be seen as a forgetful functor that forgets the ordering of a ordered set, returning only the set. For the arrows, For forgets that f is a monotone function, returning only the function. Forgetful functors are common in mathematics. Another well known example is $For : \mathbf{Grp} \to \mathbf{Sets}$. This functor receives a category of groups as input. It then forgets the group structure, returning only the underlying set. Forgetful functors plays a fundamental role in category theory, since it is the basic concept behind the concept of adjoint.

Further in this chapter we are going to see that the use of functors is needed in the definition of a 2-category.

3.1.5 Duality

Duality is a principle of category theory that states that for any category **C**, there is a dual category C^{OP}. This category is defined in the following way (AWODEY, 2012):

Definition 3.5. *Given any category $\boldsymbol{C}$, the category $\boldsymbol{C}^{OP}$ is the dual category of $\boldsymbol{C}$ and is defined in the following way::*

- $\boldsymbol{C}_0^{OP} = \boldsymbol{C}_0$.
- $\boldsymbol{C}_1^{OP} = \boldsymbol{C}_1$.
- *dom* $\boldsymbol{C}^{OP}$ = *cod* $\boldsymbol{C}$.
- *cod* $\boldsymbol{C}^{OP}$ = *dom* $\boldsymbol{C}$.

We can represent this duality using diagrams. Given any category **C** represented by the following diagram:

$$\begin{tikzcd}
A \arrow[loop left, "1_A"] \arrow[r, "f"] \arrow[dr, "g\circ f"'] & B \arrow[d, "g"] \\
& C
\end{tikzcd}$$

Thus, C^{OP} is given by:

$$\begin{tikzcd}
A \arrow[loop left, "\bar{1_A}=1_A"] & B \arrow[l, "\bar{f}"'] \\
& C \arrow[u, "\bar{g}"'] \arrow[ul, "\overline{g\circ f}=\bar{f}\circ\bar{g}"]
\end{tikzcd}$$

From a dual category, one can define a new kind of functor known as contravariant functor. A contravariant functor is given by $F : \mathbf{C}^{OP} \to \mathbf{D}$. Instead of defining a functor that acts on a category $\mathbf{C}^{OP}$, one can apply directly a contravariant functor in a category **C**. To do that, one just needs to invert the order of the arrows in the application of the functor. To better see this, consider **C** as the category of the above example. We have the following diagram of an application of a contravariant functor F:

$$\begin{tikzcd}
A \arrow[loop left, "F(1_A)"] & B \arrow[l, "F(f)"'] \\
& C \arrow[u, "F(g)"'] \arrow[ul, "F(g\circ f)"]
\end{tikzcd}$$

The use of contravariant functors in category theory is common. Also, duality is a very useful principle, since many properties of category theory appears from the dual category of some property. For example, if one has a diagram that represents the product of two categories, one just needs to get the dual category to define the coproduct.

3.1.6 Commutativity

In this subsection, we introduce two kinds of diagrams widely used in category theory, the commutative triangle and commutative square.

Definition 3.6 (Commutative triangle)**.** *A commutative triangle is the following diagram:*

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[dr, "h"'] & B \arrow[d, "g"] \\ & C \end{tikzcd}$$

Since it commutes, the triangle ABC respects the following equation: $g \circ f = h$.

Definition 3.7 (Commutative square)**.** *A commutative square is the following diagram:*

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "t"] & B \arrow[d, "g"] \\ C \arrow[r, "s"] & D \end{tikzcd}$$

Since it commutes, the square $ABCD$ respects the following equation: $g \circ f = s \circ t$.

Those two diagrams are used to define and prove many entities and properties of category theory.

3.1.7 Product between two categories

Before we define product between two categories, it is important to notice that there is a significant difference between product between categories and product between objects of a category. Product between categories is an operation that builds another category, whereas a product between objects of a category builds another object. Moreover, we can always obtain the product of two categories, but sometimes the product of two objects does not exist. Thus, in this subsection we introduce the product between two categories. We are also going to introduce the product between two objects in the sequel, since it is directly connected to the product of two types in type theory.

Definition 3.8 (Product between two categories (AWODEY, 2010))**.** *Given any categories* $\boldsymbol{C}$ *and* $\boldsymbol{D}$*, one can obtain a category* $\boldsymbol{C} \times \boldsymbol{D}$ *defined by:*

- *Objects: Objects have the form* (C, D)*, with* $C \in C_0$ *e* $D \in D_0$*.*
- *Arrows: Arrows have the form* $(f, g) : (C, D) \to (C', D')$*, with* $f : C \to C' \in C_1$ *e* $g : D \to D' \in D_1$*.*
- *Composition: Composition is defined component-wise, given by* $(f', g') \circ (f, g) = (f' \circ f, g' \circ g)$*.*
- *Identity: The identity arrow is also defined component-wise, given by* $1_{(C,D)} = (1_C, 1_D)$*.*
- *Two projection functors, represented by the following diagram:*

$$\begin{tikzcd}\boldsymbol{C} & \boldsymbol{C} \times \boldsymbol{D} \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & \boldsymbol{D}\end{tikzcd}$$

The projections respects the following equations:

$$\begin{aligned} \pi_1(C, D) = C \quad & \pi_2(C, D) = D \\ \pi_1(f, g) = f \quad & \pi_2(f, g) = g. \end{aligned}$$

This definition will play an important role in the definition of bicategory, as we are going to see further in this chapter.

3.1.8 Hom and Small Categories

In category theory, there is a important structure called Hom and defined as:

Definition 3.9 ($Hom_{\mathbf{C}}(A, B)$ (AWODEY, 2012))**.** $Hom_{\boldsymbol{C}}(A, B)$ *is a structure that contains all arrows from object* A *to object* B*, both objects of* $\boldsymbol{C}$*.*

Why we called Hom a structure instead of a set? The reason for that is the fact that sometimes Hom is more than just a set. In higher categories, for example, Hom can be a category. Also, when we talk about the size of a category, one has the following definitions:

Definition 3.10 (Large and small category (AWODEY, 2010))**.** *A category* $\boldsymbol{C}$ *is called small if* C_0 *and* C_1 *are sets. Otherwise the category is called large.*

The concept of large category rose up from the fact that in many categories the objects and arrows cannot be represented by sets. In many cases, each object of a category is another category. The same thing can happen to the arrows. Moreover, another important thing to notice is that many properties are only valid in small categories. This happens because some properties are directly dependent on the fact that the objects and arrows are sets. We have one more relevant definition:

Definition 3.11 (Locally small category (AWODEY, 2010))**.** *A category* $\boldsymbol{C}$ *is locally small if for any pair of objects* $A, B \in C_0$*, then* $Hom_{\boldsymbol{C}}(A, B)$ *is a set.*

Sometimes we just need some category **C** to be locally small. This will become more clear further in this chapter, when we investigate some properties of higher categories.

3.2 PRODUCT IN A CATEGORY

In the previous section, we have seen the definition of product between categories. In this section, our objective is to define product between objects of a category. This plays an important role in this work, since it has a direct connection with product of types, as we are going to show in this section. Based on those connections that we have decided to use category theory to propose a mathematical model for computational paths, as we are going to do in the next chapter.

Definition 3.12 (Product in a category (NLAB, 2014))**.** *In a category* $\boldsymbol{C}$*, the product between two objects* $X, Y \in C_0$ *is an object* $X \times Y$ *equipped with morphisms* $p : X \times Y \to X$ *and* $q : X \times Y \to Y$*, which are called projections. Moreover, the product has a universal property, which states that given any* $Z \in C_0$ *and maps* $f : Z \to X$ *and* $g \to Z \to Y$*, then there exists a unique map* $h : Z \to X \times Y$ *such that* $p \circ h = f$ *and* $q \circ h = g$*.*

This is represented in the following diagram (AWODEY, 2010):

$$\begin{tikzcd}
 & Z \arrow[dl, "f"'] \arrow[d, dotted, "h"] \arrow[dr, "g"] & \\
X & X \times Y \arrow[l, "p"] \arrow[r, "q"'] & Y
\end{tikzcd}$$

In the above diagram, the triangles are commutative. If one pays attention to the definition, the projections are easy to understand. Nevertheless, the part that related to the universal property, i.e., Z and the unicity of h might be a little complicated. This part can be explained in the following way: If there is an arrow $f : Z \to X$ and $g : Z \to Y$, then the unicity of $h : Z \to X \times Y$ indicates that there is only one way to construct the product from these two arrows. Therefore, we can write h uniquely as $h = (f, g)$.

Proposition 3.4. *In **Sets**, the product between two objects $A, B \in Sets_0$ is the cartesian product between two sets.*

Proof. To show that the product between two sets is the cartesian product, the first thing one needs to do is to show the projects. Thus, one just needs to define $p : A \times B \to A$ as $p(a, b) = a$ and define $q : A \times B \to B$ as $q(a, b) = b$. Now, one needs to show the universal property. Let $f : Z \to A$ and $g : Z \to B$. One can define a $h : Z \to A \times B$ in the following way: $h(x) = (f(x), g(x))$. To end this proof, one needs to check that $p \circ h = f$ and $q \circ h = g$. Indeed, $(p \circ h)(x) = p(h(x)) = p(f(x), g(x)) = f(x)$. Analogously, $(q \circ h)(x) = q(h(x)) = q(f(x), g(x)) = g(x)$. Therefore, the cartesian product of two sets should be seen a the product between two objects of **Sets**. $\square$

A natural questions arises. Is it possible to find another definition for the product of two objects in **Sets**. The following proposition answers this question (AWODEY, 2010):

Proposition 3.5. *If $\times$ and $\times'$ are products of a category **C**, then $A \times B \cong A \times' B$.*

Proof. This is one case that the a proof with the help of diagrams is much easier to see (AWODEY, 2010):

$$\begin{tikzcd}
 & A \times B \arrow[dl, "p_1"'] \arrow[dr, "p_2"] \arrow[d, dotted, "(p_1,p_2)"] & \\
A & A \times' B \arrow[l, "q_1"] \arrow[r, "q_2"'] \arrow[d, dotted, "(q_1,q_2)"] & B \\
 & A \times B \arrow[ul, "p_1"] \arrow[ur, "p_2"'] &
\end{tikzcd}$$

The main point here is that from the universal property of the products, one can obtain the arrows (p_1, p_2) and (q_1, q_2). Moreover, all triangles in the above diagram are commutative. Looking at the diagram, the following equations are obtained from the triangles $(A \times B)(A \times B)A$ and $(A \times B)(A \times B)B$:

$$p_1 \circ (q_1, q_2) \circ (p_1, p_2) = p_1$$
$$p_2 \circ (q_1, q_2) \circ (p_1, p_2) = p_2$$

Indeed, $(q_1, q_2) \circ (p_1, p_2)$ acts as the identity arrow of $A \times B$. By the unicity of those arrows, one knowns that $(q_1, q_2) \circ (p_1, p_2) = 1_{A \times B}$. Using the symmetry of the diagram, one obtains $(p_1, p_2) \circ (q_1, q_2) = 1_{A \times' B}$. Therefore (p_1, p_2) and (q_1, q_2) are inverses and establish the isomorphism between the two products. $\square$

Based on this proof, it is possible to state that every category has only one definition for the product of two objects since all possible definitions are mutually isomorphic.

3.2.1 Product in Type Theory and Category Theory

This section is going to show the close relation between type theory and category theory. To see that, we are going to show some type theoretic constructions together with the respective diagrams in type theory. We start with product formation $(\times - F)$ (NLAB, 2014):

$$\frac{A \text{ type} \qquad B \text{ type}}{A \times B \text{ type}} \times - F$$

Product formation is intuitive. Given any types A and B, it is possible to obtain $A \times B$. In categories, there is an equivalent proposition: If $A, B \in C_0$, then $A \times B in C_0$. Now we have product introduction $(\times - I)$ (NLAB, 2014):

$$\frac{a : A \qquad b : B}{\langle a, b\rangle : A \times B} \times - I$$

$$\begin{tikzcd}
 & Z \arrow[dl, "a"'] \arrow[d, dotted, "(a{,}b)"] \arrow[dr, "b"] & \\
A & A \times B & B
\end{tikzcd}$$

Indeed, given any terms $a : A$ and $b : B$, it is possible to construct $\langle a, b\rangle : A \times B$. In categories, it is represented by arrows a and b. Given those arrows, it is possible to obtain a unique $\langle a, b\rangle$ given by the universal property of the product. Now follows the first product elimination $(\times - E_1)$ and the second product elimination $(\times - E_2)$ (NLAB, 2014):

$$\frac{t : A \times B}{p_1(t) : A} \times - E_1 \qquad \frac{t : A \times B}{p_2(t) : B} \times - E_2$$

$$\begin{tikzcd}
 & Z \arrow[d, dotted, "t"] & \\
A & A \times B \arrow[l, "p_1"'] \arrow[r, "p_2"] & B
\end{tikzcd}$$

The eliminations are done using the projections. This is exactly that the above diagram represents. We also have diagram representations of the computation rules $p_1(\langle a, b\rangle) = a$ and $p_2(\langle a, b\rangle) = b$:

$$\begin{tikzcd}
 & Z \arrow[dl, "a"'] \arrow[d, dotted, "\langle a{,}b\rangle"] \arrow[dr, "b"] & \\
A & A \times B \arrow[l, "p_1"] \arrow[r, "p_2"'] & B
\end{tikzcd}$$

That way, one can see that we can translate type theoretic constructions to the language of category theory. In this subsection, we have shown the specific case of the product, but using a similar process one can translate any entity of type theory, including path-induction and the constructor J.

3.2.2 Coproduct

The objective of this section is not to give a detailed construction of the coproduct, but to show that it is possible to use duality on the constructions of the product to obtain direct constructions of the coproduct. Given any a category **C** and objects $A, B \in C_0$, the coproduct of two objects is represented by $A + B$ and given by the following diagram:

$$\begin{tikzcd}
 & Z & \\
'A \arrow[ur, "f"] \arrow[r, "i_1"'] & A + B \arrow[u, dotted, "{[a,b]}"'] & B \arrow[ul, "g"'] \arrow[l, "i_2"]
\end{tikzcd}$$

As one can see, the coproduct is exactly the dual category of the product. The projections are replaced by injections. It also has a universal product, that states the following: if there are $f : A \to Z$ and $g : B \to Z$, then there is a unique $h : A + B \to Z$. We can also use duality to prove the following proposition:

Proposition 3.6. *If $+$ and $+'$ are coproducts of a category $\boldsymbol{C}$, then $A + B \cong A +' B$.*

Proof. The proof is directly connected to the fact that the dual of an isomorphism is also an isomorphism. The duality only changes the direction of the arrow. Thus, if an arrow is an isomorphism in the original category, then it will also be in the dual. Since the isomorphism has been proved for the product, the duality keeps it □

Using the same process that we have established for the product, one can translate the construction of the coproduct in type theory to the language of category theory.

3.3 NATURAL TRANSFORMATIONS

Natural transformations are one of the most important concepts of category theory. As we have said in the introduction of this chapter, category theory was originally proposed to study natural transformations. Moreover, we are going to use this concept in the definition of bicategory. Perhaps a good way of introducing natural transformations is the following example: Suppose that A, B and C are objects of a category with products. Then, $(A \times B) \times C \cong A \times (B \times C)$ (AWODEY, 2010). We will not show the proof of this proposition, since it is pretty straightforward. One just needs to draw the diagram and establishes the equations using the commutative triangles. Nevertheless, the important piece o information is the fact that there exists an isomorphism $f : (A \times B) \times C \xrightarrow{\sim} A \times (B \times C)$. This isomorphism f has a limitation though. It establishes only the isomorphism for objects A, B and C. Ideally, one wants a general result that does not depend on the choice of objects A, B or C. To do that, one needs the concept of natural transformations.

Intuitively, a natural transformation is a morphism between functors. We can use the associativity of the product as our first example of natural transformation. First, one can conceive the following functors: $F : \mathbf{C} \times \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ defined by $F = (-_1 \times -_2) \times -_3$ and the functor $G : \mathbf{C} \times \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ defined by $G = -_1 \times (-_2 \times -_3)$. With that in mind, one needs to find a natural transformation $\theta : F \to G$. Moreover, one also needs to show that θ is an entity known as natural isomorphism. Thus, one finishes the proof that the isomorphism $(A \times B) \times C \cong A \times (B \times C)$ does not depend on the choice of the objects. In that case, it is said that the isomorphism is natural on A, B and C. Here follows a formal definition for natural transformation:

Definition 3.13 (Natural transformation (AWODEY, 2010))**.** *Given any categories* $\boldsymbol{C}$ *and* $\boldsymbol{D}$ *and functors* $F, G : \boldsymbol{C} \to \boldsymbol{D}$*, a natural transformation* $\theta : F \to G$ *is a family of arrows in* $\boldsymbol{D}$ *given by:* $(\theta_C : FC \to GC)_{C \in C_0}$*. Moreover, given a* $f : C \to C' \in C_1$*, then* $\theta_{C'} \circ F(f) = G(f) \circ \theta_C$*. This equality is represented by the following commutative square:*

$$\begin{tikzcd}
FC \arrow[r, "\theta_C"] \arrow[d, "F(f)"] & GB \arrow[d, "G(g)"] \\
FC' \arrow[r, "\theta_{C'}"'] & GD
\end{tikzcd}$$

The arrows θ_C *are called components* θ *in* C*.*

Looking at the above definition, one can conclude that to find a natural transformation between two functors, one just needs to find a suitable morphism for the components. This morphism needs to be natural, i.e, it must obey the commutative square. From the definition of natural transformation, one can define the following category:

Definition 3.14 (Category $Fun(\mathbf{C}, \mathbf{D})$ (AWODEY, 2010))**.** *Given any categories* $\boldsymbol{C}$ *and* $\boldsymbol{D}$*, it is possible to construct the category* $Fun(\boldsymbol{C}, \boldsymbol{D})$ *defined by:*

- *Objects: Objects are functors* $F : \boldsymbol{C} \to \boldsymbol{D}$*.*
- *Arrows: Arrows are natural transformations* $\theta : F \to G$*.*
- *Composition: Given any two arrows* $\theta : F \to G$ *e* $\phi : G \to H$*, it is possible to define* $(\phi \circ \theta)$ *component-wise:* $(\phi \circ \theta)_C = \phi_C \circ \theta_C$*.*

- *Identity: Given any functor F, the arrow 1_F is defined component-wise: $(1_F)_C = 1_{FC} : FC \to FC$.*

Therefore, we can now define natural isomorphism:

Definition 3.15 (Natural isomorphism (AWODEY, 2010))**.** *A natural isomorphism is a natural transformation $\theta : F \to G$ such that θ is an isomorphism in the category $Fun(\boldsymbol{C}, \boldsymbol{D})$.*

There is a easier way to check if a natural transformation is an isomorphism (AWODEY, 2010):

Proposition 3.7. *A natural transformation $\theta : F \to G$ is a natural isomorphism iff every component $\theta_C : FC \to GC$ is an isomorphism.*

Proof. ($\Rightarrow$) To prove this direction, one needs to prove that given any natural isomorphism $\theta : F \to G$,, then every component θ_C is also an isomorphism. Since θ is an isomorphism, then θ has an inverse $\phi : G \to F$. Thus, one has that $\theta \circ \phi = 1_G$ and $\phi \circ \theta = 1_F$. Thus, if one takes any component in C, one concludes that $(\theta \circ \phi)_C = (1_G)_C$ and $(\phi \circ \theta)_C = (1_F)_C$. By the definition of composition and identity in Fun, one obtains $\phi_C \circ \theta_C = 1_{FC}$ and $\theta_C \circ \phi_C = 1_{GC}$. Therefore, ϕ_C is inverse to θ_C. Therefore, every component is an isomorphism.

($\Leftarrow$) To prove the opposite direction, one needs to show that if every component θ_C of a natural transformation θ is an isomorphism, then θ is also an isomorphism. If each component is an isomorphism then for each θ_C there exists an inverse arrow ϕ_C such that $\theta_C \circ \phi_C = 1_{GC}$ and $\phi_C \circ \theta_C = 1_{FC}$. By the definition of composition and identity of Fun, one has that $\theta_C \circ \phi_C = (\theta \circ \phi)_C = 1_{GC} = (1_G)_C$. Thus, $\theta \circ \phi = 1_G$. Analogously, one shows that $\phi \circ \theta = 1_F$. Therefore, ϕ is a natural transformation inverse to θ. One concludes that θ is natural isomorphism. $\square$

The above proposition makes the notion of natural isomorphism easier to work with, since one only needs to show that every component is an isomorphism. To show how one can use natural transformations and natural isomorphisms in practice, consider the following proposition:

Proposition 3.8. *Let $\boldsymbol{C}$ be a category that has binary products and $A, B \in C_0$, then $A \times B \cong B \times A$ and the isomorphism is natural in A and in B.*

If we limited the above proposition to a proof of $A \times B \cong B \times A$, then it would be a matter of showing diagrams of the product to establish this isomorphism. Nevertheless, we also need to show that this isomorphism is natural in A and in B. There is only one way of doing that: using natural transformations. Here is the proof (AWODEY, 2010):

Proof. To prove this natural isomorphism, one needs to define the functors first. The first functor is $\times : \mathbf{C}\times\mathbf{C} \to \mathbf{C}$, defined by $\times = (-_1, -_2)$. The second is given by $\bar{\times} : \mathbf{C}\times\mathbf{C} \to \mathbf{C}$ defined by $\bar{\times} : (-_2, -_1)$. Thus, in objects $A\bar{\times}B = B \times A$ and in arrows $\alpha\bar{\times}\beta = \beta \times \alpha$. that way, one now needs to conceive a natural transformation θ between those two functors. One can define θ by its behavior when applied in terms (A, B). Thus, one can define $\theta_{(A,B)}(a, b) = (b, a)$, with a and b arrows of a product (i.e., given a Z, they are arrows $Z \to A$ e $Z \to B$ respectively). One also needs to show that θ is a natural transformation. In other words, to show that the following diagram commutes:

$$\begin{tikzcd}
A \times B \arrow[r, "\theta_{(A,B)}"] \arrow[d, "\alpha\times\beta"] & B \times A \arrow[d, "\beta\times\alpha"] \\
A' \times B' \arrow[r, "\theta_{(A',B')}"'] & B' \times A'
\end{tikzcd}$$

To show that the above diagram commutes, one only needs to take any (a, b) from Z, $a : Z \to A$ and $b : Z \to B$. Thus, we obtain the following equations:

$$(\beta \times \alpha)\theta_{(A,B)}(a, b) = (\beta \times \alpha)(b, a)$$
$$(\beta \times \alpha)(b, a) = (\beta b, \alpha a)$$
$$(\beta b, \alpha a) = \theta_{(A',B')}(\alpha a, \beta b)$$
$$(\alpha a, \beta b) = \theta_{(A',B')}(\alpha \times \beta)(a, b)$$

Therefore, $(\beta \times \alpha)\theta_{(A,B)}(a, b) = \theta_{(A',B')}(\alpha \times \beta)(a, b)$, proving that the square is commutative. Thus, one concludes that θ is a natural transformation. Nonetheless, one still needs to show that θ is an isomorphism. To do that, one can use **proposition 3.7**, i.e., one just needs to show that every component is an isomorphism. The isomorphism for each component is simple to show, since for each $\theta_{(A,B)}$ there is an inverse $\phi_{(A,B)} = \theta_{(B,A)}$. Here follows:

$$(\theta_{(A,B)} \circ \phi_{(A,B)})(B \times A) = \theta_{(A,B)}(A \times B) = B \times A$$
$$(\phi_{(A,B)} \circ \theta_{(A,B)})(A \times B) = \phi_{(B,A)}(B \times A) = A \times B$$

Therefore, one concludes:

$$(\theta_{(A,B)} \circ \phi_{(A,B)}) = 1_{\bar{\times}(A,B)}$$
$$(\phi_{(A,B)} \circ \theta_{(A,B)}) = 1_{\times(A,B)}.$$

Thus, each component θ em (A, B) is an isomorphism. Therefore, θ is an isomorphism. Thus, one concludes this proof. □

3.3.1 Adjoints

After introducing the concept of natural transformations, we can talk about adjoints. It is one of the most important concepts of this theory, since it has many applications.

We have talked about functions and how one can transform a structure into another. A question arises: what is the best way of making this transformation? In other words, we have an optimization problem in our hands. This subsection proposes a solution to this question. It introduces a concept that it is possible to optimize those transformations and its inverse process. For example, we talked about a forgetful functor that forgets the group structure and returns only the underlying set. As a matter of fact, it is the optimal way of transforming a group into a set. We also have an inverse process, we take a set and obtain a group freely generated by it. This is also optimal, thus we are going to say that those two functors form a pair of adjoints.

Perhaps one of the best way of introducing adjoints is to use the concept of monoid, the forgetful functor on a monoid and the functor that generates a free monoid. First, let's recall the classic definition of monoid:

Definition 3.16 (Monoid (NLAB, 2017))**.** *A monoid is a set M equipped with a binary operation $\mu : M \times M \to M$ and a special element $1 \in M$ (the neutral element) such that 1 and $x.y = \mu(x, y)$ satisfies the associative law:*

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

and the left and right unit laws:

$$1.x = x = x.1$$

Thus, from a monoid, one can clearly apply a forgetful functor to obtain the underlying set. From a set, one can freely generate a monoid using the elements of this set. Intuitively, a free generation works using the elements of the set as as letters and the operation of the monoid as compositions. Also, all equalities must come from the monoid axioms. Nevertheless, one can use category theory to precisely define a free monoid. Before we define that, it is important to notice some facts. If one have a monoid N, then one also have an underlying set $\|N\|$. Also, from every monoid homomorphism $f : N \to M$, one have a forgetful functor $\|f\| : \|N\| \to \|M\|$. Thus, one can define a free monoid in the following way:

Definition 3.17 (Free Monoid (AWODEY, 2010))**.** *A free monoid $M(A)$ on a set A is the monoid with the following universal mapping property: There's a function $i : A \to \|M(A)\|$ and given any monoid N and any function $A \to \|N\|$, there's a unique monoid homomorphism $\bar{f} : M(A) \to N$ such that $\|\bar{f}\| \circ i = f$. This is given by the following diagrams:*

In the category of monoids:

$$\begin{tikzcd} M(A) \arrow[r, dotted, "\bar{f}"] & N \end{tikzcd}$$

in the category of Sets:

$$\begin{tikzcd} A \arrow[r, "i"] \arrow[dr, "f"'] & M(A) \arrow[d, "\|\bar{f}\|"] \\ & \|N\| \end{tikzcd}$$

Since we have considered the forgetful functor as the optimal way of obtaining a set from a monoid, one can consider a free monoid as the optimal way of obtaining a monoid from a set. Together, those two functors makes a pair of adjoints.

Based on that, we can define adjoints formally already:

Definition 3.18 (Adjunction (AWODEY, 2010))**.** *An adjuction between categories* ***C*** *and* ***D*** *are a pair of functors:*

$$\begin{tikzcd} F : C \arrow[r, shift left, "s"] & D : U \arrow[l, shift left, "t"] \end{tikzcd}$$

and a natural transformation:

$$\eta : 1_C \to U \circ F$$

that follows the diagrams:

$$\begin{tikzcd} F(C) \arrow[r, dotted, "g"] & D \end{tikzcd}$$

$$\begin{tikzcd} C \arrow[r, "\eta_C"] \arrow[dr, "f"'] & U(F(C)) \arrow[d, "U(g)"] \\ & U(D) \end{tikzcd}$$

In the above definition, F is called the left adjoint, U the right adjoint and η is the unit of the adjunction. In the monoid example, the left adjoint is the free monoid generator and the right adjoint is the forgetful functor.

3.4 HIGHER CATEGORIES

In this section, we introduce only the concepts of higher category theory that will be necessary to prove our results in the next chapter.

Previously in this chapter, we have said that sometimes the arrows between two objects do not form only a set, but a category instead. In that case, we have categories between every pair of objects. In this sense, we can think that we have added a new dimension to the category. Indeed, this process can continue. We can take a category, then a category between pair of objects and go even further, taking a category between pairs of objects of this inner category. This process can add infinitely many dimensions. When this process goes up to the infinity, we have an $\infty - groupoid$.

Another important concept is the fact that sometimes the equalities of a category does not hold "on the nose", but only in a weak sense. For example, we have seen the difference between definitional and propositional equalities in type theory. In this theory, the equalities of a category induced by a type do not hold in the strict sense of definitional equalities, but only hold up to propositional equality, thus they are weak structures. In fact, weak structures are much more common in mathematics than strict ones. Thus, we are going to be interested in weak higher order categories.

Moreover, it is important to notice that we are going to limit the scope of this work to weak categories of the second order, called bicategories. Obtaining a weak ∞-groupoid would be a great achievement, but since it involves many highly complex concepts of higher category theory, trying to achieve this result would be unpractical.

With that said, we can finally define some basic concepts of higher category theory.

3.4.1 Globular Sets

The first important concept is globular set:

Definition 3.19 (Globular Sets (LEINSTER, 2004))**.** *Let $n \in \mathbb{N}$. A globular set X is the following diagram of sets and functions:*

$$\begin{tikzcd} X(n) \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & X(n-1) \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & \cdots \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & X(0) \end{tikzcd}$$

The diagram must respect the following equations for any $m \in \{2, ..., n\}$ and $x \in X(m)$:

$$s(s(x)) = s(t(x)), \quad t(s(x)) = t(t(x))$$

The terms $x \in X(n)$ are called n-cells.

3.4.2 Horizontal Composition

Given any globular set, a 0-cell is simply an object and is represented by the following diagram:

$$\begin{tikzcd}
a
\end{tikzcd}$$

An 1-cell is a morphism between objects, represented by:

$$\begin{tikzcd}
a \arrow[r, "s"] & b
\end{tikzcd}$$

2-cells are morphisms between 1-cells. Therefore, are represented by the following diagram:

$$\begin{tikzcd}
a \arrow[r, bend left=50, "s", ""{name=U, below}] \arrow[r, bend right=50, "t"', ""{name=D}] & b
\arrow[Rightarrow, from=U, to=D, "\alpha"]
\end{tikzcd}$$

Since a 2-cell has a higher dimension that 1-cells, it is possible to compose 2-cells in two different ways. The first way is the traditional composition written as $\circ$. The diagram

$$\begin{tikzcd}
a \arrow[r, bend left=60, "s", ""{name=U, below}] \arrow[r, "\mathrm{t}"', ""{name=M}] \arrow[r, bend right=60, "x"', ""{name=D}] & b
\arrow[Rightarrow, from=U, to=M, "\alpha"]
\arrow[Rightarrow, from=M, to=D, "\chi"]
\end{tikzcd}$$

can be represented by:

$$\begin{tikzcd}
a \arrow[r, bend left=50, "s", ""{name=U, below}] \arrow[r, bend right=50, "x"', ""{name=D}] & b
\arrow[Rightarrow, from=U, to=D, "\chi \circ \alpha"]
\end{tikzcd}$$

In a 2-cell, the traditional composition is also known as vertical composition. For 2-cells, we have an additional composition written as $\circ_h$. It is called horizontal composition (LEINSTER, 2004). It is represented by transforming the following diagram

$$\begin{tikzcd}
a \arrow[r, bend left=50, "s"{name=U}] \arrow[r, bend right=50, "x"'{name=D}] \arrow[Rightarrow, from=U, to=D, "\alpha"] & b \arrow[r, bend left=50, "t"{name=U2}] \arrow[r, bend right=50, "y"'{name=D2}] \arrow[Rightarrow, from=U2, to=D2, "\chi"] & c
\end{tikzcd}$$

into the diagram:

$$\begin{tikzcd}
a \arrow[r, bend left=50, "t \circ s"{name=U}] \arrow[r, bend right=50, "y \circ x"'{name=D}] \arrow[Rightarrow, from=U, to=D, "\chi \circ_h \alpha"] & c
\end{tikzcd}$$

Thus, when one works with categories of order higher than 1, one needs to take into account those new kind of compositions. In the case of 2-categories, the 2-cells will have those two compositions. Moreover, the horizontal composition must also respects associativity and the identity laws. Analogously, 3-cells, in addition of the two compositions present in 2-cells, has a third one. Indeed, a n-cell has n compositions. Formally, m compositions of a m-cell $\in A(m)$ is represented by (LEINSTER, 2004):

$$A(m) \times_{A(p)} A(m) = \{(x', x) \in A(m) \times A(m) \mid t^{m-p}(x) = s^{m-p}(x')\}, \text{ with } p \leq m$$

3.4.3 Bicategories

Bicategories are weak second order categories. They are defined as follows:

Definition 3.20 (Bicategories (LEINSTER, 1998))**.** *A bicategory Γ is a structure with the following data and axioms:*
Data:

- *Objects collection ob Γ (Elements are 0-cells $A, B, ...$)*
- *Categories $\Gamma(A, B)$ (1-cells $f, g, ...$ are the objects of those categories and arrows are 2-cells $\alpha, \beta, ...$)*
- *Functors:*

$$c_{ABC} : \Gamma(B,C) \times \Gamma(A,B) \to \Gamma(A,C)$$
$$(g,f) \to g \circ f = gf$$
$$(\beta,\alpha) \to \beta \circ_h \alpha$$
and $I_A : 1 \to \Gamma(A,A)$ *(thus,* I_A *is an* 1*-cell* $A \to A$*)*

- *The following natural isomorphisms:*

$$\begin{tikzcd}
\mathcal{B}(C,D) \times \mathcal{B}(B,C) \times \mathcal{B}(A,B) \arrow[r, "1 \times c_{ABC}"] \arrow[d, "c_{BCD} \times 1"'] \arrow[dr, phantom, "\nearrow a_{ABCD}"] & \mathcal{B}(C,D) \times \mathcal{B}(A,C) \arrow[d, "c_{ACD}"] \\
\mathcal{B}(B,D) \times \mathcal{B}(A,B) \arrow[r, "c_{ABD}"'] & \mathcal{B}(A,D)
\end{tikzcd}$$

$$\begin{tikzcd}
\mathcal{B}(A,B) \times \mathbf{1} \arrow[d, "1 \times I_A"'] \arrow[dr, "\sim"] \arrow[dr, phantom, "r_{AB}\nearrow"'] & \\
\mathcal{B}(A,B) \times \mathcal{B}(A,A) \arrow[r, "c_{AAB}"'] & \mathcal{B}(A,B)
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbf{1} \times \mathcal{B}(A,B) \arrow[d, "I_B \times 1"'] \arrow[dr, "\sim"] \arrow[dr, phantom, "l_{AB}\nearrow"'] & \\
\mathcal{B}(B,B) \times \mathcal{B}(A,B) \arrow[r, "c_{ABB}"'] & \mathcal{B}(A,B)
\end{tikzcd}$$

Thus, we have 2*-cells:*

$$a_{hgf} : (hg)f \xrightarrow{\sim} h(gf)$$
$$r_f : f \circ I_A \xrightarrow{\sim} f$$
$$l_f : I_B \circ f \xrightarrow{\sim} f$$

Axioms:
Consider the following configuration of 1*-cells*

$$\begin{tikzcd}
a \arrow[r, "s"] & b \arrow[r, "r"] & c \arrow[r, "p"] & d \arrow[r, "u"] & e
\end{tikzcd}$$

The following diagrams commute:

$$\begin{tikzcd}
 & ((u \circ p) \circ r) \circ s \arrow[rr, "a \circ_h 1"'] \arrow[dl, "a"'] & & (u \circ (p \circ r)) \circ s \arrow[dr, "a"] & \\
(u \circ p) \circ (r \circ s) \arrow[drr, "a"'] & & & & u \circ ((p \circ r) \circ s) \arrow[dll, "1 \circ_h a"] \\
 & & u \circ (p \circ (r \circ s)) & &
\end{tikzcd}$$

$$\begin{tikzcd}
(r \circ \rho_b) \circ s \arrow[rr, "a"'] \arrow[dr, "r_r \circ_h 1"'] & & r \circ (\rho_b \circ s) \arrow[dl, "1 \circ_h l_s"] \\
 & r \circ s &
\end{tikzcd}$$

In the literature of higher order category theory, the axioms of a weak higher order category are called coherence laws (LEINSTER, 2004).

3.5 CONCLUSION

In this chapter, we have seen the basic concepts of category theory. Most will play an important role in the next chapter. The concepts of isomorphism and groupoid, for instance, will be the base for mathematical model for computational paths. We have also seen concepts of higher category theory, since in the sequel we are going to show that computational paths forms a higher structure.

4 COMPUTATIONAL PATHS

In this chapter, we introduce the main concept of this work, an entity known as computational paths. In **chapter 2**, we have seen that it is possible to interpret the identity type semantically, considering the terms as homotopical paths between two points of a space. Nevertheless, there is no entity that represents those paths in the syntax of type theory. Thus, we add computational paths to fill this gap. In this sense, our objective is to formulate the identity type using this new entity.

This work has also been motivated by the fact that although beautifully defined, we have noticed that proofs that uses the identity type can be sometimes a little too complex. The elimination rule of the intensional identity type encapsulates lots of information, sometimes making too troublesome the process of finding the reason that builds the correct type. We have also seen this in **chapter 2**. We proposed constructions for the symmetry and transitivity of the identity type. Although one could expect those constructions to be pretty straightforward, the process of obtaining the right reason to construct those types was troublesome.

Thus, inspired by the path-based approach of the homotopy interpretation, we believe that a similar approach can be used to define the identity type in type theory. To achieve that, we have been using a notion of *computational paths*. The interpretation will be similar to the homotopy one: a term $p : Id_A(a, b)$ will be a computational path between terms $a, b : A$, and such path will be the result of a sequence of rewrites. In the sequel, we shall define formally the concept of a computational path. The main idea, i.e. proofs of equality statements as (reversible) sequences of rewrites, is not new, as it goes back to a paper entitled “Equality in labeled deductive systems and the functional interpretation of propositional equality ", presented in December 1993 at the *9th Amsterdam Colloquium*, and published in the proceedings in 1994(QUEIROZ; GABBAY, 1994).

In the previous chapter, we have seen the concept of higher structures such as higher categories. Indeed, one of the most interesting aspects of the identity type is the fact that it can be used to construct higher structures. This is a rather natural consequence of the fact that it is possible to construct higher identities. For any $a, b : A$, we have type $Id_A(a, b)$. If this type is inhabited by any $p, q : Id_A(a, b)$, then we have type $Id_{Id_A(a,b)}(p, q)$. If the latter type is inhabited, we have a higher equality between p and q(HARPER, 2012). This concept is also present in computational paths. One can show the equality between two computational paths s and t by constructing a third one between s and t. We show in this chapter a

This chapter is based on two papers published by the author (jointly with his advisor and co-advisor): *Propositional equality, identity types, and computational paths* in *South American Joournal of Logic*, 2016 (QUEIROZ; OLIVEIRA; RAMOS, 2016) and *On the Identity Type as the Type of Computational Paths* in *Logic Journal of the IGPL*, 2017 (RAMOS; QUEIROZ; OLIVEIRA, 2017).

system of rules used to establish equalities between computational paths(OLIVEIRA, 1995). Then, we show that these higher equalities go up to the infinity, forming a ∞-globular-set. We also show that computational paths naturally induce a groupoid structure. We also go a step further, showing that computational paths are capable of inducing a higher groupoid structure.

Another important question we want to answer is one that arises naturally when talking about equality: Is there a canonical proof for an expression $t_1 = t_2$? In the language of computational paths, is there a normal path between $t_1 = t_2$ such that every other path can be reduced to this one? In fact, we are going to prove that the answer is negative. Our model also refute the Uniquiness of Identity Proofs (UIP)).

4.1 INTRODUCING COMPUTATIONAL PATHS

Before we enter in details of computational paths, let's recall what motivated the introduction of computational paths to type theory. In type theory, our types are interpreted using the so-called Brower-Heyting-Kolmogorov Interpretation. That way, a semantic interpretation of types are not given by truth-values, but by the concept of proof as a primitive notion. Thus, we have (QUEIROZ; OLIVEIRA; RAMOS, 2016):

a proof of the proposition:	**is given by:**
$A \wedge B$	a proof of A **and** a proof of B
$A \vee B$	a proof of A **or** a proof of B
$A \to B$	a **function** that turns a proof of A into a proof of B
$\forall x^D.P(x)$	a **function** that turns an element a into a proof of $P(a)$
$\exists x^D.P(x)$	an element a (witness) **and a proof of** $P(a)$

Also, based on the Curry-Howard functional interpretation of logical connectives, one have (QUEIROZ; OLIVEIRA; RAMOS, 2016):

a proof of the proposition:	**has the canonical form of:**
$A \wedge B$	$\langle p, q \rangle$ where p is a proof of A and q is a proof of B
$A \vee B$	$i(p)$ where p is a proof of A or $j(q)$ where q is a proof of B ('i' and 'j' abbreviate 'into the left/right disjunct')
$A \to B$	$\lambda x.b(x)$ where $b(p)$ is a proof of B provided p is a proof of A
$\forall x^A.B(x)$	$\Lambda x.f(x)$ where $f(a)$ is a proof of $B(a)$ provided a is an arbitrary individual chosen from the domain A
$\exists x^A.B(x)$	$\varepsilon x.(f(x), a)$ where a is a witness from the domain A, $f(a)$ is a proof of $B(a)$

If one looks closely, there is one interpretation missing in the BHK-Interpretation. What constitutes a proof of $t_1 = t_2$? In other words, what is a proof of an equality statement? We answer this by proposing that an equality between those two terms should be a sequence of rewrites starting from t_1 and ending at t_2. Thus, we would have (QUEIROZ; OLIVEIRA; RAMOS, 2016):

a proof of the proposition:	**is given by:**
$t_1 = t_2$	? (Perhaps a sequence of rewrites starting from t_1 and ending in t_2?)

We call computational path the sequence of rewrites between these terms.

4.1.1 Formal Definition

Since computational path is a generic term, it is important to emphasize the fact that we are using the term computational path in the sense defined by(QUEIROZ; OLIVEIRA, 2014a). A computational path is based on the idea that it is possible to formally define when two computational objects $a, b : A$ are equal. These two objects are equal if one can reach b from a by applying a sequence of axioms or rules. This sequence of operations forms a path. Since it is between two computational objects, it is said that this path is a computational one. Also, an application of an axiom or a rule transforms (or rewrite) an term in another. For that reason, a computational path is also known as a sequence of rewrites. Nevertheless, before we define formally a computational path, we can take a look at one famous equality theory, the $\lambda\beta\eta - equality$(HINDLEY; SELDIN, 2008):

Definition 4.1. *The* $\lambda\beta\eta$-equality *is composed by the following axioms:*

(α) $\lambda x.M = \lambda y.M[y/x]$ *if* $y \notin FV(M)$;

(β) $(\lambda x.M)N = M[N/x]$;

(ρ) $M = M$;

(η) $(\lambda x.Mx) = M \quad (x \notin FV(M))$.

And the following rules of inference:

$$(\mu) \quad \frac{M = M'}{NM = NM'} \qquad (\tau)\, \frac{M = N \qquad N = P}{M = P}$$

$$(\nu)\quad \frac{M = M'}{MN = M'N} \qquad (\sigma)\ \frac{M = N}{N = M}$$

$$(\xi)\quad \frac{M = M'}{\lambda x.M = \lambda x.M'}$$

Definition 4.2. *(HINDLEY; SELDIN, 2008) P is β-equal or β-convertible to Q (notation $P =_\beta Q$) iff Q is obtained from P by a finite (perhaps empty) series of β-contractions and reversed β-contractions and changes of bound variables. That is, $P =_\beta Q$ iff* ***there exist*** *$P_0, \dots, P_n$ $(n \geq 0)$ such that $P_0 \equiv P$, $P_n \equiv Q$, $(\forall i \leq n-1)(P_i \rhd_{1\beta} P_{i+1}$ or $P_{i+1} \rhd_{1\beta} P_i$ or $P_i \equiv_\alpha P_{i+1})$.*

(Note that equality has an **existential** force, which will show in the proof rules for the identity type.)

The same happens with $\lambda\beta\eta$-equality:

Definition 4.3. *($\lambda\beta\eta$-equality(HINDLEY; SELDIN, 2008)) The equality-relation determined by the theory $\lambda\beta\eta$ is called $=_{\beta\eta}$; that is, we define*

$$M =_{\beta\eta} N \quad \Leftrightarrow \quad \lambda\beta\eta \vdash M = N.$$

Example 4.1. *Take the term $M \equiv (\lambda x.(\lambda y.yx)(\lambda w.zw))v$. Then, it is $\beta\eta$-equal to $N \equiv zv$ because of the sequence:*
$(\lambda x.(\lambda y.yx)(\lambda w.zw))v, \quad (\lambda x.(\lambda y.yx)z)v, \quad (\lambda y.yv)z, \quad zv$
which starts from M and ends with N, and each member of the sequence is obtained via 1-step β- or η-contraction of a previous term in the sequence. To take this sequence into a path, *one has to apply transitivity twice, as we do in the example below.*

Example 4.2. *The term $M \equiv (\lambda x.(\lambda y.yx)(\lambda w.zw))v$ is $\beta\eta$-equal to $N \equiv zv$ because of the sequence:*
$(\lambda x.(\lambda y.yx)(\lambda w.zw))v, \quad (\lambda x.(\lambda y.yx)z)v, \quad (\lambda y.yv)z, \quad zv$
Now, taking this sequence into a path leads us to the following:
The first is equal to the second based on the grounds:
$\eta((\lambda x.(\lambda y.yx)(\lambda w.zw))v, (\lambda x.(\lambda y.yx)z)v)$
The second is equal to the third based on the grounds:
$\beta((\lambda x.(\lambda y.yx)z)v, (\lambda y.yv)z)$
Now, the first is equal to the third based on the grounds:
$\tau(\eta((\lambda x.(\lambda y.yx)(\lambda w.zw))v, (\lambda x.(\lambda y.yx)z)v), \beta((\lambda x.(\lambda y.yx)z)v, (\lambda y.yv)z))$
Now, the third is equal to the fourth one based on the grounds:
$\beta((\lambda y.yv)z, zv)$
Thus, the first one is equal to the fourth one based on the grounds:
$\tau(\tau(\eta((\lambda x.(\lambda y.yx)(\lambda w.zw))v, (\lambda x.(\lambda y.yx)z)v), \beta((\lambda x.(\lambda y.yx)z)v, (\lambda y.yv)z)), \beta((\lambda y.yv)z, zv)))$.

The aforementioned theory establishes the equality between two λ-terms. Since we are working with computational objects as terms of a type, we need to translate the $\lambda\beta\eta$-equality to a suitable equality theory based on Martin Löf's type theory. We obtain:

Definition 4.4. *The equality theory of Martin Löf's type theory has the following basic proof rules for the* Π*-type:*

$$(\beta)\quad \frac{N : A \qquad \overset{[x:A]}{M : B}}{(\lambda x.M)N = M[N/x] : B[N/x]} \qquad (\xi)\quad \frac{\overset{[x:A]}{M = M' : B}}{\lambda x.M = \lambda x.M' : (\Pi x : A)B}$$

$$(\rho)\quad \frac{M : A}{M = M : A} \qquad (\mu)\quad \frac{M = M' : A \qquad N : (\Pi x : A)B}{NM = NM' : B[M/x]}$$

$$(\sigma)\quad \frac{M = N : A}{N = M : A} \qquad (\nu)\quad \frac{N : A \qquad M = M' : (\Pi x : A)B}{MN = M'N : B[N/x]}$$

$$(\tau)\quad \frac{M = N : A \qquad N = P : A}{M = P : A}$$

$$(\eta)\quad \frac{M : (\Pi x : A)B}{(\lambda x.Mx) = M : (\Pi x : A)B}\,(x \notin FV(M))$$

We are finally able to formally define computational paths:

Definition 4.5. *Let a and b be elements of a type A. Then, a* computational path *s from a to b is a composition of rewrites (each rewrite is an application of the inference rules of the equality theory of type theory or is a change of bound variables). We denote that by $a =_s b$.*

As we have seen in *example 4.2*, composition of rewrites are applications of the rule τ. Since change of bound variables is possible, each term is considered up to α-equivalence.

4.1.2 Equality Equations

One can use the aforementioned axioms to show that computational paths establishes the three fundamental equations of equality: the reflexivity, symmetry and transitivity:

$$\frac{a =_t b : A \qquad b =_u c : A}{a =_{\tau(t,u)} c : A}\ \textit{transitivity} \qquad \frac{a : A}{a =_\rho a : A}\ \textit{reflexivity}$$

$$\frac{a =_t b : A}{b =_{\sigma(t)} a : A}\ \textit{symmetry}$$

4.1.3 Identity Type

We have said that it is possible to formulate the identity type using computational paths. As we have seen, the best way to define any formal entity of type theory is by a set of natural deductions rules. Thus, we define our path-based approach as the following set of rules:

- Formation and Introduction rules (QUEIROZ; OLIVEIRA; RAMOS, 2016; RAMOS; QUEIROZ; OLIVEIRA, 2017):

$$\frac{A \text{ type} \qquad a : A \qquad b : A}{Id_A(a,b) \text{ type}}\; Id - F$$

$$\frac{a =_s b : A}{s(a,b) : Id_A(a,b)}\; Id - I$$

  One can notice that our formation rule is exactly equal to the traditional identity type. From terms $a, b : A$, one can form that is inhabited only if there is a proof of equality between those terms, i.e., $Id_A(a,b)$.

  The difference starts with the introduction rule. In our approach, one can notice that we do not use a reflexive constructor r. In other words, the reflexive path is not the main building block of our identity type. Instead, if we have a computational path $a =_s b : A$, we introduce $s(a,b)$ as a term of the identity type. That way, one should see $s(a,b)$ as a sequence of rewrites and substitutions (i.e., a computational path) which would have started from a and arrived at b

- Elimination rule (QUEIROZ; OLIVEIRA; RAMOS, 2016; RAMOS; QUEIROZ; OLIVEIRA, 2017):

$$\frac{m : Id_A(a,b) \qquad \begin{array}{c}[a =_g b : A] \\ h(g) : C\end{array}}{REWR(m, \acute{g}.h(g)) : C}\; Id - E$$

  Let's recall the notation being used. First, one should see $h(g)$ as a functional expression h which depends on g. Also, one should notice the use of '$\acute{}$' in $\acute{g}$. One should see '$\acute{}$' as an abstractor that binds the occurrences of the variable g introduced in the local assumption $[a =_g b : A]$ as a kind of *Skolem-type* constant denoting the *reason* why a was assumed to be equal to b.

  We also introduce the constructor $REWR$. In a sense, it is similar to the constructor J of the traditional approach, since both arise from the elimination rule of the identity type. The behavior of $REWR$ is simple. If from a computational path g that establishes the equality between a and b one can construct $h(g) : C$, then if we also have this equality established by a term $m : Id_A(a,b)$, we can put together all this information in $REWR$ to construct C, eliminating the type $Id_A(a,b)$ in the process. The idea is that we can substitute g for m in $\acute{g}.h(g)$, resulting in $h(m/g) : C$. This behavior is established next by the reduction rule.

- Reduction rule (QUEIROZ; OLIVEIRA; RAMOS, 2016; RAMOS; QUEIROZ; OLIVEIRA, 2017):

$$\dfrac{\dfrac{a =_m b : A}{m(a,b) : Id_A(a,b)}\,Id-I \qquad \begin{matrix}[a =_g b : A] \\ h(g) : C\end{matrix}}{REWR(m, \acute{g}.h(g)) : C}\,Id-E \quad \rhd_\beta \quad \begin{matrix}[a =_m b : A] \\ h(m/g) : C\end{matrix}$$

- Induction rule:

$$\dfrac{e : Id_A(a,b) \qquad \dfrac{[a =_t b : A]}{t(a,b) : Id_A(a,b)}\,Id-I}{REWR(e, \acute{t}.t(a,b)) : Id_A(a,b)}\,Id-E \quad \rhd_\eta \quad e : Id_A(a,b)$$

Our introduction and elimination rules reassure the concept of equality as an **existential force**. In the introduction rule, we encapsulate the idea that a witness of a identity type $Id_A(a,b)$ only exists if there exist a computational path establishing the equality of a and b. Also, one can notice that our elimination rule is similar to the elimination rule of the existential quantifier.

4.1.4 Path-based Examples

The objective of this subsection is to show how to use in practice the rules that we have just defined. The idea is to show construction of terms of some important types. The constructions that we have chosen to build are the reflexive, transitive and symmetric type of the identity type. Those were not random choices. The main reason is the fact that reflexive, transitive and symmetric types are essential to the process of building a groupoid model for the identity type(HOFMANN; STREICHER, 1994). As we shall see, these constructions come naturally from simple computational paths constructed by the application of axioms of the equality of type theory.

Before we start the constructions, we think that it is essential to understand how to use the eliminations rules. The process of building a term of some type is a matter of finding the right reason. In the case of J, the reason is the correct $x, y : A$ and $z : Id_A(a,b)$ that generates the adequate $C(x,y,z)$. In our approach, the reason is the correct path $a =_g b$ that generates the adequate $g(a,b) : Id(a,b)$.

4.1.4.1 Reflexivity

One could find strange the fact that we need to prove the reflexivity. Nevertheless, just remember that our approach is not based on the idea that reflexivity is the base of the identity type. As usual in type theory, a proof of something comes down to a construction of a term of a type. In this case, we need to construct a term of type $\Pi_{(a:A)}Id_A(a,a)$. The reason is extremely simple: from a term $a : A$, we obtain the computational path $a =_\rho a : A$ (RAMOS; QUEIROZ; OLIVEIRA, 2017):

$$\cfrac{\cfrac{\cfrac{[a : A]}{a =_\rho a : A}}{\rho(a,a) : Id_A(a,a)}\;Id - I}{\lambda a.\rho(a,a) : \Pi_{(a:A)}Id_A(a,a)}\;\Pi - I$$

4.1.4.2 Symmetry

The second proposed construction is the symmetry. Our objective is to obtain a term of type $\Pi_{(a:A)}\Pi_{(b:A)}(Id_A(a,b) \rightarrow Id_A(b,a))$.

We construct a proof using computational paths. As expected, we need to find a suitable reason. Starting from $a =_t b$, we could look at the axioms of *definition 4.1* to plan our next step. One of those axioms makes the symmetry clear: the σ axiom. If we apply σ, we will obtain $b =_{\sigma(t)} a$. From this, we can then infer that Id_A is inhabited by $(\sigma(t))(b,a)$. Now, it is just a matter of applying the elimination (RAMOS; QUEIROZ; OLIVEIRA, 2017):

$$\cfrac{\cfrac{\cfrac{\cfrac{\begin{matrix}[a : A] \quad [b : A] \\ [p(a,b) : Id_A(a,b)]\end{matrix} \qquad \cfrac{\cfrac{[a =_t b : A]}{b =_{\sigma(t)} a : A}}{(\sigma(t))(b,a) : Id_A(b,a)}\;Id - I}{REWR(p(a,b), \acute{t}.(\sigma(t))(b,a)) : Id_A(b,a)}\;Id - E}{\lambda p.REWR(p(a,b), \acute{t}.(\sigma(t))(b,a)) : Id_A(a,b) \rightarrow Id_A(b,a)}\;\rightarrow -I}{\lambda b.\lambda p.REWR(p(a,b), \acute{t}.(\sigma(t))(b,a)) : \Pi_{(b:A)}(Id_A(a,b) \rightarrow Id_A(b,a))}\;\Pi - I}{\lambda a.\lambda b.\lambda p.REWR(p(a,b), \acute{t}.(\sigma(t))(b,a)) : \Pi_{(a:A)}\Pi_{(b:A)}(Id_A(a,b) \rightarrow Id_A(b,a))}\;\Pi - I$$

4.1.4.3 Transitivity

The third and last construction will be the transitivity. Our objective os to obtain a term of type $\Pi_{(a:A)}\Pi_{(b:A)}\Pi_{(c:A)}(Id_A(a,b) \rightarrow Id_A(b,c) \rightarrow Id_A(a,c))$.

To build our path-based construction, the first step, as expected, is to find the reason. Since we are trying to construct the transitivity, it is natural to think that we should start with paths $a =_t b$ and $b =_u c$ and then, from these paths, we should conclude that there is a path z that establishes that $a =_z c$. To obtain z, we could try to apply the axioms of *definition 4.1*. Looking at the axioms, one is exactly what we want: the axiom τ. If we apply τ to $a =_t b$ and $b =_u c$, we will obtain a new path $\tau(t, u)$ such that $a =_{\tau(t,u)} c$. Using that construction as the reason, we obtain the following term (RAMOS; QUEIROZ; OLIVEIRA, 2017):

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}[a:A] \quad [b:A] \\ [w(a,b):Id_A(a,b)]\end{matrix} \qquad \dfrac{\begin{matrix}[c:A] \\ [s(b,c):Id_A(b,c)]\end{matrix} \qquad \dfrac{\dfrac{[a =_t b : A] \qquad [b =_u c : A]}{a =_{\tau(t,u)} c : A}}{(\tau(t,u))(a,c) : Id_A(a,c)}\,Id-I}{REWR(s(b,c), \acute{u}(\tau(t,u))(a,c)) : Id_A(a,c)}\,Id-E}{REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : Id_A(a,c)}\,Id-E}{\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : Id_A(b,c) \to Id_A(a,c)}\,\to -I}{\lambda w.\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c)}\,\to -I}{\lambda c.\lambda w.\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : \Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))}\,\Pi - I}{\lambda b.\lambda c.\lambda w.\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : \Pi_{(b:A)}\Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))}\,\Pi - I}{\lambda a.\lambda b.\lambda c.\lambda w.\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : \Pi_{(a:A)}\Pi_{(b:A)}\Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))}\,\Pi - I
$$

As one can see, each step is just straightforward applications of introduction, elimination rules and abstractions. The only idea behind this construction is just the simple fact that the axiom τ guarantees the transitivity of paths. If one compares the reason of this construction to the one that used J (we showed those constructions in chapter 2), one can clearly conclude that the reason of the path-based approach was obtained more naturally.

4.2 TERM REWRITE SYSTEM

As we have just shown, a computational path establishes when two terms of the same type are equal. From the theory of computational paths, an interesting case arises. Suppose we have a path s that establishes that $a =_s b : A$ and a path t that establishes that $a =_t b : A$. Consider that s and t are formed by distinct compositions of rewrites. Is it possible to conclude that there are cases that s and t should be considered equivalent? The answer is *yes*. Consider the following examples (RAMOS, 2015):

Example 4.3. Consider the path $a =_t b : A$. By the symmetric property, we obtain $b =_{\sigma(t)} a : A$. What if we apply the property again on the path $\sigma(t)$? We would obtain a path $a =_{\sigma(\sigma(t))} b : A$. Since we applied symmetry twice in succession, we obtained a path that is equivalent to the initial path t. For that reason, we conclude the act of applying symmetry twice in succession is a redundancy. We say that the path $\sigma(\sigma(t))$ should be reduced to the path t.

Example 4.4. Consider the reflexive path $a =_\rho a : A$. It one applies the symmetric axiom, one ends up with $a =_{\sigma(\rho)} a : A$. Thus, the obtained path is equivalent to the initial one, since the symmetry was applied to the reflexive path. Therefore, $\sigma(\rho)$ is a redundant way of expressing the path ρ. Thus, $\sigma(\rho)$ should be reduced to ρ.

Example 4.5. Consider a path $a =_t b : A$. Applying the symmetry, one ends up with $b =_{\sigma(t)} a : A$. One can take those two paths and apply the transitivity, ending up with $a =_{\tau(t,\sigma(t))} a$. Since one path is the inverse of the other, the composition of those two paths should be equivalent to the reflexive path. Thus, $\tau(t, \sigma(t))$ should be reduced to ρ.

As one could see in the aforementioned examples, different paths should be considered equal if one is just a redundant form of the other. The examples that we have just seen are just straightforward and simple cases. Since the equality theory has a total of 7 axioms, the possibility of combinations that could generate redundancies are high. Fortunately, all possible redundancies were thoroughly mapped by(OLIVEIRA, 1995). In that work, a system that establishes all redundancies and creates rules that solve them was proposed. This system, known as $LND_{EQ} - TRS$, maps a total of 39 rules that solve redundancies.

4.2.1 $LND_{EQ} - TRS$

In this subsection, we show the rules that compose the $LND_{EQ} - TRS$. All those rules comes from the mapping of redundancies between computational paths, as we have seen in the 3 previous examples.

4.2.1.1 Subterm Substitution

Before we introduce the rewriting rules, it is important to introduce the concept of subterm substitution. In Equational Logic, the subterm substitution is given by the following inference rule(OLIVEIRA; QUEIROZ, 1994):

$$\frac{s = t}{s\theta = t\theta}$$

One problem is that such rule does not respect the sub-formula property. To deal with that,(CHENADEC, 1989) proposes two inference rules:

$$\frac{M = N \qquad C[N] = O}{C[M] = O}\ IL \qquad\qquad \frac{M = C[N] \qquad N = O}{M = C[O]}\ IR$$

where M, N and O are terms.

As proposed in (QUEIROZ; OLIVEIRA; RAMOS, 2016), we can define similar rules using computational paths, as follows:

$$\frac{x =_r \mathcal{C}[y] : A \qquad y =_s u : A'}{x =_{\mathtt{sub_L}(r,s)} \mathcal{C}[u] : A} \qquad \frac{x =_r w : A' \qquad \mathcal{C}[w] =_s u : A}{\mathcal{C}[x] =_{\mathtt{sub_R}(r,s)} u : A}$$

where C is the context in which the sub-term detached by '[ ]' appears and A' could be a sub-domain of A, equal to A or disjoint to A.

In the rule above, $\mathcal{C}[u]$ should be understood as the result of replacing every occurrence of y by u in C.

4.2.1.2 Rewriting Rules

In this subsection, our objective is to show all rewrite reductions and their associated rewriting rules. The idea is to analyze all possible occurrences of redundancies in proofs which involves the rules of rewritings.

We start with the transitivity:

Definition 4.6 (reductions involving τ (QUEIROZ; OLIVEIRA; RAMOS, 2016))**.**

$$\frac{x =_r y : A \quad y =_{\sigma(r)} x : A}{x =_{\tau(r,\sigma(r))} x : A} \quad \rhd_{tr} \quad x =_\rho x : A$$

$$\frac{y =_{\sigma(r)} x : A \quad x =_r y : A}{y =_{\tau(\sigma(r),r)} y : A} \quad \rhd_{tsr} \quad y =_\rho y : A$$

$$\frac{u =_r v : A \quad v =_\rho v : A}{u =_{\tau(r,\rho)} v : A} \quad \triangleright_{trr} \quad u =_r v : A$$

$$\frac{u =_\rho u : A \quad u =_r v : A}{u =_{\tau(\rho,r)} v : A} \quad \triangleright_{tlr} \quad u =_r v : A$$

Associated rewriting rules:
$\tau(r, \sigma(r)) \triangleright_{tr} \rho$
$\tau(\sigma(r), r) \triangleright_{tsr} \rho$
$\tau(r, \rho) \triangleright_{trr} r$
$\tau(\rho, r) \triangleright_{tlr} r.$

These reductions can be generalized to transformations where the reasons r and $\sigma(r)$ (transf. 1 and 2) and r and ρ (transf. 3 and 4) appear in some context, as illustrated by the following example: (QUEIROZ; OLIVEIRA; RAMOS, 2016):

Example 4.6.

$$\frac{\dfrac{x =_r y : A}{i(x) =_{\xi_1(r)} i(y) : A + B} \quad \dfrac{\dfrac{x =_r y : A}{y =_{\sigma(r)} x : A}}{i(y) =_{\xi_1(\sigma(r))} i(x) : A + B}}{i(x) =_{\tau(\xi_1(r), \xi_1(\sigma(r)))} i(x) : A + B} \quad \triangleright_{tr} \quad \frac{x =_r y : A}{i(x) =_{\xi_1(r)} i(y) : A + B}$$

Associated rewriting:
$\tau(\xi_1(r), \xi_1(\sigma(r))) \triangleright_{tr} \xi_1(r).$

For the general context $\mathcal{C}[\ ]$:
Associated rewritings:
$\tau(\mathcal{C}[r], \mathcal{C}[\sigma(r)]) \triangleright_{tr} \mathcal{C}[\rho]$
$\tau(\mathcal{C}[\sigma(r)], \mathcal{C}[r]) \triangleright_{tsr} \mathcal{C}[\rho]$
$\tau(\mathcal{C}[r], \mathcal{C}[\rho]) \triangleright_{trr} \mathcal{C}[r]$
$\tau(\mathcal{C}[\rho], \mathcal{C}[r]) \triangleright_{tlr} \mathcal{C}[r]$

The transitivity rules are pretty straightforward. We have more complicated cases (QUEIROZ; OLIVEIRA; RAMOS, 2016):

Definition 4.7.

$$\dfrac{a : A \quad \dfrac{\begin{matrix}[x : A] \\ \vdots \\ b(x) =_r g(x) : B\end{matrix}}{\lambda x.b(x) =_{\xi(r)} \lambda x.g(x) : A \to B} \to\text{-}intr}{APP(\lambda x.b(x), a) =_{\nu(\xi(r))} APP(\lambda x.g(x), a) : B} \to\text{-}elim \qquad \rhd_{mxl} \qquad \begin{matrix} a : A \\ b(a/x) =_r g(a/x) : B \end{matrix}$$

Associated rewriting rule:

$\nu(\xi(r)) \rhd_{mxl} r$.

Definition 4.8 (reductions involving ρ and σ (QUEIROZ; OLIVEIRA; RAMOS, 2016)).

$$\dfrac{x =_\rho x : A}{x =_{\sigma(\rho)} x : A} \quad \rhd_{sr} \quad x =_\rho x : A$$

$$\dfrac{\dfrac{x =_r y : A}{y =_{\sigma(r)} x : A}}{x =_{\sigma(\sigma(r))} y : A} \quad \rhd_{sr} \quad x =_r y : A$$

Associated rewritings:

$\sigma(\rho) \rhd_{sr} \rho$

$\sigma(\sigma(r)) \rhd_{sr} r$

Definition 4.9 (Substitution rules (QUEIROZ; OLIVEIRA; RAMOS, 2016)).

$$\dfrac{u =_r \mathcal{C}[x] : A \quad x =_\rho x : A'}{u =_{\mathtt{sub}_{\mathtt{L}}(r,\rho)} \mathcal{C}[x] : A} \quad \rhd_{slr} \quad u =_r \mathcal{C}[x] : A$$

$$\dfrac{x =_\rho x : A' \quad \mathcal{C}[x] =_r z : A}{\mathcal{C}[x] =_{\mathtt{sub}_{\mathtt{R}}(\rho,r)} z : A} \quad \rhd_{srr} \quad \mathcal{C}[x] =_r z : A$$

$$\dfrac{\dfrac{z =_s \mathcal{C}[y] : A \quad y =_r w : A'}{z =_{\mathtt{sub}_{\mathtt{L}}(s,r)} \mathcal{C}[w] : D} \quad \dfrac{y =_r w : A'}{w =_{\sigma(r)} y : D'}}{z =_{\mathtt{sub}_{\mathtt{L}}(\mathtt{sub}_{\mathtt{L}}(s,r),\sigma(r))} \mathcal{C}[y] : A} \ \rhd_{sls} \ z =_s \mathcal{C}[y] : A$$

$$\dfrac{\dfrac{z =_s \mathcal{C}[y] : A \quad y =_r w : A'}{z =_{\mathtt{sub_L}(s,r)} \mathcal{C}[w] : A} \quad \dfrac{y =_r w : A'}{w =_{\sigma(r)} y : A'}}{z =_{\mathtt{sub_L}(\mathtt{sub_L}(s,r),\sigma(r))} \mathcal{C}[y] : A} \triangleright_{slss} \ z =_s \mathcal{C}[y] : A$$

$$\dfrac{x =_s w : A' \quad \dfrac{\dfrac{x =_s w : A'}{w =_{\sigma(s)} x : A'} \quad \mathcal{C}[x] =_r z : A}{\mathcal{C}[w] =_{\mathtt{sub_R}(\sigma(s),r)} z : A}}{\mathcal{C}[x] =_{\mathtt{sub_R}(s,\mathtt{sub_R}(\sigma(s),r))} z : A} \triangleright_{srs} \ \mathcal{C}[x] =_r z : A$$

$$\dfrac{\dfrac{x =_s w : A'}{w =_{\sigma(s)} x : A'} \quad \dfrac{x =_s w : A' \quad \mathcal{C}[w] =_r z : A}{\mathcal{C}[x] =_{\mathtt{sub_R}(s,r)} z : A}}{\mathcal{C}[w] =_{\mathtt{sub_R}(\sigma(s),\mathtt{sub_R}(s,r))} z : A} \triangleright_{srrr} \ \mathcal{C}[w] =_r z : A$$

Associated rewritings:
$\mathtt{sub_L}(\mathcal{C}[r], \mathcal{C}[\rho]) \triangleright_{slr} \mathcal{C}[r]$
$\mathtt{sub_R}(\mathcal{C}[\rho], \mathcal{C}[r]) \triangleright_{srr} \mathcal{C}[r]$
$\mathtt{sub_L}(\mathtt{sub_L}(s, \mathcal{C}[r]), \mathcal{C}[\sigma(r)]) \triangleright_{sls} s$
$\mathtt{sub_L}(\mathtt{sub_L}(s, \mathcal{C}[\sigma(r)]), \mathcal{C}[r]) \triangleright_{slss} s$
$\mathtt{sub_R}(s, \mathtt{sub_R}(\mathcal{C}[\sigma(s)], r)) \triangleright_{srs} r$
$\mathtt{sub_R}(\mathcal{C}[\sigma(s)], \mathtt{sub_R}(\mathcal{C}[s], r)) \triangleright_{srrr} r$

Definition 4.10 ((QUEIROZ; OLIVEIRA; RAMOS, 2016)).

β_{rewr}-$\times$-*reduction*

$$\dfrac{\dfrac{x =_r y : A \qquad z : B}{\langle x, z\rangle =_{\xi_1(r)} \langle y, z\rangle : A \times B} \times\textit{-intr}}{FST(\langle x, z\rangle) =_{\mu_1(\xi_1(r))} FST(\langle y, z\rangle) : A} \times\textit{-elim}$$

$$\triangleright_{mx2l} \quad x =_r y : A$$

$$\dfrac{\dfrac{x =_r x' : A \qquad y =_s z : B}{\langle x, y\rangle =_{\xi_\wedge(r,s)} \langle x', z\rangle : A \times B} \times\textit{-intr}}{FST(\langle x, y\rangle) =_{\mu_1(\xi_\wedge(r,s))} FST(\langle x', z\rangle) : A} \times\textit{-elim}$$

$$\triangleright_{mx2l} \quad x =_r x' : A$$

$$\dfrac{\dfrac{x =_r y : A \qquad z =_s w : B}{\langle x, z\rangle =_{\xi_\wedge(r,s)} \langle y, w\rangle : A \times B} \times\textit{-intr}}{SND(\langle x, z\rangle) =_{\mu_2(\xi_\wedge(r,s))} SND(\langle y, w\rangle) : B} \times\textit{-elim}$$

$$\triangleright_{mx2r} \quad z =_s w : B$$

$$\dfrac{\dfrac{x : A \qquad z =_s w : B}{\langle x, z\rangle =_{\xi_2(s)} \langle x, w\rangle : A \times B} \times\textit{-intr}}{SND(\langle x, z\rangle) =_{\mu_2(\xi_2(s))} SND(\langle x, w\rangle) : B} \times\textit{-elim}$$

$$\rhd_{mx2r} \quad z =_s w : B$$

Associated rewritings:

$\mu_1(\xi_1(r)) \rhd_{mx2l1} r$

$\mu_1(\xi_\wedge(r, s)) \rhd_{mx2l2} r$

$\mu_2(\xi_\wedge(r, s)) \rhd_{mx2r1} s$

$\mu_2(\xi_2(s)) \rhd_{mx2r2} s$

β_{rewr}-+-*reduction*

$$\frac{\dfrac{a =_r a' : A}{i(a) =_{\xi_1(r)} i(a') : A + B}\, +\textit{-intr} \quad \begin{matrix}[x : A] \\ f(x) =_s k(x) : C\end{matrix} \quad \begin{matrix}[y : B] \\ g(y) =_u h(y) : C\end{matrix}}{D(i(a), \acute{x}f(x), \acute{y}g(y)) =_{\mu(\xi_1(r),s,u)} D(i(a'), \acute{x}k(x), \acute{y}h(y)) : C}\, +\textit{-elim}$$

$$\rhd_{mx3l} \quad \begin{matrix} a =_r a' : A \\ f(a/x) =_s k(a'/x) : C \end{matrix}$$

$$\frac{\dfrac{b =_r b' : B}{j(b) =_{\xi_2(r)} j(b') : A + B}\, +\textit{-intr} \quad \begin{matrix}[x : A] \\ f(x) =_s k(x) : C\end{matrix} \quad \begin{matrix}[y : B] \\ g(y) =_u h(y) : C\end{matrix}}{D(j(b), \acute{x}f(x), \acute{y}g(y)) =_{\mu(\xi_2(r),s,u)} D(j(b'), \acute{x}k(x), \acute{y}h(y)) : C}\, +\textit{-elim}$$

$$\rhd_{mx3r} \quad \begin{matrix} b =_s b' : B \\ g(b/y) =_u h(b'/y) : C \end{matrix}$$

Associated rewritings:

$\mu(\xi_1(r), s, u) \rhd_{mx3l} s$

$\mu(\xi_2(r), s, u) \rhd_{mx3r} u$

β_{rewr}-Π-*reduction*

$$\frac{a : A \quad \dfrac{\begin{matrix}[x : A] \\ f(x) =_r g(x) : B(x)\end{matrix}}{\lambda x.f(x) =_{\xi(r)} \lambda x.g(x) : \Pi x : A.B(x)}}{APP(\lambda x.f(x), a) =_{\nu(\xi(r))} APP(\lambda x.g(x), a) : B(a)}$$

$$\rhd_{mxl} \quad \begin{matrix} a : A \\ f(a/x) =_r g(a/x) : B(a) \end{matrix}$$

Associated rewriting:

$\nu(\xi(r)) \rhd_{mxl} r$

β_{rewr}-Σ-*reduction*

$$\frac{\dfrac{a =_r a' : A \quad f(a) : B(a)}{\varepsilon x.(f(x), a) =_{\xi_1(r)} \varepsilon x.(f(x), a') : \Sigma x : A.B(x)} \quad \begin{matrix}[t : A, g(t) : B(t)] \\ d(g, t) =_s h(g, t) : C\end{matrix}}{E(\varepsilon x.(f(x), a), \acute{g}\acute{t}d(g, t)) =_{\mu(\xi_1(r),s)} E(\varepsilon x.(f(x), a'), \acute{g}\acute{t}h(g, t)) : C}$$

$$\rhd_{mxr} \quad \begin{matrix} a =_r a' : A \quad f(a) : B(a) \\ d(f/g, a/t) =_s h(f/g, a'/t) : C \end{matrix}$$

$$\frac{\dfrac{a : A \quad f(a) =_r i(a) : B(a)}{\varepsilon x.(f(x), a) =_{\xi_2(r)} \varepsilon x.(i(x), a) : \Sigma x : A.B(x)} \quad \begin{matrix}[t : A, g(t) : B(t)] \\ d(g, t) =_s h(g, t) : C\end{matrix}}{E(\varepsilon x.(f(x), a), \acute{g}\acute{t}d(g, t)) =_{\mu(\xi_2(r),s)} E(\varepsilon x.(i(x), a), \acute{g}\acute{t}h(g, t)) : C}$$

$$\triangleright_{mxl} \quad \begin{array}{c} a : A \quad f(a) =_r i(a) : B(a) \\ d(f/g, a/t) =_s h(i/g, a/t) : C \end{array}$$

Associated rewritings:
$\mu(\xi_1(r), s) \triangleright_{mxr} s$
$\mu(\xi_2(r), s) \triangleright_{mxl} s$

Definition 4.11 (η_{rewr} (QUEIROZ; OLIVEIRA; RAMOS, 2016)).

η_{rewr}- $\times$-*reduction*

$$\dfrac{\dfrac{x =_r y : A \times B}{FST(x) =_{\mu_1(r)} FST(y) : A}\times\textit{-elim} \quad \dfrac{x =_r y : A \times B}{SND(x) =_{\mu_2(r)} SND(y) : B}\times\textit{-elim}}{\langle FST(x), SND(x)\rangle =_{\xi(\mu_1(r),\mu_2(r))} \langle FST(y), SND(y)\rangle : A \times B}\times\textit{-intr}$$
$$\triangleright_{mx} \; x =_r y : A \times B$$

η_{rewr}- $+$-*reduction*

$$\dfrac{c =_t d : A + B \quad \dfrac{[a_1 =_r a_2 : A]}{i(a_1) =_{\xi_1(r)} i(a_2) : A + B}+\textit{-intr} \quad \dfrac{[b_1 =_s b_2 : B]}{j(b_1) =_{\xi_2(s)} j(b_2) : A + B}+\textit{-intr}}{D(c, \acute{a}_1 i(a_1), \acute{b}_1 j(b_1)) =_{\mu(t,\xi_1(r),\xi_2(s))} D(d, \acute{a}_2 i(a_2), \acute{b}_2 j(b_2))}+\textit{-elim}$$
$$\triangleright_{mxx} \quad c =_t d : A + B$$

Π-η_{rewr}-*reduction*

$$\dfrac{\dfrac{[t : A] \quad c =_r d : \Pi x : A.B(x)}{APP(c, t) =_{\nu(r)} APP(d, t) : B(t)}\Pi\textit{-elim}}{\lambda t.APP(c, t) =_{\xi(\nu(r))} \lambda t.APP(d, t) : \Pi t : A.B(t)}\Pi\textit{-intr}$$
$$\triangleright_{xmr} \qquad c =_r d : \Pi x : A.B(x)$$

where c and d do not depend on x.

Σ-η_{rewr}-*reduction*

$$\dfrac{c =_s b : \Sigma x : A.B(x) \quad \dfrac{[t : A] \quad [g(t) =_r h(t) : B(t)]}{\varepsilon y.(g(y), t) =_{\xi_2(r)} \varepsilon y.(h(y), t) : \Sigma y : A.B(y)}\Sigma\textit{-intr}}{E(c, \acute{g}\acute{t}\varepsilon y.(g(y), t)) =_{\mu(s,\xi_2(r))} E(b, \acute{h}\acute{t}\varepsilon y.(h(y), t)) : \Sigma y : A.B(y)}\Sigma\textit{-elim}$$
$$\triangleright_{mxlr} \quad c =_s b : \Sigma x : A.B(x)$$

Associated rewritings:
$\xi(\mu_1(r), \mu_2(r)) \triangleright_{mx} r$
$\mu(t, \xi_1(r), \xi_2(s)) \triangleright_{mxx} t$
$\xi(\nu(r)) \triangleright_{xmr} r$
$\mu(s, \xi_2(r)) \triangleright_{mxlr} s$

Definition 4.12 (σ and τ (QUEIROZ; OLIVEIRA; RAMOS, 2016)).

$$\dfrac{\dfrac{x =_r y : A \quad y =_s w : A}{x =_{\tau(r,s)} w : A}}{w =_{\sigma(\tau(r,s))} x : A} \quad \rhd_{stss} \quad \dfrac{\dfrac{y =_s w : A}{w =_{\sigma(s)} y : A} \quad \dfrac{x =_r y : A}{y =_{\sigma(r)} x : A}}{w =_{\tau(\sigma(s),\sigma(r))} x : A}$$

Associated rewriting:
$\sigma(\tau(r,s)) \rhd_{stss} \tau(\sigma(s),\sigma(r))$

Definition 4.13 (σ and `sub` (QUEIROZ; OLIVEIRA; RAMOS, 2016)).

$$\dfrac{\dfrac{x =_r \mathcal{C}[y] : A \quad y =_s w : A'}{x =_{\mathtt{sub}_{\mathtt{L}}(r,s)} \mathcal{C}[w] : A}}{\mathcal{C}[w] =_{\sigma(\mathtt{sub}_{\mathtt{L}}(r,s))} x : A} \quad \rhd_{ssbl} \quad \dfrac{\dfrac{y =_s w : A'}{w =_{\sigma(s)} y : A'} \quad \dfrac{x =_r \mathcal{C}[y] : A}{\mathcal{C}[y] =_{\sigma(r)} x : A}}{\mathcal{C}[w] =_{\mathtt{sub}_{\mathtt{R}}(\sigma(s),\sigma(r))} x : A}$$

$$\dfrac{\dfrac{x =_r y : A' \quad \mathcal{C}[y] =_s w : A}{\mathcal{C}[x] =_{\mathtt{sub}_{\mathtt{R}}(r,s)} w : A}}{w =_{\sigma(\mathtt{sub}_{\mathtt{R}}(r,s))} \mathcal{C}[x] : D} \quad \rhd_{ssbr} \quad \dfrac{\dfrac{\mathcal{C}[y] =_s w : A}{w =_{\sigma(s)} \mathcal{C}[y] : A} \quad \dfrac{x =_r y : A'}{y =_{\sigma(r)} x : A'}}{w =_{\mathtt{sub}_{\mathtt{L}}(\sigma(s),\sigma(r))} \mathcal{C}[x] : A}$$

Associated rewritings:
$\sigma(\mathtt{sub}_{\mathtt{L}}(r,s)) \rhd_{ssbl} \mathtt{sub}_{\mathtt{R}}(\sigma(s),\sigma(r))$
$\sigma(\mathtt{sub}_{\mathtt{R}}(r,s)) \rhd_{ssbr} \mathtt{sub}_{\mathtt{L}}(\sigma(s),\sigma(r))$

Definition 4.14 (σ and ξ (QUEIROZ; OLIVEIRA; RAMOS, 2016)).

$$\dfrac{\dfrac{x =_r y : A}{i(x) =_{\xi_1(r)} i(y) : A + B}}{i(y) =_{\sigma(\xi_1(r))} i(x) : A + B} \quad \rhd_{sx} \quad \dfrac{\dfrac{x =_r y : A}{y =_{\sigma(r)} x : A}}{i(y) =_{\xi_1(\sigma(r))} i(x) : A + B}$$

$$\dfrac{\dfrac{x =_r y : A \quad z =_s w : B}{\langle x, z\rangle =_{\xi(r,s)} \langle y, w\rangle : A \times B}}{\langle y, w\rangle =_{\sigma(\xi(r,s))} \langle x, z\rangle : A \times B} \quad \rhd_{sxss} \quad \dfrac{\dfrac{x =_r y : A}{y =_{\sigma(r)} x : A} \quad \dfrac{z =_s w : B}{w =_{\sigma(s)} z : B}}{\langle y, w\rangle =_{\xi(\sigma(r),\sigma(s))} \langle x, z\rangle : A \times B}$$

$$\dfrac{\dfrac{\begin{array}{c}[x : A] \\ f(x) =_s g(x) : B(x)\end{array}}{\lambda x.f(x) =_{\xi(s)} \lambda x.g(x) : \Pi x : A.B(x)}}{\lambda x.g(x) =_{\sigma(\xi(s))} \lambda x.f(x) : \Pi x : A.B(x)} \quad \rhd_{smss} \quad \dfrac{\begin{array}{c}[x : A] \\ \dfrac{f(x) =_s g(x) : B(x)}{g(x) =_{\sigma(s)} f(x) : B(x)}\end{array}}{\lambda x.g(x) =_{\xi(\sigma(s))} \lambda x.f(x) : \Pi x : A.B(x)}$$

Associated rewritings:
$\sigma(\xi(r)) \triangleright_{sx} \xi(\sigma(r))$
$\sigma(\xi(r,s)) \triangleright_{sxss} \xi(\sigma(r),\sigma(s))$
$\sigma(\xi(s) \triangleright_{smss} \xi(\sigma(s))$

Definition 4.15 (σ and μ (QUEIROZ; OLIVEIRA; RAMOS, 2016)).

$$\cfrac{\cfrac{x =_r y : A \times B}{FST(x) =_{\mu_1(r)} FST(y) : A}}{FST(y) =_{\sigma(\mu_1(r))} FST(x) : A} \quad \triangleright_{sm} \quad \cfrac{\cfrac{x =_r y : A \times B}{y =_{\sigma(r)} x : A \times B}}{FST(y) =_{\mu_1(\sigma(r))} FST(x) : A}$$

$$\cfrac{\cfrac{x =_r y : A \times B}{SND(x) =_{\mu_2(r)} SND(y) : A}}{SND(y) =_{\sigma(\mu_2(r))} SND(x) : A} \quad \triangleright_{sm} \quad \cfrac{\cfrac{x =_r y : A \times B}{y =_{\sigma(r)} x : A \times B}}{SND(y) =_{\mu_2(\sigma(r))} SND(x) : A}$$

$$\cfrac{\cfrac{x =_s y : A \quad f =_r g : A \to B}{APP(f,x) =_{\mu(s,r)} APP(g,y) : B}}{APP(g,y) =_{\sigma(\mu(s,r))} APP(f,x) : B}$$

$$\triangleright_{smss} \quad \cfrac{\cfrac{x =_s y : A}{y =_{\sigma(s)} x : A} \quad \cfrac{f =_r g : A \to B}{g =_{\sigma(r)} f : A \to B}}{APP(g,y) =_{\mu(\sigma(s),\sigma(r))} APP(f,x) : B}$$

$$\cfrac{\cfrac{x =_r y : A + B \quad \begin{array}{c}[s : A] \\ \vdots \\ d(s) =_u f(s) : C\end{array} \quad \begin{array}{c}[t : B] \\ \vdots \\ e(t) =_v g(t) : C\end{array}}{D(x, \acute{s}d(s), \acute{t}e(t)) =_{\mu(r,u,v)} D(y, \acute{s}f(s), \acute{t}g(t)) : C}}{D(y, \acute{s}f(s), \acute{t}g(t)) : C =_{\sigma(\mu(r,u,v))} D(x, \acute{s}d(s), \acute{t}e(t)) : C}$$

$$\triangleright_{smsss} \cfrac{\cfrac{x =_r y : A + B}{y =_{\sigma(r)} x : A + B} \quad \cfrac{\begin{array}{c}[s : A] \\ d(s) =_u f(s) : C\end{array}}{f(s) =_{\sigma(u)} d(s) : C} \quad \cfrac{\begin{array}{c}[t : B] \\ e(t) =_v g(t) : C\end{array}}{g(t) =_{\sigma(v)} e(t) : C}}{D(y, \acute{s}f(s), \acute{t}g(t)) =_{\mu(\sigma(r),\sigma(u),\sigma(v))} D(x, \acute{s}d(s), \acute{t}e(t)) : C}$$

$$\cfrac{\cfrac{e =_s b : \Sigma x : A.B(x) \quad \begin{array}{c}[t : A,\ g(t) : B(t)] \\ d(g,t) =_r f(g,t) : C\end{array}}{E(e, \acute{g}\acute{t}d(g,t)) =_{\mu(s,r)} E(b, \acute{g}\acute{t}f(g,t)) : C}}{E(b, \acute{g}\acute{t}f(g,t)) =_{\sigma(\mu(s,r))} E(e, \acute{g}\acute{t}d(g,t)) : C}$$

$$\triangleright_{smss} \cfrac{\cfrac{e =_s b : \Sigma x : A.B(x)}{b =_{\sigma(s)} e : \Sigma x : A.B(x)} \quad \cfrac{\begin{array}{c}[t : A,\ g(t) : B(t)] \\ d(g,t) =_r f(g,t) : C\end{array}}{f(g,t) =_{\sigma(r)} d(g,t) : C}}{E(b, \acute{g}\acute{t}f(g,t)) =_{\mu(\sigma(s),\sigma(r))} E(e, \acute{g}\acute{t}d(g,t)) : C}$$

Associated rewritings:
$\sigma(\mu_1(r)) \triangleright_{sm} \mu_1(\sigma(r))$

$\sigma(\mu_2(r)) \rhd_{sm} \mu_2(\sigma(r))$
$\sigma(\mu(s,r)) \rhd_{smss} \mu(\sigma(s), \sigma(r))$
$\sigma(\mu(r,u,v)) \rhd_{smsss} \mu(\sigma(r), \sigma(u), \sigma(v))$

Definition 4.16 (τ and $\mathtt{sub}$ (QUEIROZ; OLIVEIRA; RAMOS, 2016)).

$$\dfrac{\dfrac{x =_r \mathcal{C}[y] : A \quad y =_s w : A'}{x =_{\mathtt{sub}_{\mathtt{L}}(r,s)} \mathcal{C}[w] : A} \qquad \mathcal{C}[w] =_t z : A}{x =_{\tau(\mathtt{sub}_{\mathtt{L}}(r,s),t)} z : A}$$

$$\rhd_{tsbll} \quad \dfrac{x =_r \mathcal{C}[y] : A \qquad \dfrac{y =_s w : A' \quad \mathcal{C}[w] =_t z : A}{\mathcal{C}[y] =_{\mathtt{sub}_{\mathtt{R}}(s,t)} z : A}}{x =_{\tau(r,\mathtt{sub}_{\mathtt{R}}(s,t))} z : A}$$

$$\dfrac{\dfrac{y =_s w : A \quad \mathcal{C}[w] =_t z : A}{\mathcal{C}[y] =_{\mathtt{sub}_{\mathtt{R}}(s,t)} z : A} \qquad z =_u v : A}{\mathcal{C}[y] =_{\tau(\mathtt{sub}_{\mathtt{R}}(s,t),u)} v : A}$$

$$\rhd_{tsbrl} \quad \dfrac{y =_s w : D' \qquad \dfrac{\mathcal{C}[w] =_t z : A \quad z =_u v : A}{\mathcal{C}[w] =_{\tau(t,u)} v : A}}{\mathcal{C}[y] =_{\mathtt{sub}_{\mathtt{R}}(s,\tau(t,u))} v : A}$$

$$\dfrac{x =_r \mathcal{C}[z] : A \qquad \dfrac{\mathcal{C}[z] =_\rho \mathcal{C}[z] : A \quad z =_s w : A'}{\mathcal{C}[z] =_{\mathtt{sub}_{\mathtt{L}}(\rho,s)} \mathcal{C}[w] : A}}{x =_{\tau(r,\mathtt{sub}_{\mathtt{L}}(\rho,s))} \mathcal{C}[w] : A}$$

$$\rhd_{tsblr} \quad \dfrac{x =_r \mathcal{C}[z] : A \quad z =_s w : A'}{x =_{\mathtt{sub}_{\mathtt{L}}(r,s)} \mathcal{C}[w] : A}$$

$$\dfrac{x =_r \mathcal{C}[w] : A \qquad \dfrac{w =_s z : A' \quad \mathcal{C}[z] =_\rho \mathcal{C}[z] : A}{\mathcal{C}[w] =_{\mathtt{sub}_{\mathtt{R}}(s,\rho)} \mathcal{C}[z] : A}}{x =_{\tau(r,\mathtt{sub}_{\mathtt{R}}(s,\rho))} \mathcal{C}[z] : A}$$

$$\rhd_{tsbrr} \quad \dfrac{x =_r \mathcal{C}[w] : D \quad w =_s z : A'}{x =_{\mathtt{sub}_{\mathtt{L}}(r,s)} \mathcal{C}[z] : A}$$

Definition 4.17 (τ and τ (QUEIROZ; OLIVEIRA; RAMOS, 2016)).

$$\dfrac{\dfrac{x =_t y : A \quad y =_r w : A}{x =_{\tau(t,r)} w : A} \qquad w =_s z : A}{x =_{\tau(\tau(t,r),s)} z : A}$$

$$\rhd_{tt} \quad \dfrac{x =_t y : A \qquad \dfrac{y =_r w : A \quad w =_s z : A}{y =_{\tau(r,s)} z : A}}{x =_{\tau(t,\tau(r,s))} z : A}$$

Associated rewritings:
$\tau(\mathtt{sub}_{\mathtt{L}}(r,s),t) \rhd_{tsbll} \tau(r, \mathtt{sub}_{\mathtt{R}}(s,t))$
$\tau(\mathtt{sub}_{\mathtt{R}}(s,t),u)) \rhd_{tsbrl} \mathtt{sub}_{\mathtt{R}}(s, \tau(t,u))$

$\tau(r, \mathtt{sub}_{\mathtt{L}}(\tau, s)) \triangleright_{tsblr} \mathtt{sub}_{\mathtt{L}}(r, s)$
$\tau(r, \mathtt{sub}_{\mathtt{R}}(s, \tau)) \triangleright_{tsbrr} \mathtt{sub}_{\mathtt{L}}(r, s)$
$\tau(\tau(t, r), s) \triangleright_{tt} \tau(t, \tau(r, s))$

Thus, we put together all those rules to compose our rewrite system:

Definition 4.18 ($LND_{EQ} - TRS$ (QUEIROZ; OLIVEIRA; RAMOS, 2016))**.**
1. $\sigma(\rho) \triangleright_{sr} \rho$
2. $\sigma(\sigma(r)) \triangleright_{ss} r$
3. $\tau(\mathcal{C}[r], \mathcal{C}[\sigma(r)]) \triangleright_{tr} \mathcal{C}[\rho]$
4. $\tau(\mathcal{C}[\sigma(r)], \mathcal{C}[r]) \triangleright_{tsr} \mathcal{C}[\rho]$
5. $\tau(\mathcal{C}[r], \mathcal{C}[\rho]) \triangleright_{trr} \mathcal{C}[r]$
6. $\tau(\mathcal{C}[\rho], \mathcal{C}[r]) \triangleright_{tlr} \mathcal{C}[r]$
7. $\mathtt{sub}_{\mathtt{L}}(\mathcal{C}[r], \mathcal{C}[\rho]) \triangleright_{slr} \mathcal{C}[r]$
8. $\mathtt{sub}_{\mathtt{R}}(\mathcal{C}[\rho], \mathcal{C}[r]) \triangleright_{srr} \mathcal{C}[r]$
9. $\mathtt{sub}_{\mathtt{L}}(\mathtt{sub}_{\mathtt{L}}(s, \mathcal{C}[r]), \mathcal{C}[\sigma(r)]) \triangleright_{sls} s$
10. $\mathtt{sub}_{\mathtt{L}}(\mathtt{sub}_{\mathtt{L}}(s, \mathcal{C}[\sigma(r)]), \mathcal{C}[r]) \triangleright_{slss} s$
11. $\mathtt{sub}_{\mathtt{R}}(\mathcal{C}[s], \mathtt{sub}_{\mathtt{R}}(\mathcal{C}[\sigma(s)], r)) \triangleright_{srs} r$
12. $\mathtt{sub}_{\mathtt{R}}(\mathcal{C}[\sigma(s)], \mathtt{sub}_{\mathtt{R}}(\mathcal{C}[s], r)) \triangleright_{srrr} r$
13. $\mu_1(\xi_1(r)) \triangleright_{mx2l1} r$
14. $\mu_1(\xi_\wedge(r, s)) \triangleright_{mx2l2} r$
15. $\mu_2(\xi_\wedge(r, s)) \triangleright_{mx2r1} s$
16. $\mu_2(\xi_2(s)) \triangleright_{mx2r2} s$
17. $\mu(\xi_1(r), s, u) \triangleright_{mx3l} s$
18. $\mu(\xi_2(r), s, u) \triangleright_{mx3r} u$
19. $\nu(\xi(r)) \triangleright_{mxl} r$
20. $\mu(\xi_2(r), s) \triangleright_{mxr} s$
21. $\xi(\mu_1(r), \mu_2(r)) \triangleright_{mx} r$
22. $\mu(t, \xi_1(r), \xi_2(s)) \triangleright_{mxx} t$
23. $\xi(\nu(r)) \triangleright_{xmr} r$
24. $\mu(s, \xi_2(r)) \triangleright_{mx1r} s$
25. $\sigma(\tau(r, s)) \triangleright_{stss} \tau(\sigma(s), \sigma(r))$
26. $\sigma(\mathtt{sub}_{\mathtt{L}}(r, s)) \triangleright_{ssbl} \mathtt{sub}_{\mathtt{R}}(\sigma(s), \sigma(r))$
27. $\sigma(\mathtt{sub}_{\mathtt{R}}(r, s)) \triangleright_{ssbr} \mathtt{sub}_{\mathtt{L}}(\sigma(s), \sigma(r))$
28. $\sigma(\xi(r)) \triangleright_{sx} \xi(\sigma(r))$
29. $\sigma(\xi(s, r)) \triangleright_{sxss} \xi(\sigma(s), \sigma(r))$
30. $\sigma(\mu(r)) \triangleright_{sm} \mu(\sigma(r))$
31. $\sigma(\mu(s, r)) \triangleright_{smss} \mu(\sigma(s), \sigma(r))$
32. $\sigma(\mu(r, u, v)) \triangleright_{smsss} \mu(\sigma(r), \sigma(u), \sigma(v))$
33. $\tau(r, \mathtt{sub}_{\mathtt{L}}(\rho, s)) \triangleright_{tsbll} \mathtt{sub}_{\mathtt{L}}(r, s)$

34. $\tau(r, \mathtt{sub_R}(s, \rho)) \rhd_{tsbrl} \mathtt{sub_L}(r, s)$
35. $\tau(\mathtt{sub_L}(r, s), t) \rhd_{tsblr} \tau(r, \mathtt{sub_R}(s, t))$
36. $\tau(\mathtt{sub_R}(s, t), u) \rhd_{tsbrr} \mathtt{sub_R}(s, \tau(t, u))$
37. $\tau(\tau(t, r), s) \rhd_{tt} \tau(t, \tau(r, s))$
38. $\tau(\mathcal{C}[u], \tau(\mathcal{C}[\sigma(u)], v)) \rhd_{tts} v$
39. $\tau(\mathcal{C}[\sigma(u)], \tau(\mathcal{C}[u], v)) \rhd_{tst} u$.

4.2.2 Normalization

In the previous subsection, we have seen a system of rewrite rules that resolves reductions in a computational path. When we talk about these kind of systems, two questions arise: Every computational path has a normal form? And if a computational path has a normal form, is it unique? To show that it has a normal form, one has to prove that every computational path terminates, i.e., after a finite number of rewrites, one will end up with a path that does not have any additional reduction. To show that it is unique, one needs to show that the system is confluent. In other words, if one has a path with 2 or more reductions, one needs to show that the choice of the rewrite rule does not matter. In the end, one will always obtain the same end-path without any redundancies.

4.2.2.1 Termination

We are interested in the following theorem (QUEIROZ; OLIVEIRA; GABBAY, 2011; QUEIROZ; OLIVEIRA; RAMOS, 2016):

Theorem 4.1 (Termination property for $LND_{EQ}-TRS$)**.** *$LND_{EQ}-TRS$ is terminating.*

The proofs uses a special kind of ordering, known as *recursive parth ordering*, proposed by (DERSHOWITZ, 1982):

Definition 4.19 (Recursive path ordering (DERSHOWITZ, 1982; QUEIROZ; OLIVEIRA; RAMOS, 2016))**.** *Let $>$ be a partial ordering on a set of operators F. The recursive path ordering $>^*$ on the set T(F) of terms over F is defined recursively as follows:*

$$s = f(s_1, \ldots, s_m) >^* g(t_1, \ldots, t_n) = t,$$

if and only if

1. $f = g$ *and* $\{s_1, \ldots, s_m\} \gg^* \{t_1, \ldots, t_n\}$, *or*
2. $f > g$ *and* $\{s\} \gg^* \{t_1, \ldots, t_n\}$, *or*
3. $f \ngeq g$ *and* $\{s_1, \ldots, s_m\} \gg^*$ *or* $= \{t\}$

where $\gg^*$ is the extension of $>^*$ to multisets.

This definition uses the notion of partial ordering in multisets. A given partial ordering $>$ on a set S may be extended to a partial ordering $\gg$ on finite multisets of elements of S, wherein a multiset is reduced by removing one or more elements and replacing them with any finite number of elements, each oh which is smaller than one of the elements removed (DERSHOWITZ, 1982).

Thus, one can proof the termination property by showing that all rules $e \to d$ of the system, one has that $e >^* d$.

We also need to define the precedence ordering on the rewrite operators. We define as follows (QUEIROZ; OLIVEIRA; RAMOS, 2016; QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\begin{aligned}
&\sigma > \tau > \rho,\\
&\sigma > \xi,\\
&\sigma > \xi_{\wedge},\\
&\sigma > \xi_1,\\
&\sigma > \xi_2,\\
&\sigma > \mu,\\
&\sigma > \mu_1,\\
&\sigma > \mu_2,\\
&\sigma > \mathtt{sub_L},\\
&\sigma > \mathtt{sub_R},\\
&\tau > \mathtt{sub_L}
\end{aligned}$$

Thus, one can prove the termination by showing that for every rule of $e \to d$ of $LND_{EQ} - TRS$, $e >^* d$. For almost every rule it is a straightforward and tedious process. We are not going to show those steps in this work. One can check the full proof in (QUEIROZ; OLIVEIRA; GABBAY, 2011).

4.2.2.2 Confluence

Before we go to the proof of confluence, one needs to notice that $LND_{EQ}-TRS$ is a conditional term rewriting system. This means that some rules can only be applied if the terms of the associated equation follow some rules. For example, for the rule $\mu_1(\xi_{\wedge}(r, s)) \rhd_{mx2l2} r$, it is necessary to have an β-redex like $FST\langle x, y\rangle$. With that in mind, one has the following definition (QUEIROZ; OLIVEIRA; GABBAY, 2011):

Definition 4.20 (Conditional term rewriting system)**.** *In conditional term rewriting systems, the rules have conditions attached, which must be true for the rewrite occur. For example, a rewrite rule $e \to d$ with condition C is expressed as:*

$$C|e \to d$$

To prove the confluence, one should analyze all possible critical pairs using the superposition algorithm proposed by (KNUTH; BENDIX, 1970). Thus, there should not be any divergent critical pair. For example, one can take the superposition of rules 1 and 2, obtaining: $\sigma(\sigma(\rho))$. We have two possible rewrites (QUEIROZ; OLIVEIRA; GABBAY, 2011):

- $\sigma(\sigma(\rho)) \rhd_{sr} \sigma(\rho) \rhd_{sr} \rho$
- $\sigma(\sigma(\rho)) \rhd_{ss} \rho$.

As one can see, we ended up with the same term ρ. Thus, no divergence has been generated.

One should compare every pair of rules to find all critical pairs and see if there are divergences. If some divergence happens, the superposition algorithm proposed by (KNUTH; BENDIX, 1970) shows how to add new rules to the system in such a way that it becomes confluent. As a matter of fact, that was the reason why the rules 38 and 39 of $LND_{EQ} - TRS$ have been introduced to the system (QUEIROZ; OLIVEIRA; RAMOS, 2016):

38. $\tau(\mathcal{C}[u], \tau(\mathcal{C}[\sigma(u)], v)) \rhd_{tts} v$
39. $\tau(\mathcal{C}[\sigma(u)], \tau(\mathcal{C}[u], v)) \rhd_{tst} u$.

Those two rules introduced the following reductions to the system (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\frac{x =_s u : D \qquad \dfrac{\dfrac{x =_s u : D}{u =_{\sigma(s)} x : D} \qquad x =_v w : D}{u =_{\tau(\sigma(s),v)} w : D}}{x =_{\tau(s,\tau(\sigma(s),v))} w : D} \qquad \rhd_{tts} \quad x =_v w$$

$$\frac{\dfrac{x =_s w : D}{w =_{\sigma(s)} x : D} \qquad \dfrac{x =_s w : D \qquad w =_v z : D}{x =_{\tau(s,v)} z : D}}{w =_{\tau(\sigma(s),\tau(s,v))} z : D} \qquad \rhd_{ss} \quad w =_v z$$

One can check a full proof of confluence in (OLIVEIRA, 1995; OLIVEIRA; QUEIROZ, 1994; OLIVEIRA; QUEIROZ, 1999; QUEIROZ; OLIVEIRA; GABBAY, 2011).

4.2.2.3 Normalization Procedure

We can now state two normalization theorems:

Theorem 4.2 (normalization (QUEIROZ; OLIVEIRA; GABBAY, 2011))**.** *Every derivation in the $LND_{EQ} - TRS$ converts to a normal form.*

Proof. Direct consequence of the termination property. □

Theorem 4.3 (strong normalization (QUEIROZ; OLIVEIRA; GABBAY, 2011))**.** *Every derivation in the $LND_{EQ} - TRS$ converts to a unique normal form.*

Proof. Direct consequence of the termination and confluence properties. □

In this sense, every proof can be reduced to a normal one. To do that, one should identify the redundancies and, based on the rewrite rules, one can construct a proof without any redundancies. We show that in an example. It is the following (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\dfrac{\dfrac{\dfrac{\dfrac{f(x,z) =_s f(w,y) : D}{f(w,y) =_{\sigma(s)} f(x,z) : D} \qquad x =_r c : D}{f(w,y) =_{sub_L(\sigma(s),r)} f(c,z) : D}}{f(c,z) =_{\sigma(sub_L(\sigma(s),r))} f(w,y) : D} \qquad y =_t b : D}{f(c,z) =_{sub_L(\sigma(sub_L(\sigma(s),r)))} f(w,b) : D}$$

This deduction generates the following path: $sub_L(\sigma(sub_L(\sigma(s),r)))$. This path is not in normal form, having two redundancies (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$sub_L(\sigma(sub_L(\sigma(s),r))) \rhd_{ssbl} sub_L(sub_R(\sigma(r),\sigma(\sigma(s)),t)$$
$$sub_L(sub_R(\sigma(r),\sigma(\sigma(s)),t) \rhd_{ss} sub_L(sub_R(\sigma(r),s),t)$$

Thus, one can identify those reductions and conceive a deduction without any redundancies (QUEIROZ; OLIVEIRA; GABBAY, 2011):

$$\dfrac{\dfrac{\dfrac{x =_r c : D}{c =_{\sigma(r)} x : D} \qquad f(x,z) =_s f(w,y) : D}{f(c,z) =_{sub_R(\sigma(r),s)} f(w,y) : D} \qquad y =_t b : D}{f(c,z) =_{sub_L(sub_R(\sigma(r),s),t)} f(w,b) : D}$$

4.2.3 Rewrite Equality

As we have just seen, the $LND_{EQ} - TRS$ has 39 rewrite rules. We call each rule as a *rewrite rule* (abbreviation: *rw-rule*). We have the following definition:

Definition 4.21 (Rewrite Rule (RAMOS; QUEIROZ; OLIVEIRA, 2017))**.** *An rw-rule is any of the rules defined in $LND_{EQ} - TRS$.*

Similarly to the β-reduction of λ-calculus, we have a definition for rewrite reduction:

Definition 4.22 (Rewrite reduction (RAMOS; QUEIROZ; OLIVEIRA, 2017))**.** *Let s and t be computational paths. We say that $s \rhd_{1rw} t$ (read as: s rw-contracts to t) iff we can obtain t from s by an application of only one rw-rule. If s can be reduced to t by finite number of rw-contractions, then we say that $s \rhd_{rw} t$ (read as s rw-reduces to t).*

We also have rewrite contractions and equality:

Definition 4.23 (Rewrite contraction and equality (RAMOS; QUEIROZ; OLIVEIRA, 2017))**.** *Let s and t be computational paths. We say that $s =_{rw} t$ (read as: s is rw-equal to t) iff t can be obtained from s by a finite (perhaps empty) series of rw-contractions and reversed rw-contractions. In other words, $s =_{rw} t$ iff there exists a sequence $R_0, ..., R_n$, with $n \geq 0$, such that*

$$(\forall i \leq n-1)(R_i \rhd_{1rw} R_{i+1} \text{ or } R_{i+1} \rhd_{1rw} R_i)$$
$$R_0 \equiv s, \quad R_n \equiv t$$

A fundamental result is the fact that rewrite equality is an equivalence relation (RAMOS; QUEIROZ; OLIVEIRA, 2017):

Proposition 4.1. *Rewrite equality is transitive, symmetric and reflexive.*

Proof. Comes directly from the fact that rw-equality is the transitive, reflexive and symmetric closure of rw. □

Rewrite reduction and equality play fundamental roles in the groupoid model of a type based on computational paths, as we are going to see in the sequel.

4.2.4 $LND_{EQ} - TRS_2$

Until now, this subsection has concluded that there exist redundancies which are resolved by a system called $LND_{EQ} - TRS$. This system establishes rules that reduces these redundancies. Moreover, we concluded that these redundancies are just redundant uses of the equality axioms showed in *section 2*. In fact, since these axioms just define an equality theory for type theory, one can specify and say that these are redundancies of the equality of type theory. As we mentioned, the $LND_{EQ} - TRS$ has a total of 39 rules(OLIVEIRA, 1995; QUEIROZ; OLIVEIRA; RAMOS, 2016). Since the rw-equality is based on the rules of $LND_{EQ} - TRS$, one can just imagine the high number of redundancies that rw-equality could cause. In fact, a thoroughly study of all the redundancies caused by these rules could generate an entire new work. Fortunately, we are only interested in the redundancies caused by the fact that rw-equality is transitive, reflexive and symmetric with the addition of only one specific rw_2-rule. Let's say that we have a system, called $LND_{EQ} - TRS_2$, that resolves all the redundancies caused by rw-equality (the same way that $LND_{EQ} - TRS$ resolves all the redundancies caused by equality). Since we know that rw-equality is transitive, symmetric and reflexive, it should have the same redundancies that the equality had involving only these properties. Since rw-equality is just a sequence of rw-rules (also similar to equality, since equality is just a computational path, i.e., a sequence of identifiers), then we could put a name on these sequences. For example, if s and t are rw-equal because there exists a sequence $\theta : R_0, ..., R_n$ that justifies the rw-equality, then we can write that $s =_{rw_\theta} t$. Thus, we can rewrite, using rw-equality, all the rules that originated the rules involving τ, σ and ρ. For example, we have (RAMOS; QUEIROZ; OLIVEIRA, 2017):

$$\frac{\dfrac{x =_{rw_t} y : A \qquad y =_{rw_r} w : A}{x =_{rw_{\tau(t,r)}} w : A} \qquad w =_{rw_s} z : A}{x =_{rw_{\tau(\tau(t,r),s)}} z : A}$$

$$\triangleright_{tt_2} \frac{x =_{rw_t} y : A \qquad \dfrac{y =_{rw_r} w : A \qquad w =_{rw_s} z : A}{y =_{rw_{\tau(r,s)}} z : A}}{x =_{rw_{\tau(t,\tau(r,s))}} z : A}$$

Therefore, we obtain the rule tt_2, that resolves one of the redundancies caused by the transitivity of rw-equality (the 2 in tt_2 indicates that it is a rule that resolves a redundancy of rw-equality). In fact, using the same reasoning, we can obtain, for rw-equality, all the redundancies that we have shown in **definition 4.18**. In other words, we have tr_2, tsr_2, trr_2, tlr_2, sr_2, ss_2 and tt_2. Since we have now rules of $LND_{EQ} - TRS_2$, we can use all the concepts that we have just defined for $LND_{EQ} - TRS$. The only difference is that instead of having rw-rules and rw-equality, we have rw_2-rules and rw_2-equality.

There is an important rule specific to this system. It stems from the fact that transitivity of reducible paths can be reduced in different ways, but generating the same result. For example, consider the simple case of $\tau(s,t)$ and consider that it is possible to reduce s to s' and t to t'. There is two possible rw-sequences that reduces this case: The first one is $\theta : \tau(s,t) \triangleright_{1rw} \tau(s',t) \triangleright_{1rw} \tau(s',t')$ and the second $\theta' : \tau(s,t) \triangleright_{1rw} \tau(s,t') \triangleright_{1rw} \tau(s',t')$. Both rw-sequences obtained the same result in similar ways, the only difference being the choices that have been made at each step. Since the variables, when considered individually, followed the same reductions, these rw-sequences should be considered redundant relative to each other and, for that reason, there should be rw_2-rule that establishes this reduction. This rule is called *independence of choice* and is denoted by cd_2. Since we already understand the necessity of such a rule, we can define it formally:

Definition 4.24 (Independence of choice (RAMOS; QUEIROZ; OLIVEIRA, 2017)). *Let θ and ϕ be rw-equalities expressed by two rw-sequences: $\theta : \theta_1, ..., \theta_n$, with $n \geq 1$, and $\phi : \phi_1, ..., \phi_m$, with $m \geq 1$. Let T be the set of all possible rw-equalities from $\tau(\theta_1, \phi_1)$ to $\tau(\theta_n, \theta_m)$ described by the following process: $t \in T$ is of the form $\tau(\theta_{l_1}, \phi_{r_1}) \triangleright_{1rw} \tau(\theta_{l_2}, \phi_{r_2}) \triangleright_{1rw} ... \triangleright_{1rw} \tau(\theta_{l_x}, \phi_{r_y})$, with $l_1 = 1, r_1 = 1$, $l_x = n, r_y = m$ and $l_{i+1} = 1 + l_i$*

and $r_{i+1} = r_i$ or $l_{i+1} = l_i$ and $r_{i+1} = 1 + r_i$. The independence of choice, denoted by cd_2, is defined as the rule of $LND_{EQ} - TRS_2$ that establishes the equality between any two different terms of T. In other words, if $x, y \in T$ and $x \neq y$, then $x =_{cd_2} y$ and $y =_{cd_2} x$.

Analogously to the rw-equality, rw_2-equality is also an equivalence relation (RAMOS; QUEIROZ; OLIVEIRA, 2017):

Proposition 4.2. *rw_2-equality is transitive, symmetric and reflexive.*

Proof. Analogous to Proposition 4.1. □

4.3 GROUPOID MODEL

In this section, we use categorical concepts introduced in the previous chapter to show that we can use computational paths to induce a groupoid structure of a type. Nevertheless, before we do that, we show that computational paths form a globular structure.

4.3.1 Globular Structure

In the previous section, we have just shown the existence of paths between computational paths. Such paths were originated from redundancies caused by rw-equality. Analogously, one could think of redundancies caused by rw_2-equality, which would be resolved by the establishment of computational paths between two equivalent rw_2-equalities. With that in mind, we can extend this process up to the infinity. That way, we obtain an infinite number of rw_n-equalities and $LND_{EQ} - TRS_n$ systems. We have seen in the previous chapter a mathematical entity suitable to express this fact. It is known as *globular set*. With that in mind, we have the following definition:

Definition 4.25 (Level of a path (RAMOS; QUEIROZ; OLIVEIRA, 2017))**.** *The level of a path is defined as follows:*

- *If $a, b : A$ are terms such that a and b are not rw-equalities, then we say that a path $a =_s b$ has level 1.*
- *If $r, s : A$ are rw_n-equalities, then a path $r =_\theta s$ has level $n + 1$.*

The idea is that if we have two n-level paths r and s and if r can be rewritten into s by a sequence of rewrites established by $LND_{EQ} - TRS_n$, then this sequence of rewrites is a $(n + 1)$-level path that establishes $r =_{rw_n} s$.

Computational paths can form a globular-set structure. To see this, consider $X(n)$ as the set of n-level paths between two $(n - 1)$-level paths (for $n = 0$, just consider two non-path objects) and for any path $x =_m y$, consider that $s(m) = x$ and $t(m) = y$. That way, if $n \geq 2$ and $m \in X(n)$, then $s(m) \in X(n - 1)$ and $t(m) \in X(n - 1)$. Now, if we consider two n-level paths $t =_\theta m$, and $t =_\alpha m$ and a $(n+1)$-level path $\theta =_\phi \alpha$, the following equations hold (RAMOS; QUEIROZ; OLIVEIRA, 2017):

$$s(s(\phi)) = s(\theta) = t = s(\alpha) = s(t(\phi)).$$
$$t(s(\phi)) = t(\theta) = m = t(\alpha) = t(t(\phi)).$$

All globular-set conditions hold. Then, we can think that computational paths form a globular set structure all up to infinity, i.e., a ∞-globular-set.

4.3.2 The Induced Groupoid

The relation between the algebraic structure of groupoid and the identity type was first noticed by(HOFMANN; STREICHER, 1994). In this work, the authors claim that it is possible to think of any type of type theory as having a groupoid structure. To do that, one could think of the terms of a type as objects and the propositional identities between these terms to be morphisms. Using J,(HOFMANN; STREICHER, 1994) showed that the following types, which correspond to the groupoid equations, are inhabited:

- $Id_{Id_A(x,z)}(trans(trans(p,q),r), trans(p, trans(q,r)))$
- $Id_{Id_A(x,y)}(trans(refl(x),r),r)$
- $Id_{Id_A(x,y)}(trans(r, refl(x)),r)$
- $Id_{Id_A(y,y)}(trans(symm(r),r), refl(x))$
- $Id_{Id_A(x,x)}(trans(r, symm(r)), refl(x))$
- $Id_{Id_A(x,y)}(symm(symm(r)),r)$

After this important work, the authors further develop their ideas in (HOFMANN; STREICHER, 1998), showing how J can be interpreted categorically.

Since our approach is not based on J, but on the fact that identities are established by computational paths, our objective in this section is to show that computational paths, together with the reduction rules discussed in the last section, are also capable of inducing groupoidal structures.

Before we conclude that computational paths induces categories with groupoid properties, we need to make clear the difference between a strict and a weak category. As one will see, the word weak will appear many times. This will be the case because some of the categorical equalities will not hold "on the nose", so to say. They will hold up to rw-equality or up to higher levels of rw-equalities. This is similar to the groupoid model of the identity type proposed by(HOFMANN; STREICHER, 1994). In (HOFMANN; STREICHER, 1994), the equalities do not hold "on the nose ", they hold up to propositional equality (or up to homotopy if one uses the homotopy interpretation). To indicate that these equalities hold only up to some property, we say that the induced structure is a weak categorical structure. This will be even more clear in the proof of the following proposition (RAMOS; QUEIROZ; OLIVEIRA, 2017):

Proposition 4.3. *For each type A, computational paths induces a weak categorical structure called A_{rw}.*

Proof. Since we are working with categories, a good understanding of the basic concept of what is a category is essential. If one is not familiarized with the concept, one should check the **chapter 3** of this work. To sum up, a category is a structure with objects and morphisms between these objects. These morphisms must obey two main laws: the associative and identity laws. To define a weak categorical structure A_{rw}, the first step is to define the objects, the morphisms:

- Objects: The objects of A_{rw} are terms a of the type A, i.e., $a : A$.
- Morphisms: A morphism (arrow) between terms $a : A$ and $b : A$ are arrows $s : a \to b$ such that s is a computational path between the terms, i.e., $a =_s b : A$.

We need now to define the composition of morphisms, the identity morphism and check the associative and identity laws. Since we are working in a weak context, the associative and identity law needs only to hold up to rw-equality. Before we check all these conditions, we can represent this structure in the following diagram:

$$\begin{tikzcd}
a : A \arrow[loop left, "\rho_a"] \arrow[r, "s"] \arrow[dr, "\tau(s,r)"'] & b : A \arrow[loop above, "\rho_b"] \arrow[d, "r"] \arrow[dr, "\tau(r,t)"] & \\
& c : A \arrow[loop below, "\rho_c"] \arrow[r, "t"] & d : A \arrow[loop right, "\rho_d"]
\end{tikzcd}$$

- Compositions: Given arrows (paths) $s : a \to b$ and $r : b \to c$, we need to find an arrow $t : a \to c$ such that $t = r \circ s$. To do that, we first need to define the meaning of a composition of paths. Since equality has the transitive axiom, it is natural to define the composition as an application of the transitivity, i.e. $t = r \circ s = \tau(s, r)$. Therefore, for any $s : a \to b$ and $r : b \to c$, we always have a $t = r \circ s = \tau(s, r)$.
- Associativity of the composition: Given arrows $s : a \to b$, $r : b \to c$ and $t : c \to d$, since we are working with a weak categorical structure, we need to conclude that $t \circ (r \circ s) =_{rw} (t \circ r) \circ s$. Substituting the compositions by the corresponding transitivities, we need to conclude that $\tau(\tau(s, r), t) =_{rw} \tau(s, \tau(r, t))$. This is a direct

consequence of the rule tt: $\tau(\tau(s,r),t) \rhd_{tt} \tau(s,\tau(r,t))$. Therefore, we conclude that $\tau(\tau(s,r),t) =_{rw} \tau(s,\tau(r,t))$.

- Identity morphism: For any object a, consider the reflexive path $a =_{\rho} a$ as the identity arrow 1_a. Let's call this path as ρ_a (to indicate that it is a reflexive path of a).

- Identity law: For any arrows (paths) $s : a \to b$, we need to show that $s \circ 1_a = s = 1_b \circ s$. This can be shown (also in a weak sense) with a straightforward application of rw-rules. We have that $s \circ 1_a = s \circ \rho_a = \tau(\rho_a, s) \rhd_{tlr} s$, therefore $\tau(\rho_a, s) =_{rw} s$. For the other equation, we have that $1_b \circ s = \rho_b \circ s = \tau(s, \rho_b) \rhd_{trr} s$ and thus, $\tau(s, \rho_b) =_{rw} s$.

Since all these conditions have been satisfied, we conclude that the structure A_{rw} induced by computational paths is a weak categorical structure. □

Is it possible to induce a strict (i.e., one that the equalities will hold "on the nose ") using computational paths? The answer is *yes*. Since rw-equality is transitive, symmetric and reflexive, rw-equality is an equivalence relation. If we use equivalent classes of rw-equalities as arrows, the equalities will hold "on the nose ". Since we are trying to prove similar results to the ones obtained for identity types by(HOFMANN; STREICHER, 1994) (in (HOFMANN; STREICHER, 1994), as mentioned before, the equalities only in a weak sense), we are more interested in this weak categorical structure that we have just obtained.

We can now show that A_{rw} has a groupoid structure (RAMOS; QUEIROZ; OLIVEIRA, 2017):

Proposition 4.4. *The induced structure A_{rw} has a weak groupoidal structure.*

Proof. We need to show that every arrow is an isomorphism. Since we are working in a weak sense, the isomorphism equalities need only to hold up to rw-equality. To show that, for every arrow $s : a \to b$, we need to show a $t : b \to a$ such that $t \circ s =_{rw} 1_a$ and $s \circ t =_{rw} 1_b$. To do that, recall that every computational path has an inverse path $\sigma(s)$. Put $t = \sigma(s)$. Thus, we have that $s \circ t = s \circ \sigma(s) = \tau(\sigma(s), s) \rhd_{tsr} \rho_b$. Since $\rho_b = 1_b$, we conclude that $s \circ t =_{rw} 1_b$. We also have that $t \circ s = \sigma(s) \circ s = \tau(s, \sigma(s)) \rhd_{tr} \rho_a$. Therefore, $t \circ s = 1_a$. We conclude that every arrow is a weak isomorphism and thus, A_{rw} is a weak groupoidal structure. □

If we work with classes of equivalences as arrows, we can obtain a groupoidal structure in the strict sense. With that, we finally conclude that computational paths have a groupoid model. As one could notice, we only needed to use the properties of computational paths, together with rules of the rewrites $LND_{EQ} - TRS$. Having these concepts and rules, we have just needed to show that computational paths induces simple structures of category theory. The groupoidal model came naturally from the concept of computational paths, without the need of constructing any complex term of an identity type, as done in (HOFMANN; STREICHER, 1994). The fact that computational paths have a natural groupoidal model is an advantage, since it will automatically imply that the identity type has a natural groupoidal model (without needing to build any term using a constructor, like the constructor J of the traditional identity type).

4.3.3 Higher Structures

We have just shown that computational paths induce a weak groupoidal structure known as A_{rw}. We also know that the arrows (or morphisms) of A_{rw} are computational paths between two terms of a type A. As we saw in the previous subsection, sometimes a computational path can be reduced to another by rules that we called rw-rules. That way, if we have terms $a, b : A$ and paths between these terms, we can define a new structure. This new structure, called $A_{2rw}(a, b)$, has objects as computational paths between a and b and there is a morphism between paths $a =_s b$ and $a =_t b$ iff $s =_{rw} t$. Since rw-equality is transitive, reflexive and symmetric, A_{2rw} is a weak categorical structure which the equalities hold up to rw_2-equality. The proof of this fact is analogous to the *proposition 4.1*. The sole difference is the fact that since the morphisms are rw-equalities, instead of computational paths, all the equalities will hold up to rw_2-equality. To see this, take the example of the associativity. Looking at the $LND_{EQ} - TRS_2$ system, we have that $\tau(\tau(\theta, \sigma), \phi) \rhd_{tt_2} \tau(\theta, \tau(\sigma, \phi))$ (θ, σ and ϕ represent rw-equalities between paths from a to b). Therefore, $\tau(\tau(\theta, \sigma), \phi) =_{rw_2} \tau(\theta, \tau(\sigma, \phi))$. The associative law holds up to rw_2-equality. As one can easily check, the identity law will also hold up to rw_2-equality. Therefore, $A_{2rw}(a, b)$ has a weak categorical structure. Analogous to *proposition 4.4*, the groupoid law will also hold up to rw_2-equality.

Instead of considering A_{rw} and $A_{2rw}(a, b)$ as separated structures, we can think of a unique structure with 2-levels. The first level is A_{rw} and the second one is the structure $A_{2rw}(a, b)$ between each pair of objects $a, b \in A_{rw}$. We call this structure $2 - A_{rw}$. The morphisms of the first level are called 1-morphisms and the ones of the second level are called 2-morphisms (also known as 2-arrows or 2-cells). Since it has multiple levels, it is considered a higher structure. We want to prove that this structure is a categorical structure known as weak 2-category. The main problem is the fact that in a weak 2-category, the last level (i.e., the second level) needs to hold up in a strict sense. This is not the case for $2 - A_{2rw}$, since each $A_{2rw}(a, b)$ only holds up to rw_2-equality. Nevertheless, there still a way to induce this weak 2-category. Since rw-equality is an equivalence relation (because it is transitive, symmetric and reflexive), we can consider a special $A_{2rw}(a, b)$, where the arrows are the arrows of $A_{2rw}(a, b)$ modulo rw_2-equality. That way, since the equalities hold up to rw_2-equality in $A_{2rw}(a, b)$, they will hold in a strict sense when we consider the equivalence classes of rw_2-equality. We call this structure $[A_{2rw}(a, b)]$. In this structure, consider the composition of arrows defined as: $[\theta]_{rw_2} \circ [\phi]_{rw_2} = [\theta \circ \phi]_{rw_2}$. Now, we can think of the structure $[2 - A_{rw}]$. This structure is similar to $2 - A_{rw}$. The difference is that the categories of the second level are $[A_{2rw}(a, b)]$ instead of $A_{2rw}(a, b)$. We can now prove that $[2 - A_{rw}]$ is a weak 2-category (RAMOS; QUEIROZ; OLIVEIRA, 2017):

Proposition 4.5. *Computational paths induce a weak* 2*-category called* $[2 - A_{rw}]$*.*

Proof. First of all, let's draw a diagram that represents $[2 - A_{rw}]$:

$$\begin{tikzcd}
a \arrow[r, bend left=60, "s", ""{name=S, below}] \arrow[r, "t"{description}, ""{name=T}] \arrow[r, bend right=60, "x"', ""{name=X, above}] \arrow[Rightarrow, from=S, to=T, "{[\alpha]_{rw_2}}"] \arrow[Rightarrow, from=T, to=X, "{[\chi]_{rw_2}}"] & b \arrow[r, bend left=60, "r", ""{name=R, below}] \arrow[r, "w"{description}, ""{name=W}] \arrow[r, bend right=60, "y"', ""{name=Y, above}] \arrow[Rightarrow, from=R, to=W, "{[\theta]_{rw_2}}"] \arrow[Rightarrow, from=W, to=Y, "{[\varphi]_{rw_2}}"] & c \arrow[r, bend left=60, "p", ""{name=P, below}] \arrow[r, "q"{description}, ""{name=Q}] \arrow[r, bend right=60, "z"', ""{name=Z, above}] \arrow[Rightarrow, from=P, to=Q, "{[\psi]_{rw_2}}"] \arrow[Rightarrow, from=Q, to=Z, "{[\phi]_{rw_2}}"] & d
\end{tikzcd}$$

In this diagram we represent 1-arrows and 2-arrows between these 1-arrows. The fact that 2-arrows are equivalence classes is represented by the brackets.

Since we are working with a higher structures, some new properties should be checked. One of these properties is the fact that 2-arrows can be composed horizontally(LEINSTER, 1998). In other words, given 1-morphisms $s : a \to b$, $r : b \to c$, $t : a \to b$, $w : b \to c$ and 2-morphisms $[\alpha]_{rw_2} : s \to t$ and $[\theta]_{rw_2} : r \to w$, one should be able to define a horizontal composition $\circ_h$: $([\theta]_{rw_2} \circ_h [\alpha]_{rw_2}) : r \circ s \to w \circ t$.

Given $[\alpha]_{rw_2} : [s = \alpha_1, ..., \alpha_n = t]$ and $[\theta]_{rw_2} : [r = \theta_1, ..., \theta_m = w]$, then we define the horizontal composition $([\theta]_{rw_2} \circ_h [\alpha]_{rw_2})$ as the sequence $[\tau(s = \alpha_1, r = \theta_1), ..., \tau(\alpha_n, \theta_1) ..., \tau(\alpha_n = t, \theta_m = w)]_{rw_2}$.

We also need to verify the associative and identity law for $\circ_h$. Since we are working with a weak 2-category, these laws should hold up to natural isomorphism (LEINSTER, 1998). To verify these laws, the idea is that every 2-morphism of $[A_{rw_2}]$ is an isomorphism. To see that, remember that rw-equality is transitive, symmetric and reflexive. Therefore, if we have $[\theta]_{rw_2}$, we can think of the inverse $[\sigma(\theta)]_{rw_2}$. If we compose them, we have that $[\theta]_{rw_2} \circ [\sigma(\theta)]_{rw_2} = [\theta \circ \sigma(\theta)]_{rw_2}$. Since we have in $LND_{EQ} - TRS_2$ that $(\theta \circ \sigma(\theta)) = \tau(\sigma(\theta), \theta)) \rhd_{tsr_2} \rho_r$, then $\theta \circ \sigma(\theta) =_{rw_2} \rho_r$ and thus, $[\theta \circ \sigma(\theta)]_{rw_2} = [\rho_r]_{rw_2}$. Analogously, one can prove that $[\sigma(\theta) \circ \theta]_{rw_2} = [\rho_w]_{rw_2}$. Therefore, every 2-morphism of $[A_{rw_2}]$ is an isomorphism. Since a natural transformation is a natural isomorphism iff every component is an isomorphism (as one can check in the **chapter 3**), we conclude that finding isomorphisms for the associative and identity laws is just a matter of finding the correct morphisms.

For the associative law, we need to check that there is a natural isomorphism between $(([\psi]_{rw_2} \circ_h [\theta]_{rw_2}) \circ_h [\alpha]_{rw_2})$ and $([\psi]_{rw_2} \circ_h ([\theta]_{rw_2} \circ_h [\alpha]_{rw_2}))$. To do this, by the definition of horizontal composition, a component of $(([\psi]_{rw_2} \circ_h [\theta]_{rw_2})$ is a term of the form $\tau(\alpha_x, \tau(\theta_y, \psi_z))$, with x, y, and z being suitable natural numbers that respect the order of the horizontal composition. Analogously, the same component of $([\psi]_{rw_2} \circ_h ([\theta]_{rw_2} \circ_h [\alpha]_{rw_2}))$

is just a suitable term $\tau(\tau(\alpha_x, \theta_y), \psi_z)$. The isomorphism between these component is clearly established by the inverse *tt* rule, i.e., $\tau(\alpha_x, \tau(\theta_y, \psi_z)) =_{rw_{\sigma(tt)}} \tau(\tau(\alpha_x, \theta_y), \psi_z)$

The identity laws use the same idea. We need to check that $([\alpha]_{rw_2} \circ_h [\rho_{\rho_a}]_{rw_2}) = [\alpha]_{rw_2}$. To do that, we need to take components $(\rho_{\rho_a}, \alpha_y)$ and α_y and establish their isomorphism: $(\rho_{\rho_a}, \alpha_y) =_{rw_{tlr}} \alpha_y$.

The other natural isomorphism, i.e., the isomorphism between $([\rho_{\rho_b}]_{rw_2} \circ_h [\alpha]_{rw_2})$ and $[\alpha]_{rw_2}$ can be established in an analogous way, just using the rule *trr* instead of *tlr*. Just for purpose of clarification, ρ_{ρ_a} comes from the reflexive property of *rw*-equality. Since ρ_a is the identity path, using the reflexivity we establish that $\rho_a =_{rw} \rho_a$, generating ρ_{ρ_a}.

We call the associative morphism generated by $\sigma(tt)$ as *assoc* (sometimes also called simply a), the morphism generate by *tlr* as l^*_s and the one generated by *trr* as r^*_s. With the associative and identity isomorphisms established, we now need to check the *interchange law*(LEINSTER, 1998). We need to check that:

$$([\varphi]_{rw_2} \circ [\theta]_{rw_2}) \circ_h ([\chi]_{rw_2} \circ [\alpha]_{rw_2}) = ([\varphi]_{rw_2} \circ_h [\chi]_{rw_2}) \circ ([\theta]_{rw_2} \circ_h [\alpha]_{rw_2})$$

From $(([\varphi]_{rw_2} \circ [\theta]_{rw_2}) \circ_h ([\chi]_{rw_2} \circ [\alpha]_{rw_2}))$, we have:

$$\begin{gathered}(([\varphi]_{rw_2} \circ [\theta]_{rw_2}) \circ_h ([\chi]_{rw_2} \circ [\alpha]_{rw_2})) = \\ [\tau(\theta, \varphi)]_{rw_2} \circ_h [\tau(\alpha, \chi)]_{rw_2} = \\ [\theta_1, ..., \theta_n = \varphi_1, ..., \varphi_{n'}]_{rw_2} \circ_h [(\alpha_1, ..., \alpha_m = \chi_1, ...\chi_{m'}]_{rw_2} = \\ [\tau(\alpha_1, \theta_1), ..., \tau(\alpha_m = \chi_1, \theta_1), ..., \tau(\chi_n, \theta_1), ..., \tau(\chi_n, \theta_{m'} = \varphi_1), ..., \tau(\chi_n, \varphi_{n'})]_{rw_2}\end{gathered}$$

From $(([\varphi]_{rw_2} \circ_h [\chi]_{rw_2}) \circ ([\theta]_{rw_2} \circ_h [\alpha]_{rw_2})))$:

$$\begin{gathered}(([\varphi]_{rw_2} \circ_h [\chi]_{rw_2}) \circ ([\theta]_{rw_2} \circ_h [\alpha]_{rw_2}))(r \circ s) = \\ ([\tau(\chi_1, \varphi_1), ..., \tau(\chi_n, \varphi_1), ..., \tau(\chi_n, \varphi_{n'})]_{rw_2} \circ [\tau(\alpha_1, \theta_1), ..., \tau(\alpha_m, \theta_1), ..., \tau(\alpha_m, \theta_{m'})]_{rw_2} = \\ [\tau(\alpha_1, \theta_1), ..., \tau(\alpha_m, \theta_1), ..., \tau(\alpha_m, \theta_{m'}), ..., \tau(\chi_1, \varphi_1), ..., \tau(\chi_n, \varphi_1), ..., \tau(\chi_n, \varphi_{n'})]_{rw_2}\end{gathered}$$

If one looks closely, one can notice that this is a suitable to apply cd_2. Individually, every variable that appears in the sequence of transitivities follows the same expansion in both cases. The only difference is how the choices have been made. That way, if we construct T, as defined in Definition 4.6, one can check that $\tau(\alpha_1, \theta_1), ..., \tau(\alpha_m = \chi_1, \theta_1), ..., \tau(\chi_n, \theta_1), ..., \tau(\chi_n, \theta_{m'} = \varphi_1), ..., \tau(\chi_n, \varphi_{n'}) \in T$ and $\tau(\alpha_1, \theta_1), ..., \tau(\alpha_m, \theta_1), ..., \tau(\alpha_m, \theta_{m'}), ...$
T For that reason, we establish:

$$\begin{gathered}[\tau(\alpha_1, \theta_1), ..., \tau(\alpha_m = \chi_1, \theta_1), ..., \tau(\chi_n, \theta_1), ..., \tau(\chi_n, \theta_{m'} = \varphi_1), ..., \tau(\chi_n, \varphi_{n'})]_{rw_2} =_{cd_2} \\ [\tau(\alpha_1, \theta_1), ..., \tau(\alpha_m, \theta_1), ..., \tau(\alpha_m, \theta_{m'}), ..., \tau(\chi_1, \varphi_1), ..., \tau(\chi_n, \varphi_1), ..., \tau(\chi_n, \varphi_{n'})]_{rw_2}.\end{gathered}$$

Since we are working with equivalence classes, this equality holds strictly.

To end the proof, there is one more thing that we should verify. Since we are working with a weak structure, there is some laws that should hold. These laws are known as *coherence laws*. In a weak 2-category, the coherence laws are the following fact(LEINSTER, 1998):

Given the following diagram of $1 - morphisms$:

$$\begin{tikzcd}
a \arrow[r, "s"] & b \arrow[r, "r"] & c \arrow[r, "p"] & d \arrow[r, "u"] & e
\end{tikzcd}$$

The following diagrams should commute:

$$\begin{tikzcd}
((u \circ p) \circ r) \circ s \arrow[rr, "assoc \circ_h [\rho_s]_{rw_2}"'] \arrow[d, "assoc"'] & & (u \circ (p \circ r)) \circ s \arrow[d, "assoc"] \\
(u \circ p) \circ (r \circ s) \arrow[dr, "assoc"'] & & u \circ ((p \circ r) \circ s) \arrow[dl, "{[\rho_u]_{rw_2} \circ_h assoc}"] \\
& u \circ (p \circ (r \circ s)) &
\end{tikzcd}$$

$$\begin{tikzcd}
(r \circ \rho_b) \circ s \arrow[rr, "assoc"'] \arrow[dr, "{r_r^* \circ_h [\rho_s]_{rw_2}}"'] & & r \circ (\rho_b \circ s) \arrow[dl, "{[\rho_r]_{rw_2} \circ_h l_s^*}"] \\
& r \circ s &
\end{tikzcd}$$

The proofs are straightforward. Remember that $assoc$ is just an application of $\sigma(tt)$, r_r^* is an application of trr and l_s^* an application of tlr. First, let's start with $((u \circ p) \circ r) \circ s = \tau(s, \tau(r, \tau(p, u)))$ going to the right of the diagram:

$$(assoc \circ_h [\rho_s]_{rw_2})(\tau(s, \tau(r, \tau(p, u)))) = \tau(s, assoc(\tau(r, \tau(p, u)))) =$$
$$\tau(s, \tau(\tau(r, p), u))$$
$$assoc(\tau(s, \tau(\tau(r, p), u))) = \tau(\tau(s, \tau(r, p)), u)$$
$$([\rho_u]_{rw_2} \circ_h assoc)(\tau(\tau(s, \tau(r, p)), u)) = \tau(assoc(\tau(s, \tau(r, p))), u) =$$
$$\tau(\tau(\tau(s, r), p)), u) = u \circ (p \circ (r \circ s))$$

Now, starting from the same $\tau(s, \tau(r, \tau(p, u)))$ and going bottom left:

$$assoc(\tau(s, \tau(r, \tau(p, u)))) = \tau(\tau(s, r), \tau(p, u))$$
$$assoc(\tau(\tau(s, r), \tau(p, u))) = \tau(\tau(\tau(s, r), p), u) = u \circ (p \circ (r \circ s))$$

Therefore, the first diagram commutes. We now need to check the second one. Let's start from $((r \circ \rho_b) \circ s) = \tau(s, \tau(\rho_b, r))$ and going to the right of the diagram:

$$assoc(\tau(s,\tau(\rho_b,r))) = \tau(\tau(s,\rho_b),r)$$
$$([\rho_r]_{rw_2} \circ_h l_s^*)\tau(\tau(s,\rho_b),r) = \tau(l_s^*(\tau(s,\rho_b)),r) =$$
$$\tau(s,r) = r \circ s$$

Now, starting from the same $\tau(s,\tau(\rho_b,r))$ and going right bottom:

$$(r_r^* \circ_h [\rho_s]_{rw_2})\tau(s,\tau(\rho_b,r)) = \tau(s, r_r^*(\tau(\rho_b,r))) =$$
$$\tau(s,r) = r \circ s$$

Thus, the second diagram commutes. The coherence laws hold. We finally finish the proof that $[2 - A_{rw}]$ is a weak 2-category. □

We can also conclude that $[2 - A_{rw}]$ has a weak 2-groupoid structure. That is the case because we already know (from *proposition 4.4*) that the groupoid laws are satisfied by 1-morphisms up to the isomorphism of the next level, i.e., up to rw-equality and the 2-morphisms, as we have just seen, are isomorphisms (that hold in a strict way, since the second level is using classes of equivalence). With that, we showed that computational paths induces a 2-weak groupoid. If one compares this groupoid with the one obtained by the homotopy interpretation, this 2-weak category is similar to the fundamental weak 2-groupoid of a space S, denoted by $\Pi_2 S$ (that $\Pi_2 S$ forms a weak 2-category can be seen in (LEINSTER, 2004)).

Since we could induce a weak 2-categorical structure using computational paths, would be possible to induce even higher structures? The answer is *yes*, since we have infinite levels of rw-rules established by infinite $LND_{EQ} - TRS_n$ systems. For example, we could add a new level $A_{3rw}(\theta, \alpha)$. where θ and α are 2-morphisms. We would have a structure with 3 levels and we could try to prove that this structure is a weak 3-category. The problem is, as one could see in the proof of *proposition 4.5*, working with higher structures can be difficult. A weak 3-weak category has more types of compositions than the 2-weak one, since we have an additional level. Moreover, since it is weak, there are coherence laws that must be checked. For a 3-weak category, these laws are much more complicated than the ones for 2-weak categories. For that reason, we will leave the study of higher structures induced by computational paths with more than 2 levels for the future, since it is still work in progress. In fact, our aim is to prove, in a future work, that computational path induces a weak category with infinite levels, known as weak ω-category. In fact, we want to show that this weak infinite category forms a weak ω-groupoid. We believe that it is possible to achieve these results, since it was proved by(LUMSDAINE, 2009; BERG; GARNER, 2011) that the identity type induces such structure. Given the connection between computational paths and terms of identity types, we should be able to prove that computational paths also induces a weak ω-groupoid.

4.4 UNIQUENESS OF IDENTITY PROOFS

One of the main results that arises from the groupoid interpretation of(HOFMANN; STREICHER, 1998) is that it refutes the uniqueness of identity proofs, also known as UIP.(HOFMANN; STREICHER, 1998) defines UIP as the following property:

Definition 4.26. *Let a and b be objects of a type A. UIP is the following property: given any proofs p and q of proposition a equals to b, then there is always another proof establishing the equality of p and q.*

In terms of computational paths, UIP asks if, for every pair of objects t and s between a and b of a type A, there is an rw-equality $t =_{rw_\theta} s$. To better see this situation, a concrete example is given by(QUEIROZ; OLIVEIRA; RAMOS, 2016):

Example 4.7. *Consider the term $(\lambda x.(\lambda y.yx)(\lambda w.zw))v$. We want to reduce this term to the term zv. To achieve this goal, we can follow 3 different paths:*

- $(\lambda x.(\lambda y.yx)(\lambda w.zw))v \rhd_\eta (\lambda x.(\lambda y.yx)z)v \rhd_\beta (\lambda y.yv)z \rhd_\beta zv.$
- $(\lambda x.(\lambda y.yx)(\lambda w.zw))v \rhd_\beta (\lambda x.(\lambda w.zw)x)v \rhd_\eta (\lambda x.zx)v \rhd_\beta zv.$
- $(\lambda x.(\lambda y.yx)(\lambda w.zw))v \rhd_\beta (\lambda x.(\lambda w.zw)x)v \rhd_\beta (\lambda w.zw)v \rhd_\eta zv.$

Thus, if UIP holds, those 3 paths are just different ways of expressing the same thing. In other words, it would be possible to find rw-equalities establishing equalities between them.

The objective of this section is to show that, similarly to the traditional approach for the identity type, our path-based one also refutes UIP. Given a type A, by the definition of UIP, one can conceive a type $UIP(A)$ that is inhabited iff for any terms $a, b : A$ and paths $a =_s b$ and $a =_t b$, then $s =_{rw} t$.

As stated by(HOFMANN; STREICHER, 1998), if UIP holds, then every type A is a trivial groupoid structure in which there is only one morphism between every pair of objects. To show that UIP is false, one only needs to construct an A such that there is more than one morphism between a pair of objects (i.e., there is at least one pair of objects $a, b : A$ and a pair of paths s and t between them such that $sthat\ s \neq_{rw} t$). To achieve that, we can take advantage of our previous example, and consider $(\lambda x.(\lambda y.yx)(\lambda w.zw))v : A$. Therefore (RAMOS; QUEIROZ; OLIVEIRA, 2017; QUEIROZ; OLIVEIRA; RAMOS, 2016):

Theorem 4.4. *The type UIP is empty.*

Proof. We construct A_{rw}:

$$\begin{tikzcd}
& (\lambda x.(\lambda y.yx)(\lambda w.zw))v \arrow[dl, "\eta"'] \arrow[dr, "\beta"] \arrow[ddd, bend right=20, "\tau(\tau(\eta,\beta),\beta)"'] \arrow[ddd, bend left=20, "\tau(\tau(\beta,\beta),\eta)"] & \\
(\lambda x.(\lambda y.yx)x)v \arrow[d, "\beta"'] & & (\lambda x.(\lambda w.zw)x)v \arrow[d, "\beta"] \\
(\lambda y.yv)z \arrow[dr, "\beta"'] & & (\lambda w.zw)v \arrow[dl, "\eta"] \\
& zv &
\end{tikzcd}$$

As depicted above, there are (at least) two possible paths between $(\lambda x.(\lambda y.yx)(\lambda w.zw))v$ and zv. The first path is given by $\tau(\tau(\eta,\beta),\beta)$ and the second by $\tau(\tau(\beta,\beta),\eta)$. Moreover, looking at all rw-rules (check **definition 4.18**), there is no rule that establishes the rw-equality between these two paths. That way, A_{rw} is not a trivial groupoid and thus, UIP is empty. □

With that, we conclude that our approach based on computational paths refutes the uniqueness of identity proofs.

4.5 CONCLUSION

Inspired by a recent discovery that the propositional equality between terms can be interpreted as the type of homotopy paths, we have revisited the formulation of the intensional identity type, proposing a approach based on an entity known as *computational path*. We have proposed that a computational path $a =_s b : A$ gives grounds to building a term $s(a,b)$ of the identity type, i.e., $s(a,b) : Id_A(a,b)$, and is formed by a composition of basic rewrites, each with their identifiers taken as constants. We have also developed our approach, showing how the path-based identity type can be rather straightforwardly used in deductions. In particular, we have shown the simplicity of our elimination rule, demonstrating that it is based on path constructions, which are built from applications of simple axioms of the equality for type theory. To make our point even clearer, we have exposed three path-based constructions. More specifically, constructions that prove the transitivity, reflexivity and symmetry of the propositional equality. We have also argued that, for these constructions, the process of finding the reason that allows for building the desired term was simple and straightforward in our approach. At the same time, in the traditional (or pathless) approach, this is not entirely true, since finding the correct reason was a cumbersome process.

After establishing the foundations of our approach, we analyzed one important structure that the traditional identity type induces: the algebraic structure known as groupoid. Our objective was to show that our approach is on a par with the pathless one, i.e., our path-based identity also induces a groupoid structure. To prove that, we have shown that the axioms of equality generate redundancies, which are resolved by paths between paths. We mentioned that there already exists a system called $LND_{EQ} - TRS$ that maps and resolves these redundancies. We have gone further, proposing the existence of a higher $LND_{EQ} - TRS_2$ which resolves redundancies generated by the $LND_{EQ} - TRS$ system. Using $LND_{EQ} - TRS$, we have proved that a computational path is capable of inducing a weak groupoid structure. Using the higher rewriting system, we have induced a structure known as weak 2-groupoid. With that, we believe we have opened the way, in a future work, for a possible proof establishing that computational paths induces a weak ω-groupoid.

We have also shown that, using our groupoid of computational paths, it is possible to refute the principle of uniqueness of identity proofs for the path-based approach. This result ties in with the one obtained by (HOFMANN; STREICHER, 1998) for the traditional identity type.

5 HOMOTOPY TYPE THEORY

In **chapter 2**, we said that one of the most interesting concepts of type theory is the identity type. We have also said that the reason for that is the fact one can see the identity type as a homotopical path between two points of a space, giving rise to a homotopical interpretation of type theory. The connection between those two theories created a whole new area of research known as homotopy type theory. In this work, we introduced computational paths as the syntactic counterpart of those homotopical paths, since they only exist in a semantical sense. Nevertheless, we have not talked yet how one can use computational paths in homotopy type theory.

Thus, in this chapter, we develop one of the main objectives of this work.

We want to show that some of the foundational definitions, propositions and theorems of homotopy type theory still hold in our path-based approach. In other words, we use our approach to construct the building blocks of more complex results.

One important fact to notice is that every proof that does not involve the identity type is valid in the path-based approach. This is obvious, since the only difference between the traditional approach and ours is the formulation of the identity type. If a proof uses it, we need to reformulate this proof using our path-based approach, instead of using the induction principle of the traditional one. Thus, every part of a proof that is not directly or indirectly related to identity type is still valid in our approach.

In a path-based proof, we are going to use the formulation proposed in the previous chapter. We also are going to use the reduction rules of $LND_{EQ} - TRS$. In the process of developing the theory of this section, we noticed that $LND_{EQ} - TRS$, as proposed in the previous chapter is still incomplete. We state this based on the fact that we found new reduction rules that are not part of the original $LND_{EQ} - TRS$. That way, we added these new rules to the system, expanding it.

5.1 GROUPOID LAWS

In the previous chapter we have seen that computational paths form a groupoid structure. Let's check again those rules using our $REWR$ constructor directly:

Lemma 5.1. *The type $\Pi_{(a:A)} Id_A(a,a)$ is inhabited.*

Proof. We construct an witness for the desired type:

Parts of this chapter were introduced by the author in the talk *Computational paths, transport and the univalence axiom* at *XVIII Brazilian Logic Conference* in 2017. (RAMOS, 2017). This chapter also yielded an still unpublished paper, "Explicit Computational Paths". Preprint version available in (RAMOS; QUEIROZ; OLIVEIRA, 2018)

$$\dfrac{\dfrac{\dfrac{[a : A]}{a =_{\rho} a : A}}{\rho(a,a) : Id_A(a,a)}\, Id - I_1}{\lambda a.\rho(a,a) : \Pi_{(a:A)} Id_A(a,a)}\, \Pi - I$$

□

Lemma 5.2. *The type* $\Pi_{(a:A)}\Pi_{(b:A)}(Id_A(a,b) \rightarrow Id_A(b,a))$ *is inhabited.*

Proof. Similar to the previous lemma, we construct an witness:

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}[a : A] \quad [b : A] \\ [p(a,b) : Id_A(a,b)]\end{matrix} \qquad \dfrac{\dfrac{[a =_t b : A]}{b =_{\sigma(t)} a : A}}{(\sigma(t))(b,a) : Id_A(b,a)}\, Id - I}{REWR(p(a,b), \acute{t}.(\sigma(t))(b,a)) : Id_A(b,a)}\, Id - E}{\lambda p.REWR(p(a,b), \acute{t}.(\sigma(t))(b,a)) : Id_A(a,b) \rightarrow Id_A(b,a)}\, \Pi - I}{\lambda b.\lambda p.REWR(p(a,b), \acute{t}.(\sigma(t))(b,a)) : \Pi_{(b:A)}(Id_A(a,b) \rightarrow Id_A(b,a))}\, \Pi - I}{\lambda a.\lambda b.\lambda p.REWR(p(a,b), \acute{t}.(\sigma(t))(b,a)) : \Pi_{(a:A)}\Pi_{(b:A)}(Id_A(a,b) \rightarrow Id_A(b,a))}\, \Pi - I$$

□

Lemma 5.3. *The type* $\Pi_{(a:A)}\Pi_{(b:A)}\Pi_{(c:A)}(Id_A(a,b) \rightarrow Id_A(b,c) \rightarrow Id_A(a,c))$ *is inhabited.*

Proof. We construct the following witness:

□

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}[a : A] \quad [b : A] \\ [w(a,b) : Id_A(a,b)]\end{array} \qquad \dfrac{\begin{array}{c}[c : A] \\ [s(b,c) : Id_A(b,c)]\end{array} \qquad \dfrac{\dfrac{[a =_t b : A] \qquad [b =_u c : A]}{a =_{\tau(t,u)} c : A}}{(\tau(t,u))(a,c) : Id_A(a,c)}\,Id - I}{REWR(s(b,c), \acute{u}(\tau(t,u))(a,c)) : Id_A(a,c)}\,Id - E}{REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : Id_A(a,c)}\,Id - E}{\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : Id_A(b,c) \to Id_A(a,c)}\,\Pi - I}{\lambda w.\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c)}\,\Pi - I}{\lambda c.\lambda w.\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : \Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))}\,\Pi - I}{\lambda b.\lambda c.\lambda w.\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : \Pi_{(b:A)}\Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))}\,\Pi - I}{\lambda a.\lambda b.\lambda c.\lambda w.\lambda s.REWR(w(a,b), \acute{t}REWR(s(b,c), \acute{u}(\tau(t,u))(a,c))) : \Pi_{(a:A)}\Pi_{(b:A)}\Pi_{(c:A)}(Id_A(a,b) \to Id_A(b,c) \to Id_A(a,c))}\,\Pi - I
$$

Lemmas **1**, **2** and **3** correspond respectively to the reflexivity, symmetry and transitivity of the identity type. From now on, the reflexivity will be represented by ρ, symmetry by σ and transitivity by τ.

Lemma 5.4. *For any type A, $x, y, z, w : A$ and $p : Id_A(x, y)$ and $q : Id_A(y, z)$ and $r : Id_A(z, w)$, the following types are inhabited:*

1. $\Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(p, \rho_y \circ p)$ *and* $\Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(p, p \circ \rho_x)$.
2. $\Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(\sigma(p) \circ p, \rho_x)$ *and* $\Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(p \circ \sigma(p), \rho_y)$
3. $\Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(\sigma(\sigma(p)), p)$
4. $\Pi_{(x,y,z,w:A)}\Pi_{(p:Id_A(x,y))}\Pi_{(q:Id_A(y,z))}\Pi_{(r:Id_A(z,w))}Id_{Id_A(x,w)}(r \circ (q \circ p), (r \circ q) \circ p)$

Proof. The proof of each statement follows from the same idea. We just need to look for suitable reduction rules already present in the original $LND_{EQ} - TRS$.

1. The first thing to notice is that a composition in our path-based approach corresponds to a transitive operation, i.e., $(p \circ \rho_x)$ can be written as $\tau(\rho_x, p)$. The terms follow from rules number **5** and **6**:

$$\frac{x =_r y : A \qquad y =_\rho y : A}{x =_{\tau(r,\rho)} y : A} \quad \rhd_{trr} \quad x =_r y : A$$

$$\frac{x =_\rho x : A \qquad x =_r y : A}{x =_{\tau(\rho,r)} y : A} \quad \rhd_{tlr} \quad x =_r y : A$$

Thus, we have:

$$\dfrac{\dfrac{\tau(p, \rho_y) =_{trr} p : Id_A(x, y)}{(trr)(\tau(p, \rho_y), p) : Id_{Id_A(x,y)}(p, \rho_y \circ p)}}{\lambda x.\lambda y.\lambda p.(trr)(\tau(p, \rho_y), p) : \Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(p, \rho_y \circ p)}$$

$$\dfrac{\dfrac{\tau(\rho_x, p) =_{tlr} p : Id_A(x, y)}{(tlr)(\tau(\rho_x, p), p) : Id_{Id_A(x,y)}(p, p \circ \rho_x)}}{\lambda x.\lambda y.\lambda p.(tlr)(\tau(\rho_x, p), p) : \Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(p, p \circ \rho_x)}$$

2. We use rules **3** and **4**:

$$\dfrac{x =_r y : A \qquad y =_{\sigma(r)} x : A}{x =_{\tau(r,\sigma(r))} x : A} \quad \rhd_{tr} \quad x =_\rho x : A$$

$$\dfrac{y =_{\sigma(r)} x : A \qquad x =_r y : A}{y =_{\tau(\sigma(r),r)} y : A} \quad \rhd_{tsr} \quad y =_\rho y : A$$

Thus:

$$\dfrac{\dfrac{\tau(p,\sigma(p)) =_{tr} \rho_x : Id_A(x,y)}{(tr)(\tau(p,\sigma(p)),\rho_x) : Id_{Id_A(x,y)}(\sigma(p)\circ p,\rho_x)}}{\lambda x.\lambda y.\lambda p.(tr)(\tau(p,\sigma(p),\rho_x) : \Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(\sigma(p)\circ p,\rho_x)}$$

$$\dfrac{\dfrac{\tau(\sigma(p),p) =_{tsr} \rho_y : Id_A(x,y)}{(tsr)(\tau(\sigma(p),p),\rho_y) : Id_{Id_A(x,y)}(p\circ\sigma(p),\rho_y)}}{\lambda x.\lambda y.\lambda p.(tsr)(\tau(p,\sigma(p),\rho_y) : \Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(p\circ\sigma(p),\rho_y)}$$

3. We use rule **2**:

$$\dfrac{\dfrac{x =_r y : A}{y =_{\sigma(r)} x : A}}{x =_{\sigma(\sigma(r))} y : A} \quad \rhd_{ss} \quad x =_r y : A$$

Thus:

$$\dfrac{\dfrac{\sigma(\sigma(p)) =_{ss} p : Id_A(x,y)}{(ss)(\sigma(\sigma(p),p)) : Id_{Id_A(x,y)}(\sigma(\sigma(p)),p)}}{\lambda x.\lambda y.\lambda p.(ss)(\sigma(\sigma(p),p)) : \Pi_{(x,y:A)}\Pi_{(p:Id_A(x,y))}Id_{Id_A(x,y)}(\sigma(\sigma(p)),p)}$$

4. We use rule **37**:

$$\dfrac{\dfrac{x =_t y : A \qquad y =_r w : A}{x =_{\tau(t,r)} w : A} \qquad w =_s z : A}{x =_{\tau(\tau(t,r),s)} z : A}$$

$$\rhd_{tt} \dfrac{x =_t y : A \qquad \dfrac{y =_r w : A \qquad w =_s z : A}{y =_{\tau(r,s)} z : A}}{x =_{\tau(t,\tau(r,s))} z : A}$$

Thus:

$$\dfrac{\dfrac{\tau(\tau(p,q),r) =_{tt} \tau(p,\tau(q,r)) : Id_A(x,w)}{(tt)(\tau(\tau(p,q),r) =_{tt} \tau(p,\tau(q,r))) : Id_{Id_A(x,w)}(r \circ (q \circ p), (r \circ q) \circ p)}}{\lambda x.\lambda y.\lambda z.\lambda w.\lambda p.\lambda q.\lambda r.(ss)(\sigma(\sigma(p),p)) : \Pi_{(p:Id_A(x,y))}\Pi_{(q:Id_A(y,z))}\Pi_{(r:Id_A(z,w))} Id_{Id_A(x,w)}(r \circ (q \circ p), (r \circ q) \circ p)}$$

□

With the previous lemma, we showed that our path-based approach yields the groupoid structure of a type up to propositional equality using the syntax introduced in the previous chapter.

5.2 FUNCTORIALITY

We want to show that functions preserve equality(Univalent Foundations Program, 2013).

Lemma 5.5. *The type* $\Pi_{(x,y:A)}\Pi_{(f:A\to B)}(Id_A(x,y)\to Id_B(f(x),f(y)))$ *is inhabited.*

Proof. It is a straightforward construction:

$$\dfrac{\dfrac{\dfrac{\dfrac{[x =_s y : A] \qquad [f : A \to B]}{f(x) =_{\mu_f(s)} f(y) : B}}{\mu_f(s)(f(x),f(y)) : Id_B(f(x),f(y))} \qquad [p : Id_A(x,y)]}{REWR(p,\lambda s.\mu_f(s)(f(x),f(y))) : Id_B(f(x),f(y))}}{\lambda x.\lambda y.\lambda f.\lambda p.REWR(p,\lambda s.\mu_f(s)(f(x),f(y))) : \Pi_{(x,y:A)}\Pi_{(f:A\to B)}(Id_A(x,y)\to Id_B(f(x),f(y)))}$$

□

Lemma 5.6. *For any functions* $f : A \to B$ *and* $g : B \to C$ *and paths* $p : x =_A y$ *and* $q : y =_A z$*, we have:*

1. $\mu_f(\tau(p,q)) = \tau(\mu_f(p),\mu_f(q))$
2. $\mu_f(\sigma(p)) = \sigma(\mu_f(p))$
3. $\mu_g(\mu_f(p)) = \mu_{g\circ f}(p)$
4. $\mu_{Id_A}(p) = p$

Proof. 1. For the first time, we need to add a new rule to the original 39 rules of $LND_{EQ} - TRS$. We introduce rule **40**:

$$\dfrac{\dfrac{x =_p y : A \qquad [f : A \to B]}{f(x) =_{\mu_f(p)} f(y) : B} \qquad \dfrac{y =_q z : A \qquad [f : A \to B]}{f(y) =_{\mu_f(q)} f(z) : B}}{f(x) = \tau(\mu_f(p),\mu_f(q)) f(z) : B}$$

$$\rhd_{tf} \dfrac{\dfrac{x =_p y : A \qquad y =_q z : A}{x =_{\tau(p,q)} z : A} \qquad f : A \to B}{f(x) =_{\mu_f(\tau(p,q))} f(z) : B}$$

Thus, we have $\mu_f(\tau(p,q)) =_{\sigma(tf)} \tau(\mu_f(p),\mu_f(q))$

2. This one follows from rule **30**:

$$\dfrac{\dfrac{x =_p y : A \qquad [f : A \to B]}{f(x) =_{\mu_f(p)} f(y) : B}}{f(y) =_{\sigma(\mu_f(p))} f(x) : B}$$

$$\rhd_{sm} \dfrac{\dfrac{x =_p y : A}{y =_{\sigma(p)} x : A} \qquad [f : A \to B]}{f(y) =_{\mu_f(\sigma(p))} f(x) : B}$$

We have $\mu_f(\sigma(p)) =_{\sigma(sm)} \sigma(\mu_f(p))$

3. We introduce rule **41**:

$$\dfrac{\dfrac{x =_p y : A \qquad [f : A \to B]}{f(x) =_{\mu_f(p)} f(y) : B} \qquad [g : B \to C]}{g(f(x)) =_{\mu_g(\mu_f(p))} g(f(y)) : C}$$

$$\rhd_{cf} \dfrac{x =_p y : A \qquad \dfrac{\dfrac{\dfrac{[x : A] \qquad [f : A \to B]}{f(x) : B} \qquad [g : B \to C]}{g(f(x)) : C}}{\lambda x.g(f(x)) \equiv (g \circ f) : A \to C}}{g(f(x)) =_{\mu_{g \circ f}(p)} g(f(y)) : C}$$

Then, $\mu_g(\mu_f(p)) =_{cf} \mu_{g \circ f}(p)$

4. We introduce rule **42**:

$$\dfrac{\dfrac{x =_p y : A \qquad [Id_A : A \to A]}{Id_A(x) = \mu_{Id_A(p)} Id_A(y) : A}}{x =_{\mu_{Id_A}(p)} y : A} \qquad \rhd_{ci} \qquad x =_p y : A$$

It follows that $\mu_{Id_A}(p) =_{ci} p$

$\square$

5.3 TRANSPORT

As stated in (QUEIROZ; OLIVEIRA, 2014a), substitution can take place when no quantifier is involved. In this sense, there is a 'quantifier-less' notion of substitution. In type theory, this 'quantifier-less' substitution is given by a operation known as transport (Univalent Foundations Program, 2013). In our path-based approach, we formulate a new inference rule of 'quantifier-less' substitution (QUEIROZ; OLIVEIRA, 2014a):

$$\frac{x =_p y : A \qquad f(x) : P(x)}{p(x,y) \circ f(x) : P(y)}$$

We use this transport operation to solve one essential issue of our path-based approach. We know that given a path $x =_p y : A$ and function $f : A \to B$, the application of axiom μ yields the path $f(x) =_{\mu_f(p)} f(y) : B$. The problem arises when we try to apply the same axiom for a dependent function $f : \Pi_{(x:A)} P(x)$. In that case, we want $f(x) = f(y)$, but we cannot guarantee that the type of $f(x) : P(x)$ is the same as $f(y) : P(y)$. The solution is to apply the transport operation and thus, we can guarantee that the types are the same:

$$\frac{x =_p y : A \qquad f : \Pi_{(x:A)} P(x)}{p(x,y) \circ f(x) =_{\mu_f(p)} f(y) : P(y)}$$

Lemma 5.7. *(Leibniz's Law) The type $\Pi_{(x,y:A)}(Id_A(x,y) \to P(x) \to P(y))$ is inhabited.*

Proof. We construct the following tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{[x =_p y : A] \qquad [f(x) : P(x)]}{p(x,y) \circ f(x) : P(y)}}{\lambda f(x).p(x,y) \circ f(x) : P(x) \to P(y)} \qquad [z : Id_A(x,y)]}{REWR(z, \lambda p.\lambda f(x).p(x,y) \circ f(x)) : P(x) \to P(y)}}{\lambda x.\lambda y.\lambda z.REWR(z, \lambda p.\lambda f(x).p(x,y) \circ f(x)) : \Pi_{(x,y:A)}(Id_A(x,y) \to P(x) \to P(y))}$$

□

The function $\lambda f(x).p(x,y) \circ f(x) : P(x) \to P(y)$ is usually written as $transport^p(p, -)$ and $transport^p(p, f(x)) : P(y)$ is usually written as $p_*(f(x))$.

Lemma 5.8. *For any $P(x) \equiv B$, $x =_p y : A$ and $b : B$, there is a path $transport^P(p, b) = b$.*

Proof. The first to notice is the fact that in our formulation of transport, we always need a functional expression $f(x)$, and in this case we have only a constant term b. To address this problem, we consider a function $f = \lambda.b$ and then, we transport over $f(x) \equiv b$:

$$transport^P(p, f(x) \equiv b) =_{\mu(p)} (f(y) \equiv b).$$

Thus, $transport^P(p, b) =_{\mu(p)} b$. We sometimes call this path $transportconst^B_p(b)$.

□

Lemma 5.9. *For any $f : A \to B$ and $x =_p y : A$, we have*

$$\mu(p)(p_*(f(x)), f(y)) = \tau(transportconst^B_p, \mu_f(p))(p_*(f(x)), f(y))$$

Proof. The first thing to notice is that in this case, $transportconst^B_p$ is the path $\mu(p)(p * (f(x), f(x))$ by lemma **8**. As we did to the rules of $LND_{EQ} - TRS$, we establishes this equality by getting to the same conclusion from the same premises by two different trees:

In the first tree, we consider $f(x) \equiv b : B$ and transport over $b : B$:

$$\dfrac{\dfrac{\dfrac{x =_p y : A \qquad f(x) \equiv b : B}{p(x,y) \circ (f(x) \equiv b) : B}}{p_*(f(x)) =_{\mu_f(p)} b \equiv f(x)} \qquad \dfrac{x =_p y : A \qquad f : A \to B}{f(x) =_{\mu_f(p)} f(y) : B}}{p_*(f(x)) =_{\tau(\mu_f(p), \mu_f(p))} f(y) : B}$$

In the second one, we consider $f(x)$ as an usual functional expression and thus, we transport the usual way:

$$\dfrac{\dfrac{x =_p y : A \qquad f(x) : B}{p(x,y) \circ f(x) : B}}{p_*(f(x)) =_{\mu_f(p)} f(y) : B}$$

□

Lemma 5.10. *For any $x =_p y : A$ and $q : y =_A z : A$, $f(x) : P(x)$, we have*

$$q_*(p_*(f(x))) = (p \circ q)_*(f(x))$$

Proof. We develop both sides of the equation and wind up with the same result:

$$q_*(p_* f(x)) =_{\mu(p)} q_*(f(y)) =_{\mu(q)} f(z)$$
$$(p \circ q)_*(f(x)) =_{\mu(p \circ q)} f(z)$$

□

Lemma 5.11. *For any $f : A \to B$, $x =_p y : A$ and $u : P(f(x))$, we have:*

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

Proof. This lemma hinges on the fact that there is two possible interpretations of u that stems from the fact that $(g \circ f)(x) \equiv g(f(x))$. Thus, we can see u as functional expression g on $f(x)$ or an expression $g \circ f$ on x:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x =_p y : A \qquad u \equiv (g \circ f)(x) : (P \circ f)(x)}{p(x,y) \circ (g \circ f)(x) : (P \circ f)(y)}}{p(x,y) \circ (g \circ f)(x) =_{\mu(p)} (g \circ f)(y) : (P \circ f)(y)}}{p(x,y) \circ (g \circ f)(x) =_{\mu(p)} g(f(y)) : P(f(y))} \qquad \dfrac{\dfrac{\dfrac{\dfrac{x =_p y : A}{f(x) =_{u_f(p)} f(y) : B} \qquad u \equiv g(f(x)) : P(f(x))}{\mu_f(p)(f(x), f(y)) \circ g(f(x)) : P(f(y))}}{\mu_f(p)(f(x), f(y)) \circ g(f(x)) =_{\mu(p)} g(f(y)) : P(f(y))}}{g(f(y)) =_{\sigma(\mu(p))} \mu_f(p)(f(x), f(y)) \circ g(f(x)) : P(f(y))}}{p(x,y) \circ (g \circ f)(x) =_{\tau(\mu(p),\sigma(\mu(p)))} \mu_f(p)(f(x), f(y)) \circ g(f(x)) : P(f(y))}}{transport^{P \circ f}(p, u) =_{\tau(\mu(p),\sigma(\mu(p)))} transport^{P}(\mu_f(p), u)}$$

□

Lemma 5.12. *For any $f : \Pi_{(x:A)} P(x) \to Q(x)$, $x =_p y : A$ and $u(x) : P(x)$, we have:*

$$transport^Q(p, f(u(x))) = f(transport^P(p, u(x)))$$

Proof. We proceed the usual way, constructing a derivation tree that establishes the equality:

$$\dfrac{\dfrac{\dfrac{\dfrac{x =_p y : A \qquad f(u(x)) : Q(x)}{p(x,y) \circ f(u(x)) : Q(y)}}{p(x,y) \circ f(u(x)) =_{\mu(p)} f(u(y)) : Q(y)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{x =_p y : A \qquad u(x) : P(x)}{p(x,y) \circ u(x) : P(y)}}{p(x,y) \circ u(x) =_{\mu(p)} u(y) : P(y)} \qquad f : \Pi_{(x:A)} P(x) \to Q(x)}{f(p(x,y) \circ u(x)) =_{\mu_f(\mu(p))} f(u(y)) : Q(y)}}{f(u(y)) =_{\sigma(\mu_f(\mu(p)))} f(p(x,y) \circ u(x)) : Q(y)}}{p(x,y) \circ f(u(x)) =_{\tau(\mu(p),\sigma(\mu_f(\mu(p))))} f(p(x,y) \circ u(x))}}{transport^Q(p, f(u(x))) =_{\tau(\mu(p),\sigma(\mu_f(\mu(p))))} f(transport^P(p, u(x)))}$$

□

5.4 HOMOTOPIES

In Homotopy Type Theory, a homotopy is defined as follows (Univalent Foundations Program, 2013):

Definition 5.1. *For any $f, g : \Pi_{(x:A)} P(x)$, a homotopy from f to g is a dependent function of type:*

$$(f \sim g) \equiv \Pi_{(x:A)}(f(x) = g(x))$$

In our path-based approach, we have a homotopy $f, g : \Pi_{(x:A)} P(x)$ if for every $x : A$ we have a computational path between $f(x) = g(x)$. Thus, if we have a homotopy $H_{f,g} : f \sim g$, we derive the following rule:

$$\dfrac{H_{f,g} : f \sim g \qquad f, g : \Pi_{(x:A)} P(x) \qquad x : A}{f(x) =_{H_{f,g}(x)} g(x) : P(x)}$$

And:

$$\dfrac{f, g : \Pi_{(x:A)} P(x) \qquad x : A \qquad \begin{matrix}[f, g : \Pi_{(x:A)} P(x), x : A] \\ f(x) =_p g(x)\end{matrix}}{H^p_{f,g} : f \sim g}$$

Lemma 5.13. *For any $f, g, h : A \to B$, the following types are inhabited:*

1. $f \sim f$
2. $(f \sim g) \to (g \sim f)$
3. $(f \sim g) \to (g \sim h) \to (f \sim h)$

Proof. 1. We construct the following term:

$$\dfrac{f : A \to B \qquad x : A \qquad \dfrac{\dfrac{[x : A]}{x =_\rho x} \qquad [f : A \to B]}{f(x) =_{\mu_f(\rho)} f(x) : B}}{H^{\mu_f(\rho)}_{f,f} : f \sim f}$$

2. We construct:

$$\dfrac{\dfrac{f, g : A \to B \qquad x : A \qquad \dfrac{\dfrac{[H_{f,g} : f \sim g] \qquad [f, g : A \to B] \qquad [x : A]}{f(x) =_{H_{f,g}(x)} g(x) : B}}{g(x) =_{\sigma(H_{f,g}(x))} f(x) : B}}{H^{\sigma(H_{f,g}(x))}_{g,f} : g \sim f}}{\lambda H_{f,g}.H^{\sigma(H_{f,g}(x))}_{g,f} : (f \sim g) \to (g \sim f)}$$

3. We construct:

$$\dfrac{\dfrac{f, h : A \to B \qquad x : A \qquad \dfrac{\dfrac{[H_{f,g} : f \sim g] \qquad [f, g : A \to B] \qquad [x : A]}{f(x) =_{H_{f,g}(x)} g(x) : B} \qquad \dfrac{[H_{g,h} : g \sim h] \qquad [g, h : A}{g(x) =_{H_{g,h}(x)} h(}}{f(x) =_{\tau(H_{f,g}(x), H_{g,z}(x))} h(x) : B}}{H^{\tau(H_{f,g}(x), H_{g,z}(x))}_{f,h} : f \sim h}}{\lambda H_{f,g}.\lambda H_{g,h}.H^{\tau(H_{f,g}(x), H_{g,z}(x))}_{f,h} : (f \sim g) \to (g \sim h) \to (f \sim h)}$$

□

Lemma 5.14. *For any $H_{f,g} : f \sim g$ and functions $f, g : A \to B$ and a path $x =_p y : A$ we have:*

$$\tau(H_{f,g}(x), \mu_g(p)) = \tau(\mu_f(p), H_{f,g}(y))$$

Proof. To establish this equality, we need to add a new rule to our $LND_{EQ} - TRS$. We introduce rule **43**:

$$\dfrac{\dfrac{H_{f,g} : f \sim g \qquad x : A \qquad f,g : A \to B}{f(x) =_{H_{f,g}(x)} g(x) : B} \qquad \dfrac{x =_p y : A}{g(x) =_{\mu_g(p)} g(y) : B}}{f(x) =_{\tau(H_{f,g}(x),\mu_g(p))} g(y) : B}$$

$$\rhd_{hp}$$

$$\dfrac{\dfrac{x =_p y : A}{f(x) =_{\mu_f(p)} f(y) : B} \qquad \dfrac{H_{f,g} : f \sim g \qquad x : A \qquad f,g : A \to B}{f(y) =_{H_{f,g}(y)} g(y) : B}}{f(x) =_{\tau(\mu_f(p),H_{f,g}(y)} g(y) : B}$$

And thus:

$$\tau(H_{f,g}(x), \mu_g(p)) =_{hp} \tau(\mu_f(p), H_{f,g}(y))$$

□

After this section, we start to study specific lemmas and theorems involving basic types of type theory. Nevertheless, several of those theorems are statements about the notion of **equivalence** (notation: $\simeq$). Before we define equivalence, we need the following definition (Univalent Foundations Program, 2013):

Definition 5.2. *A **quasi-inverse** of a function $f : A \to B$ is a triple (g, α, β) such that g is a function $g : B \to A$ and α and β are homotopies such that $\alpha : f \circ g \sim Id_B$ and $\beta : g \circ f \sim Id_A$*

A quasi-inverse of f is usually written as $qinv(f)$.

Definition 5.3. *A function $f : A \to B$ is an equivalence if there is a quasi-inverse $qinv(f) : B \to A$.*

5.5 CARTESIAN PRODUCT

We start proving some important lemmas and theorems for the Cartesian product type. As we did in previous subsections, we proceed using our path-based approach. Before we prove our first theorem, it is important to remember that given a term $x : A \times B$, we can extract two projections, $FST(x) : A$ and $SND(x) : B$. Thus, given a path $x =_p y : A \times B$, we extract paths $FST(x) = SND(y) : A$ and $SND(x) = SND(y) : B$.

Theorem 5.1. *The function $(x =_p y : A \times B) \to (FST(x) = FST(y) : A) \times (SND(x) = SND(y) : B)$ is an equivalence for any x and y.*

Proof. To show the equivalence, we need to show the following

1. From $x =_p y : A \times B$ we want to obtain $(FST(x) = FST(y) : A) \times (SND(x) = SND(y) : B)$ and from that, we want to go back to $x =_p y : A \times B$.

2. We want to do the inverse process. From $(FST(x) = FST(y) : A) \times (SND(x) = SND(y) : B)$ we want to obtain $x =_p y : A \times B$ and then go back to $(FST(x) = FST(y) : A) \times (SND(x) = SND(y) : B)$.

To show the first part, we need rule **21**:

$$\dfrac{\dfrac{x =_p y : A \times B}{FST(x) =_{\mu_1(p)} FST(y) : A} \qquad \dfrac{x =_p y : A \times B}{SND(x) =_{\mu_2(p)} SND(y) : B}}{\langle FST(x), SND(x)\rangle =_{\epsilon(\mu_1(p),\mu_2(p))} \langle FST(y), SND(y)\rangle : A \times B}$$
$$\rhd_{mx}\ x =_p y : A \times B.$$

Thus, applying rule mx we showed the first part of our proof. For the second part, we need rules **14** and **15**:

$$\dfrac{\dfrac{x =_r x' : A \qquad y =_s z : B}{\langle x, y\rangle =_{\epsilon_\wedge(r,s)} \langle x', z\rangle : A \times B}}{FST(\langle x, y\rangle) =_{\mu_1(\epsilon_\wedge(r,s))} FST(\langle x', z\rangle) : A}$$
$$\rhd_{mx2l}\ x =_r x' : A.$$

And:

$$\dfrac{\dfrac{x =_r y : A \qquad z =_s w : B}{\langle x, z\rangle =_{\epsilon_\wedge(r,s)} \langle y, w\rangle : A \times B}}{FST(\langle x, z\rangle) =_{\mu_2(\epsilon_\wedge(r,s))} FST(\langle y, w\rangle) : B}$$
$$\rhd_{mx2r}\ z =_s w : B.$$

We also use the η-reduction for the Cartesian product:

$$\langle FST(x), SND(x)\rangle : A \times B \rhd_\eta x : A \times B$$

We construct the following derivation tree:

$$\dfrac{\dfrac{\dfrac{\langle FST(x) =_s FST(y), SND(x) =_t SND(y)\rangle}{FST(x) =_s FST(y) : A} \qquad \dfrac{\langle FST(x) =_s FST(y), SND(x) =_t SND(y)\rangle}{SND(x) =_t SND(y) : B}}{\langle FST(x), SND(x)\rangle =_{\epsilon_\wedge(s,t)} \langle FST(y), SND(y)\rangle : A \times B}}{x =_{\epsilon(s,t)} y : A \times B} \rhd_\eta$$

From $x =_{\epsilon(s,t)} y : A \times B$, we have:

$$\cfrac{\cfrac{\cfrac{x =_{\epsilon(s,t)} y : A \times B}{FST(x) =_{\mu_1(\epsilon_\wedge(s,t)} FST(y) : A} \qquad \cfrac{x =_{\epsilon(s,t)} y : A \times B}{SND(x) =_{\mu_2(\epsilon_\wedge(s,t)} SND(y) : B}}{\langle FST(x) =_{\mu_1(\epsilon_\wedge(s,t)} FST(y), SND(x) =_{\mu_2(\epsilon_\wedge(s,t)} SND(y)\rangle} \wedge - I}{\langle FST(x) =_{s} FST(y), SND(x) =_{t} SND(y)\rangle} \rhd_{mx2l,mx2r}$$

Thus, we showed part 2 and concluded the proof of this theorem. □

Theorem 5.2. *For any type families $\Pi_{(z:Z)}A, \Pi_{(z:Z)}B$ and a type family defined by $(A \times B)(z) \equiv A(z) \times B(z)$, a path $z =_p w : Z$ and $f(z) : A(z) \times B(z)$, we have:*

$$transport^{A \times B}(p, f(z)) = \langle transport^{A}(p, FST(f(z))), transport^{B}(p, SND(f(z)))\rangle : A(w) \times B(w)$$

Proof. We construct a derivation tree that establishes the equality:

$$\dfrac{\dfrac{\dfrac{\dfrac{z =_p w : Z \qquad f(z) : A(z) \times B(z)}{p(z,w) \circ f(z) : A(w) \times B(w)}}{p(z,w) \circ f(z) =_{\mu(p)} f(w) : A(w) \times B(w)}}{z, w) \circ f(z) =_{\tau(\mu(p),\eta)} \langle FST(f(w)), SND(f(w)) \rangle : A(w) \times B(w)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{z =_p w : Z \qquad FST(f(z)) : A(z)}{p(z,w) \circ FST(f(z)) : A(w)} \qquad \dfrac{z =_p w : Z \qquad SND(f(z)) : B(z)}{p(z,w) \circ SND(f(z)) : B(w)}}{\langle p(z,w) \circ FST(f(z)), p(z,w) \circ SND(f(z)) \rangle : A(w) \times B(w)}}{\langle p(z,w) \circ FST(f(z)), p(z,w) \circ SND(f(z)) \rangle =_{\mu(p)} \langle FST(f(w)), SND(f(w)) \rangle}}{\langle FST(f(w)), SND(f(w)) \rangle =_{\sigma(\mu(p))} \langle p(z,w) \circ FST(f(z)), p(z,w) \circ SND(f(z)) \rangle}}{\dfrac{p(z,w) \circ f(z) =_{\tau(\tau(\mu(p),\eta),\sigma(\mu(p)))} \langle p(z,w) \circ FST(f(z)), p(z,w) \circ SND(f(z)) \rangle : A(w) \times B(w)}{transport^{A \times B}(p, f(z)) =_{\tau(\tau(\mu(p),\eta),\sigma(\mu(p)))} \langle transport^{A}(p, FST(f(z))), transport^{B}(p, SND(f(z))) \rangle : A(w) \times B(w)}}$$

□

Theorem 5.3. *For any $x, y : A \times B$, $FST(x) =_p FST(y) : A$, $SND(x) =_q SND(y) : B$, functions $g : A \to A'$, $h : B \to B'$ and $f : A \times B \to A' \times B'$ defined by $f(x) \equiv \langle g(FST(x)), h(SND(x)\rangle$, we have:*

$$\mu_f(\epsilon_\wedge(p, q)) = \epsilon_\wedge(\mu_g(p), \mu_h(q))$$

Proof. We introduce rule **44**:

$$\cfrac{\cfrac{\cfrac{FST(x) =_p FST(y) : A \qquad SND(x) =_q SND(y) : B}{\langle FST(x), SND(x)\rangle =_{\epsilon_\wedge(p,q)} \langle FST(y), SND(y)\rangle : A \times B}}{x =_{\epsilon_\wedge(p,q)} y : A \times B} =_\eta}{f(x) =_{\mu_f(\epsilon_\wedge(p,q))} f(y) : A' \times B'}$$

$$\rhd_{mxc}$$

$$\cfrac{\cfrac{\cfrac{FST(x) =_p FST(y) : A}{g(FST(x)) =_{\mu_g(p)} g(FST(y)) : A'} \qquad \cfrac{SND(x) =_q SND(y) : B}{h(SND(x)) =_{\mu_h(q)} h(SND(y)) : B'}}{\langle g(FST(x), h(SND(x))\rangle =_{\epsilon_\wedge(\mu_g(p),\mu_h(q))} \langle g(FST(y)), h(SND(y))\rangle : A' \times B'}}{f(x) =_{\epsilon_\wedge(\mu_g(p),\mu_h(q))} f(y) : A' \times B'}$$

And thus:

$$\mu_f(\epsilon_\wedge(p, q)) =_{mxc} \epsilon_\wedge(\mu_g(p), \mu_h(q))$$

□

5.6 UNIT TYPE

For the unit type 1, our objective is to show the following theorem:

Theorem 5.4. *For any $x, y : 1$, there is a path t such that $x =_t y$. Moreover, $t = \rho$.*

Proof. To show that there is such t, we need to use the induction for the unit type (Univalent Foundations Program, 2013):

$$* \rhd_\eta x : 1$$

Therefore, given $x, y : 1$, we have:

$$\cfrac{x =_{\sigma(\eta)} * : 1 \qquad * =_\eta y : 1}{x =_{\tau(\sigma(\eta)),\eta)} y : 1}$$

Moreover, by rule **4**, we have:

$$\tau(\sigma(\eta), \eta) =_{tsr} \rho.$$

Thus, $t \equiv \tau(\sigma(\eta), \eta)$ and $t =_{tsr} \rho$. $\square$

5.7 FUNCTION EXTENSIONALITY

In this subsection, we are interested in the property of function extensionality. In other words, we want to conclude that given any two functions f, g, if for any x we have that $f(x) = g(x)$, then $f = g$. That $f = g$ implies $f(x) = g(x)$ by rules of basic type theory is shown in the sequel. Nonetheless, basic type theory is insufficient to derive function extensionality (Univalent Foundations Program, 2013). Our approach using computational paths also cannot derive full function extensionality. Nevertheless, we end up proving a weakened version which says that if A is non-empty, then the above principle of function extensionality over $A \to B$ holds (QUEIROZ; OLIVEIRA; RAMOS, 2016):

$$A \to (\Pi f^{A\to B} \Pi g^{A\to B} (\Pi x^A \mathtt{Id}_B(APP(f, x), APP(g, x)) \to \mathtt{Id}_{A\to B}(f, g)))$$

The proof is as follows (QUEIROZ; OLIVEIRA; RAMOS, 2016):

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}[f : A \to B] \\ [g : A \to B] \\ \Pi x^A Id_B(APP(f,x), APP(g,x))]\end{array}}{: Id_B(APP(f,z), APP(g,z))} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[f : A \to B]}{\lambda z APP(f,z) =_\eta f : A \to B}}{f =_{\sigma(\eta)} \lambda z.APP(f,z) : A \to B} \quad \dfrac{[APP(f,z) =_t APP(g,z) : B]}{\lambda z APP(f,z) =_{\xi(t)} \lambda z.APP(g,z) : A \to B}}{f =_{\tau(\sigma(\eta),\xi(t))} \lambda z.APP(g,z) : A \to B} \quad \dfrac{[g : A \to B]}{\lambda z.APP(g,z) =_\eta g : A \to B}}{f =_{\tau(\tau(\sigma(\eta),\xi(t)),\eta)} g : A \to B}}{(\tau(\tau(\sigma(\eta),\xi(t)),\eta))(f,g) : Id_{A \to B}(f,g)}}{REWR(APP(v,z), \acute{t}.(\tau(\tau(\sigma(\eta),\xi(t)),\eta))(f,g)) : Id_{A \to B}(f,g)}}{\lambda v.REWR(APP(v,z), \acute{t}.(\tau(\tau(\sigma(\eta),\xi(t)),\eta))(f,g)) : \Pi x^A.Id_B(APP(f,x), AP(g,x)) \to Id_{A \to B}(f,g)}$$

$$\dfrac{\dfrac{\lambda g.\lambda v.REWR(APP(v,z), \acute{t}.(\tau(\tau(\sigma(\eta),\xi(t)),\eta))(f,g)) : \Pi g^{A \to B}(\Pi x^A.Id_B(APP(f,x), AP(g,x)) \to Id_{A \to B}(f,g))}{g.\lambda v.REWR(AP(v,z), \acute{t}.(\tau(\tau(\sigma(\eta),\xi(t)),\eta))(f,g)) : \Pi f^{A \to B}\Pi g^{A \to B}(\Pi x^A.Id_B(APP(f,x), AP(g,x)) \to Id_{A \to B}(f,g))}}{v.REWR(APP(v,z), \acute{t}.(\tau(\tau(\sigma(\eta),\xi(t)),\eta))(f,g)) : A \to (\Pi f^{A \to B}\Pi g^{A \to B}(\Pi x^A.Id_B(APP(f,x), AP(g,x)) \to Id_{A \to B}(f,g)))}$$

Nevertheless, if we want full function extensionality and not just a weak version, we need to add a new rule to type theory. First, we let's prove the following lemma:

Lemma 5.15. *The following function exists:*

$$(f = g) \to \Pi_{(x:A)}(f(x) = g(x) : B(x))$$

Proof. The construction is straightforward:

$$\frac{\dfrac{[f =_s g] \qquad [x : A]}{f(x) =_{\nu(s)} g(x) : B(x)}}{\lambda s.\lambda x.(f(x) =_{\nu(s)} g(x)) : (f = g) \to \Pi_{(x:A)}(f(x) = g(x) : B(x))}$$

□

Now, to add function extensionality to our system, we need to add the following inference rule:

$$\frac{\lambda x.(f(x) =_t g(x)) : \Pi_{(x:A)}B}{f =_{ext(t)} g}\ ext$$

This rule is only needed if one wants to work with an extensional system. In that case, together with this inference rule, we also need to introduce two important reduction rules related to extensionality:

$$ext(\nu(s)) =_{extr} s$$

$$\nu(ext(t)) =_{extl} t$$

Since these rules are connected only to extensionality, we do not consider them as part of the basic rules of our rewriting system. Nevertheless, we can now prove the following:

Lemma 5.16. $(f = g) \simeq \Pi_{(x:A)}(f(x) = g(x) : B(x))$

Proof. This theorem is the direct application of the aforementioned extensionality rules. We have:

$$\frac{\dfrac{\dfrac{f =_s g : \Pi_{(x:A)}B \qquad x : A}{f(x) =_{\nu(s)} g(x) : B(x)}}{\lambda x.(f(x) =_{\nu(s)} g(x)) : \Pi_{(x:A)}B}}{f =_{ext(\nu(s))} g : \Pi_{(x:A)}B} \triangleright_{extr} \quad f =_s g : \Pi_{(x:A)}B$$

We also have:

$$\cfrac{\cfrac{\cfrac{\lambda x.(f(x) =_t g(x)) : \Pi_{(x:A)}B}{f =_{ext(t)} g : \Pi_{(x:A)}B} \qquad [x : A]}{f(x) =_{\nu(ext(t))} g(x) : B(x)}}{\lambda x.(f(x) =_{\nu(ext(t))} g(x)) : \Pi_{(x:A)}B} \rhd_{extl} \quad \lambda x.(f(x) =_t g(x)) : \Pi_{(x:A)}B$$

Those two derivations tree establish the equivalence. □

Before we prove the next theorem, we need to revisit transport. For any function $f : A(x) \to B(x)$, it is possible to transport along this function f, resulting in $p_*(f) : A(y) \to B(y)$. In our approach, one should think of $p_*(f)$ as a function that has transport of a term $a : A(x)$ as input, i.e., $p_*(a) : A(y)$. Thus, we define $p_*(f)$ point-wise:

$$p_*(f)(p_*(a)) \equiv p_*(f(a))$$

Lemma 5.17. *For any path $x =_p y : X$ and functions $f : A(x) \to B(x)$ and $g : A(y) \to B(y)$, we have the following equivalence:*

$$(p_*(f) = g) \simeq \Pi_{(a:A(x))}(p_*(f(a)) = g(p_*(a)))$$

Proof. We give two derivations tree, using the rules that we have established in the previous theorem:

$$\cfrac{\cfrac{\cfrac{\cfrac{p_*(f) =_p g \qquad [a : A(x)]}{p_*(f)(p_*(a)) =_{\nu(p)} g(p_*(a)) : B(y)}}{\lambda a.(p_*(f)(p_*(a) \equiv f(a)) =_{\nu(p)} g(p_*(a))) : \Pi_{(a:A(x))}(p_*(f(a)) = g(p_*(a)))}}{p_*(f) =_{ext(\nu(p))} g}}{p_*(f) =_p g} \rhd_{extl}$$

And:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\lambda a.(p_*(f(a)) =_t g(p_*(a)))}{\lambda a.(p_*(f)(p_*(a)) =_t g(p_*(a)))}}{p_*(f) =_{ext(t)} g} \qquad [a : A(x)]}{\cfrac{p_*(f)(p_*(a)) =_{\nu(ext(t))} g(p_*(a))}{p_*(f(a)) =_{\nu(ext(t))} g(p_*(a))}}}{\cfrac{p_*(f(a)) =_t g(p_*(a))}{\lambda a.(p_*(f(a)) =_t g(p_*(a)))}} \rhd_{extr}$$

□

5.8 UNIVALENCE AXIOM

The first thing to notice is that in our approach the following lemma holds:

Lemma 5.18. *For any types A and B, the following function exists:*

$$idtoeqv : (A = B) \to (A \simeq B)$$

Proof. The idea of the proof is similar to the one shown in (Univalent Foundations Program, 2013). We define *idtoeqv* to be $p_* : A \to B$. Thus, to end this proof, we just need to show that p_* is an equivalence.

For any path p, we can form a path $\sigma(p)$ and thus, we have $(\sigma(p))_* : B \to A$. Now, we show that $(\sigma(p)))_*$ is a quasi-inverse of p_*.

We need to check that:

1. $p_*((\sigma(p)_*(b)) = b$
2. $(\sigma(p))_*(p_*(a)) = a$

Both equations can be shown by an application of lemma **5.10**:

1. $p_*((\sigma(p)_*(b)) = (\sigma(p) \circ p)_*(b) = \tau(p, \sigma(p))_*(b) =_{tr} \rho_*(b) =_{\mu(p)} b$.
2. $(\sigma(p))_*(p_*(a)) = (p \circ \sigma(p))\text{ť}_*(a) = \tau(\sigma(p), p)_*(a) =_{tsr} \rho_*(a) =_{\mu(p)} a$

□

As we did in the previous section in lemma **5.15**, we showed that a function exists, but we did not show that it is an equivalence. In fact, basic type theory cannot conclude that *idtoeqv* is an equivalence(Univalent Foundations Program, 2013). If we want this equivalence to be a property of our system, we must add a new axiom. This axiom is known as Voevodsky's univalence axiom(Univalent Foundations Program, 2013):

Axiom 5.1. *For any types A, B, idtoeqv is an equivalence, i.e., we have:*

$$(A = B) \simeq (A \simeq B)$$

Lemma 5.19. *For any $x, y : A$, $u(x) : B(x)$ and path $x =_p y : A$, we have:*

$$transport^B(p, u(x)) = transport^{X \to X}(\mu_B(p), u(x)) = idtoeqv(\mu_B(p))(u(x))$$

Proof. We develop every term of the equation and show that they arrive at the same conclusion:

$$\dfrac{\dfrac{x =_p y : A \qquad u(x) : B(x)}{p(x,y) \circ u(x) : B(y)}}{p(x,y) \circ u(x) =_{\mu(p)} u(y) : B(y)}$$

$$\dfrac{\dfrac{B(x) =_{\mu_B(p)} B(y) \qquad u(x) : B(x)}{\mu_B(p)(B(x), B(y)) \circ u(x) : B(y)}}{\mu_B(p)(B(x), B(y)) \circ u(x) =_{\mu(p)} u(y) : B(y)}$$

Since $idtoeqv \equiv p_*$, we have that $idtoeqv(\mu_B(p))(u(x))$ is the same as $p_*(\mu_B(p))(u(x))$ that is the same as $transport^{X \to X}(\mu_B(p), u(x))$. □

5.9 IDENTITY TYPE

In this section, we investigate specific lemmas and theorems related to the identity type. We start with the following theorem:

Theorem 5.5. *if $f : A \to B$ is an equivalence, then for $x, y : A$ we have:*

$$\mu_f : (x = y : A) \to (f(x) = f(y) : B)$$

Proof. We will omit the specific details of this proof, since it is equal to the one of **theorem 2.11.1** presented in (Univalent Foundations Program, 2013). This is the case because this proof is independent of the usage of the induction principle of the identity type. The only difference is that at some steps we need to cancel inverse paths. In our approach, this is done by straightforward applications of **rules 3,4,5** and **6**. □

Lemma 5.20. *For any $a : A$, with $x_1 =_p x_2$*

1. $transport^{x \to (a=x)}(p, q(x_1)) = \tau(q(x_1), p),$ *for* $q(x_1) : a = x_1$
2. $transport^{x \to (x=a)}(p, q(x_1)) = \tau(\sigma(p), q(x_1),$ *for* $q(x_1) : x_1 = a$
3. $transport^{x \to (x=x)}(p, q(x_1)) = \tau(\sigma(p), \tau(q(x_1), p))$ *for* $q(x_1) : x_1 = x_1$

Proof. 1. We start establishing the following reduction:

$$\dfrac{a =_{q(x_1)} x_1 \qquad x_1 =_p x_2}{a =_{\tau(q(x_1,p))} x_2} \triangleright \quad a =_{q(x_2)} x_2$$

Thus, we just need to show that $transport^{x \to (a=x)}(p, q(x_1))$ also reduces to $a =_{q(x(2))} x_2$:

$$\frac{x_1 =_p x_2 \qquad q(x_1) : a = x_1}{p(x_1, x_2) \circ q(x_1) : a = x_2} =_{\mu(p)} \quad (a =_{q(x_2)} x_2)$$

2. We use the same idea:

$$\frac{x_2 =_{\sigma(p)} x_1 \qquad x_1 =_{q(x_1)} a}{x_2 =_{\tau(\sigma(p),q(x_1))} a} \triangleright \quad x_2 =_{q(x_2)} a$$

$$\frac{x_1 =_p x_2 \qquad q(x_1) : x_1 = a}{p(x_1, x_2) \circ q(x_1) : x_2 = a} =_{\mu(p)} \quad (x_2 =_{q(x_2)} a)$$

3. Same as the previous cases:

$$\frac{\dfrac{x_2 =_{\sigma(p)} x_1 \qquad x_1 =_{q(x_1)} x_1}{x_2 =_{\tau(\sigma(p),q(x_1))} x_1} \qquad x_1 =_p x_2}{x_2 =_{\tau(\tau(\sigma(p),q(x_1)),p)} x_2} \triangleright \quad x_2 =_{q(x_2)} x_2$$

$$\frac{x_1 =_p x_2 \qquad q(x_1) : x_1 = x_1}{p(x_1, x_2) \circ q(x_1) : x_2 = x_2} =_{\mu(p)} \quad (x_2 =_{q(x_2)} x_2)$$

□

Theorem 5.6. *For any $f, g : A \to B$, with $a =_p a' : A$ and $f(a) =_{q(a)} g(a) : B$, we have:*

$$transport^{x \mapsto (f(x) = g(x)):B}(p, q) = \tau(\tau(\sigma(\mu f(p)), q(a)), \mu_g(p)) : f(a') = g(a')$$

Proof. This proof is analogous to the proof of the previous lemma:

$$\frac{\dfrac{\dfrac{\dfrac{a =_p a' : A}{f(a) =_{\mu_f(p)} f(a')}}{f(a') =_{\sigma(\mu_f(p))} f(a)} \qquad f(a) =_{q(a)g(a)}}{f(a) =_{\tau(\sigma(\mu_f(p)),q(a))} g(a)} \qquad \dfrac{a =_p a'}{g(a) =_{\mu_g(p)} g(a')}}{f(a') =_{\tau(\tau(\sigma(\mu f(p)),q(a)),\mu_g(p))} g(a')} \triangleright \quad f(a') =_{q(a')} g(a')$$

And:

$$\frac{a =_p a' \qquad q(a) : f(a) = g(a)}{p(a,a') \circ q(a) : f(a') = g(a')} =_{\mu(p)} \quad (f(a') =_{q(a')} g(a'))$$

□

Theorem 5.7. *For any $f, g : \Pi_{(x:A)}B(x)$, with $a =_p a' : A$ and $f(a) =_{q(a)} g(a) : B(a)$, we have:*

$$transport^{x \to (f(x)=g(x):B(x))}(p, q) = \tau(\tau(\sigma(apd_f(p)), \mu_{transport^B p}(q)), apd_g(p))$$

where $apd_f(p) \equiv (p(a,a') \circ f(a) =_{\mu(p)} f(a'))$ and $apd_g \equiv (p(a,a') \circ g(a) =_{\mu(p)} g(a'))$

Proof. Similar to previous theorem:

$$\dfrac{\dfrac{\dfrac{p(a,a') \circ f(a) =_{\mu(p)} f(a')}{f(a') =_{\sigma(\mu(p))} p(a,a') \circ f(a)} \qquad \dfrac{f(a) =_{q(a)} g(a)}{p(a,a') \circ f(a) =_{\mu_{trans^B p}(q(a))} p(a,a') \circ g(a)}}{f(a') =_{\tau(\sigma(\mu(p)),\mu_{trans^B p}(q(a)))} p(a,a') \circ g(a)} \qquad p(a,a') \circ g(a) =_{\mu(p)} g(a}{f(a') =_{\tau(\tau(\sigma(\mu(p)),\mu_{trans^B p}(q(a))),\mu(p)} g(a')}$$

$$\rhd \quad f(a') =_{q(a')} g(a')$$

And:

$$\frac{a =_p a' \qquad q(a) : f(a) = g(a)}{p(a,a') \circ q(a) : f(a') = g(a')} \rhd_{\mu(p)} \quad f(a') =_{q(a')} g(a')$$

□

Theorem 5.8. *For any $a =_p a' : A$, $a =_q a$ and $a' =_r a'$, we have:*

$$(transport^{x \to (x=x)}(p, q) = r) \simeq (\tau(q, p) = \tau(p, r))$$

Proof. We use lemma **5.20** to prove this theorem, together with **rules 3,4,5,6** and **37**. We also consider functions $f(x) \equiv \tau(p, x) : (a' = z) \to (a = z)$ and $f^{-1}(x) \equiv \tau(\sigma(p), x) : (a = z) \to (a' = z)$. We proceed the same way as we have done to prove previous equivalences. In other words, we show two derivations trees. They are as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{transport^{x \to (x=x)}(p, q) = r}{\tau(\sigma(p), \tau(q, p)) = r}\text{lemma 5.20}}{\tau(p, \tau(\sigma(p), \tau(q, p))) = \tau(p, r)}\mu_f}{\tau(\tau(p, \sigma(p)), \tau(q, p)) = \tau(p, r)}}{\tau(\rho, \tau(q, p)) = \tau(p, r)}}{\tau(q, p) = \tau(q, p)}$$

And:

$$\begin{array}{cl}
\tau(q,p) = \tau(p,r) & \\ \hline
\tau(\sigma(p),\tau(q,p)) = \tau(\sigma(p),\tau(p,r)) & \mu_{f^{-1}} \\ \hline
\tau(\sigma(p),\tau(q,p)) = \tau(\tau(\sigma(p),p),r) & \\ \hline
\tau(\sigma(p),\tau(q,p)) = \tau(\tau(\rho,r) & \\ \hline
\tau(\sigma(p),\tau(q,p)) = r & \\ \hline
\mathit{transport}^{x\to(x=x)}(p,q) = r & \text{lemma 5.20}
\end{array}$$

□

5.10 COPRODUCT

One essential thing to remember is that a product $A+B$ has a left injection $\mathit{inl} : A \to A+B$ and $\mathit{inr} : B \to A+B$. As described in (Univalent Foundations Program, 2013), it is expected that $A+B$ contains copies of A and B disjointly. In our path based approach, we achieve this by constructing every path $\mathit{inl}(a) = \mathit{inl}(b)$ and $\mathit{inr}(a) = \mathit{inr}(b)$ by applications of axiom μ on paths $a = b$. Thus we show that we get the following equivalences:

1. $(\mathit{inl}(a_1) = \mathit{inl}(a_2)) \simeq (a_1 = a_2)$
2. $(\mathit{inr}(b_1) = \mathit{inr}(b_2)) \simeq (b_1 = b_2)$
3. $(\mathit{inl}(a) = \mathit{inr}(b)) \simeq 0$

To prove this, we use the same idea as in (Univalent Foundations Program, 2013). We characterize the type:

$$(x \to (\mathit{inl}(a_0) = x)) : \Pi_{(x:A+B)}(\mathit{inl}(a_0 = x))$$

To do this, we define a type *code*:

$$x : A + B \vdash \mathit{code}(x) \text{ type}$$

Our main objective is to prove the equivalence $\Pi_{(x:A+B)}((\mathit{inl}(a_0) = x) \simeq \mathit{code}(x))$. Using the recursion principle of the coproduct, we can define *code* by two equations:

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

Theorem 5.9. *For any $x : A + B$, we have $\mathit{inl}(a_0 = x) \simeq \mathit{code}(x)$*

Proof. To show this equivalence, we use the same method as the one shown in (Univalent Foundations Program, 2013). The main idea is to define functions

$$encode : \Pi_{(x:A+B)}\Pi_{(p:inl(a_0)=x)}code(x)$$
$$decode : \Pi_{(x:A+B)}\Pi_{(c:code(x))}(inl(a_0) = x))$$

such that *decode* acts as a quasi-inverse of *encode*.

We start defining *encode*:

$$encode(x, s) \equiv transport^{code}(s, \rho_{a_0})$$

We notice that $\rho_{a_0} : code(inl(a_0))$, since $code(inl(a_0)) \equiv (a_0 =_\rho a_0)$ We also notice that for *encode*, it is only possible for the argument x to be of the form $x \equiv inl(a)$, since the other possibility is $x \equiv inr(a)$, but that case is not possible, because we would have a function to $code(inr(b)) \equiv 0$.

For *decode*, when $x \equiv inl(a)$, we have that $code(x) \equiv a_0 =_c a$ and thus, we define decode as $(inl(a_0) =_{\mu(c)} inl(a))$. When $x \equiv inr(a)$, then $code(x) \equiv 0$ and thus, we define *decode* as having any value, given by the elimination of the type 0. Now, we can finally prove the equivalence.

Starting with *encode*, we have $x \equiv inl(a)$, $inl(a_0) =_s x$. Since
$encode(x, s) \equiv transport^{code}(s, \rho_{a_0})$, we have:

$$\cfrac{\cfrac{inl(a_0) =_s inl(a) \qquad \rho_{a_0} : code(inl(a_0))}{s(inl(a_0), inl(a)) \circ \rho_{a_0} : code(inl(a))}}{\rho_a : code(inl(a)) \equiv code(x)} =_{\mu(s)}$$

Now, we can go back to $inl(a_0) = inl(a)$ by an application of *decode*, since:

$$decode(\rho_a : code(x)) \equiv inl(a_0) =_{\mu_{inl}} inl(a)$$

And we conclude this part, since in our approach $inl(a_0) =_s inl(a)$ is constructed by applications of axiom μ.

Now, we start from decode. Let $c : code(x)$. If $x \equiv inl(a)$, then $c : a_0 = a$ and thus, $decode(c) \equiv inl(a_0) =_{\mu(c)} inl(a)$. Now, we apply *encode*. We have:

$$\begin{aligned} encode(x, \mu_c) &= transport^{code}(\mu_c, \rho_{a_0}) \\ &= transport^{a \to (a_0 = a)}(c, \rho_{a_0}) \quad && (\textbf{Lemma 11}) \\ &= \tau(\rho_{a_0}, c) && (\textbf{Lemma 20}) \\ &= c && (\textbf{Rule 6}) \end{aligned}$$

If $x \equiv inr(b)$, we have that $c : 0$ and thus, as stated in (Univalent Foundations Program, 2013), we can conclude anything we wish.

□

5.11 REFLEXIVITY

In this section, our objective is to conclude an important result related to the reflexive path ρ:

Theorem 5.10. *For any type A and a path $x =_\rho x : A$, if a path s is obtained by a series (perhaps empty) of applications of axioms and rules of inference of $\lambda\beta\eta$-equality theory for type theory to the path ρ, then there is a path t' such that $s =_{t'} \rho$.*

Proof. • Base Case:

We can start only with a path $x =_\rho$. In that case, it is easy, since we have $\rho =_\rho \rho$.

Now, we consider the inductive steps. Starting from a path s and applying τ, σ, we already have rules yield the desired path:

- $s = \sigma(s')$, with $s' =_{t'} \rho$.

  In this case, we have $s = \sigma(s') = \sigma(\rho) =_{sr} \rho$.

- $s = \tau(s', s'')$, with $s' =_{t'} \rho$ and $s'' =_{t''} \rho$.

  We have that $s = \tau(s', s'') = \tau(\rho, \rho) =_{trr} \rho$

  The cases for applications of μ, ν and ξ remain to be proved. We introduce three new rules that handle these cases.

- $s = \mu(s')$, with $s' =_{t'} \rho$.

  We introduce rule **45**:

$$\frac{x =_{\rho_x} x : A \qquad [f : A \to B]}{f(x) =_{\mu(\rho_x)} f(x) : B(x)} \triangleright_{mxp} \quad f(x) =_{\rho_{f(x)}} f(x) : B(x)$$

  This rule is also valid for the dependent case:

$$\frac{x =_{\rho_x} x : A \qquad [f : \Pi_{(x:A)} B(x)]}{p(x,x) \circ f(x) =_{\mu(\rho_x)} f(x) : B(x)} \triangleright_{mxp} \quad f(x) =_{\rho_{f(x)}} f(x) : B(x)$$

  Thus, we have $s = \mu(s') = \mu(\rho) =_{mxp} \rho$.

- $s = \nu(s')$, with $s' =_{t'} \rho$.

  We introduce rule **46**:

$$\frac{f =_\rho f : \Pi_{(x:A)} B(x)}{f(x) =_{\nu(\rho_x)} f(x) : B(x)} \triangleright_{nxp} \quad f(x) =_{\rho_{f(x)}} f(x)$$

Thus, $s = \nu(s') = \nu(\rho) =_{nxp} \rho$.

- $s = \xi(s')$, with $s' =_{t'} \rho$.

  We introduce rule **47**:

$$\frac{b(x) =_{\rho} b(x) : B \qquad x : A}{\lambda x.b(x) =_{\xi(\rho)} \lambda x.b(x) : A \to B} \rhd_{xxp} \quad \lambda x.b(x) =_{\rho} \lambda x.b(x)$$

  Thus, $s = \xi(s') = \xi(\rho) =_{xxp} \rho$.

  If we consider function extensionality, this theorem still holds:

- $s = ext(s')$, with $s' =_{t'} \rho$.

  We introduce a new rule to handle this case. Since it is related only to extensionality (i.e., when one admits the inference rule *ext* to the system), we do not add this to the basic rules of our system.

$$\frac{\lambda x.(f(x) =_{\rho} f(x)) : \Pi_{(x:A)} B(x)}{f =_{ext(\rho)} f} \rhd exp \quad f =_{\rho} f$$

  Thus, $s = ext(s') = ext(\rho) =_{exp} \rho$.

□

5.12 NATURAL NUMBERS

The Natural Numbers is a type defined inductively by an element $0 : \mathbb{N}$ and a function $succ : \mathbb{N} \to \mathbb{N}$. In our approach, the path space of the naturals is also characterized inductively. We start from the reflexive path $0 =_{\rho} 0$. All subsequent paths are constructed by applications of the inference rules of $\lambda\beta\eta$-equality. We show that this characterization is similar to the one constructed in (Univalent Foundations Program, 2013). To do this, we use *code*, *encode* and *decode*. For $\mathbb{N}$, we define *code* recursively (Univalent Foundations Program, 2013):

$$\begin{gathered} code(0,0) \equiv 1 \\ code(succ(m),0) \equiv 0 \\ code(0,succ(m)) \equiv 0 \\ code(succ(m),succ(n)) \equiv code(m,n) \end{gathered}$$

We also define a dependent function $r : \Pi_{(n:\mathbb{N})} code(m,n)$, with:

$$r(0) \equiv *$$
$$r(succ(n)) \equiv r(n)$$

Theorem 5.11. *For any $m, n : \mathbb{N}$, if there is a path $m =_t n : \mathbb{N}$, then $t \rhd \rho$.*

Proof. Since all paths are constructed from the reflexive path $0 =_\rho 0$, this is a direct application of theorem **5.10**. □

Theorem 5.12. *For any $m, n : \mathbb{N}$, we have $(m = n) \simeq code(m, n)$*

Proof. We need to define *encode* and *decode* and prove that they are quasi-inverses. We define $encode : \Pi_{(m,n:\mathbb{N})}(m = n) \to code(m, n)$ as:

$$encode(m, n, p) \equiv transport^{code(m,-)}(p, r(m))$$

We define $decode : \Pi_{(m,n:\mathbb{N})}code(m, n) \to (m = n)$ recursively:

$$decode(0, 0, c) \equiv 0 =_\rho 0$$
$$decode(succ(m), 0, c) \equiv 0$$
$$decode(0, succ(m), c) \equiv 0)$$
$$decode(succ(m), succ(n), c) \equiv \mu_{succ}(decode(m, n, c))$$

We now prove that if $m =_p n$, then $decode(code(m, n)) = \rho$. We prove by induction. The base is trivial, since $decode(0, 0, c) \equiv \rho$. Now, consider $decode(succ(m), succ(n), c)$. We have that $decode(succ(m), succ(n), c) \equiv \mu_{succ}(decode(m, n, c))$. By the inductive hypothesis, $decode(m, n, c) \equiv \rho$. Thus, we need to prove that $\mu_{succ} = \rho$. This last step is a straightforward application of **rule 47**. Therefore, $\mu_{succ} =_{mxp} \rho$. With this information, we can start the proof of the equivalence.

For any $m =_p n$, we have:

$$encode(m, n, p) \equiv transport^{code(m,-)}(p, r(m))$$

Thus:

$$\frac{m =_p n \qquad r(m) : code(m, m)}{p(m, n) \circ r(m) : code(m, n)} =_{\mu(p)} \quad (r(n) : code(m, n))$$

Now, we know that $decode(r(n) : code(m, n)) = \rho$ and,by theorem **5.11**, $p = \rho$.

The proof starting from a $c : code(m, n)$ is equal to the one presented in (Univalent Foundations Program, 2013). We prove by induction. If m and n are 0, we have the trivial path $0 =_\rho 0$, thus $decode(0, 0, c) = \rho_0$, whereas $encode(0, 0, \rho_0) \equiv r(0) \equiv *$. We conclude this part recalling that every $x : 1$ is equal to $*$, since we have $x =_{\sigma(\eta)} * : 1$. In the case of $decode(succ(m), 0, c)$ or $decode(0, succ(n), c)$, $c : 0$. The only case left is for $decode(succ(m), succ(n), c)$. Similar to (Univalent Foundations Program, 2013), we prove by induction:

$$\begin{aligned}
&encode(succ(m), succ(n), decode(succ(m), succ(n), c)) \\
&\qquad = encode(succ(m), succ(n), \mu_{succ}(decode(m, n, c)) \\
&\qquad = transport^{code(succ(m),-)}(\mu_{succ}(decode(m, n, c)), r(succ(m)) \\
&\qquad = transport^{code(succ(m),succ(-)}(decode(m, n, c), r(succ(m))) \\
&\qquad = transport^{code(m,-)}(decode(m, n, c), r(m)) \\
&\qquad = encode(m, n, decode(m, n, c)) \\
&\qquad = c
\end{aligned}$$

□

5.13 SETS AND AXIOM K

In this subsection, our objective is to prove, using our computational path approach, important results related to sets. First, We define the concept of set as done traditionally in Homotopy Type Theory. Then, we show that the connection between the axiom K and sets is also valid in our approach. We use these results to show that the naturals numbers are a set. We also prove Hedberg's theorem is valid in our theory, since only a few steps of its proof differs from method developed in (Univalent Foundations Program, 2013).

We start with the definition of set (Univalent Foundations Program, 2013):

Definition 5.4. *A type A is a set if for all $x, y : A$ and all $p, q : x = y$, we have $p = q$.*

In type theory, if a type is a set, we also say that it has the uniqueness of identity proof (UIP) property, since all proofs of the equality of two terms $x = y$ are equal.

We now introduce the following axiom, known as **axiom K**(HOFMANN; STREICHER, 1994):

For all $x : X$ and $p : (x =_X x)$, we have $p = refl_x$.

Of course, this previous formulation is one familiar to classic type theory. In our approach, axiom K can be understood as the following formulation:

Axiom 5.2. *For all $x : X$ and $x =_t x : X$, we have $t = \rho_x$.*

Our objective is to establish a connection between sets and axiom K. The following lemma has proved to be useful:

Lemma 5.21. *For every path t, there is a path t^{-1} such that $t \circ t^{-1} = \rho$ and $t^{-1} \circ t = \rho$. Furthermore, t^{-1} is unique up to propositional identity.*

Proof. We claim that $t^{-1} = \sigma(t)$. The identities are straightforward. First, we have that $t \circ t^{-1} \equiv \tau(\sigma(t), t) =_{tsr} \rho$. We also have $t^{-1} \circ t \equiv \tau(t, \sigma(t)) =_{tr} \rho$. Now, suppose we have s such that $\tau(t, s) =_{s'} \rho$. Thus, we have that $\tau(t, \sigma(t)) =_{\tau(tr, \sigma(s'))} \tau(t, s)$ and thus, $\sigma(t) = s$. □

Theorem 5.13. *A type X is a set iff it satisfies axiom K.*

Proof. First, If X is a set, we want to show that it satisfies axiom K. Suppose we have a path $x =_t x$. From the axioms of $\lambda\beta\eta$-equality, we also have that $x =_\rho x$. Since X is a path, it is always the case that $t = \rho$ and thus, X satisfies axiom K.

If X satisfies axiom K, we want to show that X is a set. To do this, we want to show that given paths $p, q : x = y$, then we have $p = q$. We show this in the following manner. From a path $x =_q y$, we apply σ to obtain the inverse path $y =_{\sigma(q)x} x$. Then, we can concatenate p and $\sigma(q)$, obtaining a path $x =_{\tau(p,\sigma(q))} x$. By axiom K, we have $\tau(p, \sigma(q)) = \rho$. Analogously, $\tau(\sigma(q), p) = \rho$. Thus, by the previous lemma, $\sigma(q) = p^{-1} = \sigma(p)$ and thus, $q = p$. Thus, by an application of σ, $p = q$. □

Theorem 5.14. $\mathbb{N}$ *is a set.*

Proof. That $\mathbb{N}$ satisfies axiom K is a direct consequence of theorem **5.11**. Thus, from theorem **5.13**, we conclude that $\mathbb{N}$ is a set. □

In classic homotopy type theory, one can achieve the previous result by following a different path. Firstly, one should be aware of the following concept (Univalent Foundations Program, 2013):

Definition 5.5. *A type X has **decidable equality** if for all $x, y : X$, the following type is inhabited:*

$$(x = y : X) + \neg(x = y : X)$$

Theorem 5.15. *If X has decidable equality, then X is a set.*

This theorem is known as Hedberg's theorem . We will not show a full proof of it, since one can follow exactly the steps established in (Univalent Foundations Program, 2013). One should only be careful to notice that the path *apd* of classic homotopy type theory is just our application of axiom μ on a dependent function f and that **lemma 2.9.6** of (Univalent Foundations Program, 2013) has already been proved in this work, in the form of lemma **5.17**.

We can also use Hedberg's theorem to give an alternative proof of theorem **5.14**, one similar to the one given in classic type theory:

Theorem 5.16. $\mathbb{N}$ *has decidable equality and thus, is a set.*

Proof. For any $x, y : \mathbb{N}$, we want to show that $(x = y) + \neg(x = y)$ is inhabited. We proceed by induction in x. For the base case, we have $x = 0$. If $y = 0$, then we have $0 =_\rho 0$. If $y = succ(n)$, we use the *encode* type for $\mathbb{N}$. We have that $encode(0, succ(n)) : (0 = succ(n)) \to \mathbf{0}$ and thus, $encode(0, succ(n)) : \neg(0 = succ(n))$.

For the inductive step, we consider $x = succ(m)$. If $y = 0$, then we can use *encode* again to obtain $\neg(succ(m) = 0)$. If $y = succ(n)$, by the inductive hypothesis, we have two more cases to consider. If $m = n$, then we apply axiom μ, and thus $succ(m) =_{\mu_{succ}} succ(n)$. If $\neg(m = n)$, then we just need to show that $succ$ is injective to obtain $\neg(succ(m) = succ(n))$. But that $succ$ is injective is a direct consequence of applying encode and then decode to $succ(m) = succ(n)$, since from that we conclude $m = n$. Therefore, from $\neg(succ(m) = succ(n))$ we conclude $\neg(m = n)$. $\square$

5.14 FUNDAMENTAL GROUP OF A CIRCLE

The objective of this section is to show that it is possible to use computational paths to obtain one of the main results of homotopy theory, the fact that the fundamental group of a circle is isomorphic to the integers group. First, we define a circle as follows:

Definition 5.6 (The circle S^1)**.** *A circle is the type generated by:*

- *A point base* : S^1
- *A computational path base* $=_{loop}$ *base* : S^1.

The first thing one should notice is that this definition doest not use only the points of the type S^1, but also a computational path *loop* between those points. That is way it is called a higher inductive type (Univalent Foundations Program, 2013). Our approach differs from the classic one on the fact that we do not need to simulate the path-space between those points, since computational paths exist in the syntax of the theory. Thus, if one starts with a path $base =_{loop} base : S^1.$, one can naturally obtain additional paths applying the path-axioms ρ, τ and σ. Thus, one has a path $\sigma(loop) = loop^{-1}$, $\tau(loop, loop)$, etc. In classic type theory, the existence of those additional paths comes from establishing that the paths should be freely generated by the constructors (Univalent Foundations Program, 2013). In our approach, we do not have to appeal for this kind of argument, since all paths comes naturally from direct applications of the axioms.

With that in mind, one can define the fundamental group of a circle. In homotopy theory, the fundamental group is the one formed by all equivalence classes up to homotopy of paths (loops) starting from a point a and also ending at a. Since the we use computational paths as the syntax counterpart in type theory of homotopic paths, we use it to propose the following definition:

Definition 5.7 ($\Pi_1(A, a)$ structure)**.** $\Pi_1(A, a)$ *is a structure defined as follows:*

$$\Pi_1(A, a) = \{[path]_{rw} \mid a =_{path} a : A\}$$

We use this structure to define the fundamental group of a circle. We also need to show that it is indeed a group.

Proposition 5.1. $(\Pi_1(S, a), \circ)$ *is a group.*

Proof. The first thing to define is the group operation $\circ$. Given any $a =_r a : S^1$ and $a =_t a : S^1$, we define $r \circ s$ as $\tau(s, r)$. Thus, we now need to check the group conditions:

- Closure: Given $a =_r a : S^1$ and $a =_t a : S^1$, $r \circ s$ must be a member of the group. Indeed, $r \circ s = \tau(s, r)$ is a computational path $a =_{\tau(s,r)} a : S^1$.

- Inverse: Every member of the group must have an inverse. Indeed, if we have a path r, we can apply $\sigma(r)$. We claim that $\sigma(r)$ is the inverse of r, since we have:

$$\sigma(r) \circ r = \tau(r, \sigma(r)) =_{tr} \rho$$
$$r \circ \sigma(r) = \tau(\sigma(r), r) =_{tsr} \rho$$

  Since we are working up to rw-equality, the equalities hold strictly.

- Identity: We use the path $a =_\rho a : S^1$ as the identity. Indeed, we have:

$$r \circ \rho = \tau(\rho, r) =_{tlr} r$$
$$\rho \circ r = \tau(r, \rho) =_{trr} r.$$

- Associativity: Given any members of the group $a =_r a : S^1$, $a =_t a$ and $a =_s a$, we want that $r \circ (s \circ t) = (r \circ s) \circ t$:

$$r \circ (s \circ t) = \tau(\tau(t, s), r) =_{tt} \tau(t, \tau(s, r)) = (r \circ s) \circ t$$

All conditions have been satisfied. $(\Pi_1(S, a), \circ)$ is a group. □

Thus, $(\Pi_1(S, a), \circ)$ is indeed a group. We call this group the fundamental group of S^1. Therefore, the objective of this section is to show that $\Pi_1(S, a) \cong \mathbb{Z}$.

Before we start developing this proof, the following lemma will prove to be useful:

Lemma 5.22. *All paths generated by a path $a =_{loop} a : S$ are rw-equal to a path $loop^n$, for a $n \in \mathbb{Z}$.*

We have said that from a *loop*, one freely generate different paths applying the composition τ and the symmetry. Thus, one can, for example, obtain something such as $loop \circ loop \circ loop^{-1} \circ loop$.... Our objective with this lemma is to show that, in fact, this path can be reduced to a path of the form $loop^n$, for $n \in \mathbb{Z}$.

Proof. The idea is to proceed by induction on the number n of loops, i.e., $loop^n$. We start from a base ρ. For the base case, it is trivially true, since we define it to be equal to $loop^0$. From ρ, one can construct more complex paths by composing with *loop* or $\sigma(loop)$ on each step. We have the following induction steps:

- A path of the form ρ concatenated with *loop*: We have $\rho \circ loop = \tau(loop, \rho) =_{trr} loop = loop^1$;

- A path of the form ρ concatenated with $\sigma(loop)$: We have $\rho\circ\sigma(loop) = \tau(\sigma(loop), \rho) =_{trr}= \sigma(loop) = loop^{-1}$

- A path of the form $loop^n$ concatenated with *loop*: We have $loop^n \circ loop = loop^{n+1}$.

- A path of the form $loop^n$ concatenated with $\sigma(loop)$: We have $loop^n \circ \sigma(loop) = (loop^{n-1}\circ loop)\circ\sigma(loop) =_{tt} loop^{n-1}\circ(loop\circ\sigma(loop)) = loop^{n-1}\circ(\tau(\sigma(loop), loop)) =_{tsr}= loop^{n-1} \circ \rho = \tau(\rho, loop^{n-1}) =_{tlr} loop^{n-1}$

- A path of the form $loop^{-n}$ concatenated with *loop*: We have $loop^{-n} = loop^{-(n-1)} \circ loop^{-1} = loop^{-(n-1)}\circ\sigma(loop)$. Thus, we have $(loop^{-(n-1)}\circ\sigma(loop))\circ loop =_{tt} loop^{-(n-1)}\circ (\sigma(loop)\circ loop) = loop^{-(n-1)}\circ\tau(loop, \sigma(loop)) =_{tr} = loop^{-(n-1)}\circ\rho = \tau(\rho, loop^{-(n-1)}) =_{tlr} loop^{-(n-1)}$.

- a path of the form $loop^{-n}$ concatenated with $\sigma(loop)$: We have $loop^{-n} \circ loop^{-1} = loop^{-(n+1)}$

Thus, every path is of the form $loop^n$, with $n \in \mathbb{Z}$. □

This lemma shows that every path of the fundamental group can be represented by a path of the form $loop^n$, with $n \in \mathbb{Z}$.

Theorem 5.17. $\Pi_1(S, a) \cong \mathbb{Z}$

To prove this theorem, one could use the approach proposed in (Univalent Foundations Program, 2013), defining an encode and decode functions. Nevertheless, since our computational paths are part of the syntax, one does not need to rely on this kind of approach to simulate a path-space, we can work directly with the concept of path.

Proof. The proof is done by establishing a function from $\Pi_1(S, a)$ to $\mathbb{Z}$ and then an inverse from $\mathbb{Z}$ to $\Pi_1(S, a)$. Since we have access to the previous lemma, this task is not too difficult. The main idea is that the n on $loop^n$ means the amount of times one goes around the circle, while the sign gives the direction (clockwise or anti-clockwise). In other words, it is the *winding* number. Since we have shown that every path of the fundamental group is of the form $loop^n$, with $n \in \mathbb{Z}$, then we just need to translate $loop^n$ to an integer n and an integer n to a path $loop^n$. We define two functions, $toInteger : \Pi_1(S, a) \to \mathbb{Z}$ and $toPath : \mathbb{Z} \to \Pi_1(S, a)$:

- *toInteger*: To define this function, we use the help of two functions defined in $\mathbb{Z}$: the successor function *succ* and the predecessor function *pred*. We define *toInteger* as follows. Of course, we use directly the fact that every loop of S is of the form $loop^n$ with $n \in \mathbb{Z}$:

$$toInteger : \begin{cases} toInteger([loop^n]_{rw} \equiv [\rho]_{rw}) = 0 & n = 0 \\ toInteger([loop^n]_{rw}) = succ(toInteger([loop^{n-1}]_{rw})) & n > 0 \\ toInteger([loop^n]_{rw}) = pred(toInteger([loop^{n+1})]_{rw}) & n < 0 \end{cases}$$

- *toPath*: We just need to transform an integer n into a path $loop^n$:

$$toPath : \begin{cases} toPath(n) = [\rho]_{rw} & n = 0 \\ toPath(n) = toPath(n-1) \circ [loop]_{rw} & n > 0 \\ toPath(n) = toPath(n+1) \circ [\sigma(loop)]_{rw} & n < 0 \end{cases}$$

Now we just need to show that they are inverses. To do this, we have to check two equations:

1. $toPath(toInteger([x]_{rw})) = [x]_{rw}$
2. $toInteger(toPath(n)) = n$

From a path $[x]_{rw}$, we apply lemma 5.22 to obtain $[x]_{rw} = [loop^n]_{rw}$ for some integer n. Thus, we have $toPath(toInteger([x]_{rw})) = toPath(toInteger([loop^n]_{rw})) = toPath(n) = [loop^n]_{rw} = [x]_{rw}$. The opposite direction is straightforward: $toInteger(toPath(n)) = toInteger([loop^n]_{rw}) = n$. Thus, we conclude that they are inverses. Also, the map between the operations of the groups is direct, since we can easily map $\circ$ to $+$. This is due to the fact that $loop^n \circ loop^m = loop^{n+m}$ and thus, $toInteger[[loop^{n+m}]_{rw} = n + m$. It is also straightforward that $toPath(n+m) = [loop^{n+m}]_{rw}$. Thus, we establish the isomorphism $\Pi_1(S, a) \cong \mathbb{Z}$. □

5.15 RULES ADDED TO $LND_{EQ} - TRS$

In this chapter, we have introduced 7 new rules to the $LND_{EQ} - TRS$ system. It is the following list of rules:

40. $\tau(\mu(r), \mu(s)) =_{tf} \mu(\tau(r, s))$
41. $\mu_g(\mu_f(p)) =_{cf} \mu_{g \circ f}(p)$
42. $\mu_{Id_A}(p) =_{ci} p$
43. $\tau(H_{f,g}(x), \mu_g(p)) =_{hp} \tau(\mu_f(p), H_{f,g}(y))$
44. $\mu_f(\epsilon_\wedge(p, q)) =_{mxc} \epsilon_\wedge(\mu_g(p), \mu_h(q))$

45. $\mu_f(\rho_x) =_{mxp} \rho_{f(x)}$
46. $\nu(\rho_x) =_{nxp} \rho_{f(x)}$
47. $\xi(\rho) =_{xxp} \rho$

Moreover, if one adds extensionality to the theory, one winds up with three additional rules:

$\nu(ext(t)) =_{extl} t$
$\mu_f(\rho_x) =_{mxp} \rho_{f(x)}$
$ext(\rho) =_{exp} \rho.$

Thus, we end this work with a total of 47 rewrites rules + 3 optional ones. Of course, to be formal we need to show that the system will terminate and is still confluent. Indeed, the confluence process can even yield new rules to solve some critical pair divergence. We leave this verification out of the scope of this work, but given the importance of these proofs, we plan to finish them in a future work.

5.16 CONCLUSION

In this chapter we have developed one of the main objectives of this work. We connected our computational path approach to homotopy type theory. Using the algebra of computational paths, we have established important results of Homotopy Type Theory. That way, we have shown that our approach yields the main building blocks of Homotopy Type Theory, on par with the classic approach. We have also improved the rewrite system, adding new reduction rules. Indeed, we have ended this chapter with one of the most classic results of algebraic topology, the fact that the fundamental group of the circle is isomorphic to the group of the integers.

In view of all results achieved in this chapter, we have developed a valid alternative approach to the identity type and homotopy type theory, based on this algebra of paths. We also believe that we have opened the way, in future works, for possible expansions of this results, formulating and proving even more intricate concepts and theorems of homotopy type theory using computational paths. In fact, based on the concepts developed in this chapter, we have constructed the fundamental group of many structures, including the Möbius band, the cylinder, torus and the projective plane. These results appear in a still unpublished work "On the Calculation of Fundamental Groups in Homotopy Type Theory by Means of Computational Paths". A preprint version is available in (VERAS et al., 2018).

6 CONCLUSION

Motivated by seeing the equality between two computational objects as a sequence of rewrites between then, we have proposed in this work a entity known as computational paths. We have also accomplished 3 main objectives. The first one was the proposal of computational paths as an new entity of type theory. In this proposal, we pointed out the fact that computational paths should be seen as the syntax counterpart of the homotopical paths between terms of a type. We have also proposed a formalization of the identity type using computational paths. The second objective was the proposal of a mathematical structure for a type using computational paths. We have shown that using categorical semantics it was possible to induce a groupoid structure for a type and also a higher groupoid structure, using computational paths and a rewrite system. We have used this groupoid structure to show that computational paths also refutes the uniqueness of identity proofs. The last objective was to formulate and prove the main concepts and building blocks of homotopy type theory. We ended this last objective with a proof of the isomorphism between the fundamental group of the circle and the group of the integers.

The introduction of this work was focused on giving a brief introduction to the foundations of mathematics and a brief explanation why axiomatic set theory, ZFC, is not suitable of working as a foundation of computation. Thus, we have said that homotopy type theory is a suitable alternative, since it can be used as a foundation of mathematics and computation at the same time. We have also pointed out that the identity type is the main concept of homotopy type theory, since it is responsible for the homotopical interpretation of a type. Nevertheless, we said that this interpretation is limited to a semantical interpretation, since there is no entity in the syntax of type theory that represents those paths. Thus, we said that the focus of this work is to add such entity to type theory, which is known as computational path.

The second chapter has been focused on introducing the basic concepts of type theory. In this chapter we have introduced the constructions of the main types, showing the formation, introduction, elimination and computation rules. We have also shown the current approach for the identity type using the constructor J. We have also shown the use of those rules in practice, showing how one can use J to construct basic types of type theory, such as the symmetry and transitivity properties of the identity type.

The third chapter focused on introducing the basic concepts of category theory. We have said that one of the objectives achieved here was the proposal of a mathematical structure for computational paths. To achieve that, we have used the framework of category theory. Thus, the needed concepts of this theory has been introduced in chapter 3. We have also shown some concepts of higher category theory, since we use then in chapter 4 to construct a higher groupoid.

The fourth chapter has been focused on introducing a new entity known as computational paths to the syntax of type theory. To achieve that, we have used concepts developed in chapter 2 and some basic concepts of λ-calculus. After introducing the concept of computational paths, we have shown how one can formally define the identity type using this entity. To do that, we show the formation, introduction, elimination and computation rules for the identity type. To make our approach clearer, we have used this newly introduced rules to construct three basic types of equality, the reflexivity, symmetry and transitivity. We also showed that the process of obtaining those constructions was easier than using J. After that, we have shown the existence of a rewrite system that establishes equalities between two computational paths. We have also said that this system does that by mapping all possible redundancies that can appear in a computational path. We also pointed out that this system is confluent and terminates. Thus, we have said that it is possible to think of a strong normal form for a computational path. In the sequel, we have used the concepts of chapter 3 to construct a groupoid model for those computational paths. We also have gone one step further, showing that it is possible to think of a higher groupoid structure. Moreover, we have put this groupoid structure together with the existence of a strong normal form to show that this approach also refutes the uniqueness of identity proofs.

The fifth chapter has been focused on accomplishing the third objective of this work. In this chapter we have developed the main building blocks of homotopy theory using computational paths. Instead of using path-induction in our proofs, we have used our algebra of paths based on our rewrite system. Proceeding that way, we have shown dozens of lemmas and theorems of homotopy type theory. Thus, we have shown that computational paths are capable of developing this theory. We finished this chapter showing one of the most classic proofs of algebraic topology: the fundamental group of the circle is isomorphic to the group of integers.

6.1 FUTURE WORK

Since we have successfully accomplished our main objectives, we hope that we have made a strong case for the power of computational paths in homotopy type theory. This approach seems promising, since it has been possible to prove many lemmas and theorems of homotopy type theory. Also, another promising aspect is that our mathematical structure mirrors the one obtained using the traditional identity type.

Thus, with those results in mind, it is possible to obtain further results in two fronts: we can focus on the mathematical interpretation of computational paths. In this work, we have limited our scope to bicategories, but it is possible, in the future, to go even further, adding more dimensions, eventually obtaining an important result: computational paths should be able of inducing a weak ∞-groupoid. We expect this fact, since this has been recently achieved using the traditional approach of the identity type (LUMSDAINE, 2009;

BERG; GARNER, 2011). The other direction is to continue to develop the main concepts and theorems of homotopy theory. We have shown dozens in chapter 5, but there are many more results. Thus, we could focus on this and use our algebra of computational paths to obtain further results. In fact, based on the concepts developed in chapter 5, we have constructed the fundamental group of many structures, including the Möbius band, the cylinder, torus and the projective plane. These results appear in a still unpublished work "On the Calculation of Fundamental Groups in Homotopy Type Theory by Means of Computational Paths". A preprint version is available in (VERAS et al., 2018).

REFERENCES

ALAMA, J. The lambda calculus. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Spring 2015. [S.l.: s.n.], 2015.

AVIGAD, J. Philosophy of mathematics. In: BOUNDAS, C. (Ed.). *The Edinburgh Companion to Twentieth-Century Philosophies*. [S.l.]: Edinburgh University Press, 2007. p. 234–251.

AWODEY, S. *Category theory*. 2nd. ed. [S.l.]: Oxford University Press, 2010.

AWODEY, S. *Category Theory Foundations*. 2012. Category Theory Foundations, Lecture at Oregon Programming Languages Summer School, Eugene, Oregon.

BARKER-PLUMMER, D. Turing machines. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Summer 2013. [S.l.: s.n.], 2013.

BERG, B. van den; GARNER, R. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society*, Oxford University Press, v. 102, n. 2, p. 370–394, 2011.

BRIDGES, D.; PALMGREN, E. Constructive mathematics. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Winter 2013. [S.l.: s.n.], 2013.

CHENADEC, P. L. On the logic of unification. *Journal of Symbolic computation*, Elsevier, v. 8, n. 1, p. 141–199, 1989.

DERSHOWITZ, N. Orderings for term-rewriting systems. *Theoretical computer science*, Elsevier, v. 17, n. 3, p. 279–301, 1982.

DUMMETT, M. *Elements of Intuitionism*. [S.l.]: Oxford, 1977. (Oxford Logic Guides, v. 39). ISBN 978-0198505242.

HARPER, R. *Type Theory Foundations*. 2012. Type Theory Foundations, Lecture at Oregon Programming Languages Summer School, Eugene, Oregon.

HINDLEY, J. R.; SELDIN, J. P. *Lambda-calculus and combinators: an introduction*. [S.l.]: Cambridge University Press, 2008.

HOFMANN, M.; STREICHER, T. The groupoid model refutes uniqueness of identity proofs. In: *Logic in Computer Science, 1994. LICS'94. Proceedings., Symposium on*. [S.l.]: IEEE, 1994. p. 208–212.

HOFMANN, M.; STREICHER, T. The groupoid interpretation of type theory. In: *Twenty-five years of constructive type theory (Venice, 1995)*. New York: Oxford Univ. Press, 1998, (Oxford Logic Guides, v. 36). p. 83–111.

HORSTEN, L. Philosophy of mathematics. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Spring 2015. [S.l.: s.n.], 2015.

HOWARD, W. A. The formulas-as-types notion of construction. In: SELDIN, J. P.; HINDLEY, J. R. (Ed.). *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. [S.l.]: Academic Press, 1980. p. 479–490. Reprint of 1969 article.

HRBACEK, K.; JECH, T. *Introduction to Set Theory, Revised and Expanded.* [S.l.]: Chapman & Hall/CRC, 1999. (Pure and Applied Mathematics, v. 220).

IRVINE, A. D. Bertrand russell. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy.* Winter 2014. [S.l.: s.n.], 2014.

KNUTH, D. E.; BENDIX, P. B. Simple word problems in universal algebras. In: *Computational problems in abstract algebra.* [S.l.: s.n.], 1970. p. 263–297.

LEINSTER, T. Basic bicategories. *arXiv preprint math.CT/9810017*, Citeseer, 1998.

LEINSTER, T. *Higher Operads, Higher Categories.* [S.l.]: Cambridge University Press, 2004. (London Mathematical Society Lecture Note Series (Book 298)). Also http://arxiv.org/abs/math/0305049, May, 2003. ISBN 0-521-53215-9.

LUMSDAINE, P. L. Weak ω-categories from intensional type theory. In: *Typed lambda calculi and applications.* [S.l.]: Springer, 2009. (LNCS, v. 5608), p. 172–187.

MARQUIS, J.-P. Category theory. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy.* Winter 2014. [S.l.: s.n.], 2014.

MARTIN-LÖF, P. An intuitionistic theory of types: Predicative part. In: ROSE, H.; SHEPHERDSON, J. (Ed.). *Logic Colloquium '73 Proceedings of the Logic Colloquium.* Elsevier, 1975, (Studies in Logic and the Foundations of Mathematics, v. 80). p. 73 – 118. Available at: <http://www.sciencedirect.com/science/article/pii/S0049237X08719451>.

MARTIN-LÖF, P. Constructive mathematics and computer programming. In: COHEN, L.; LOS, J.; PFEIFFER, H.; PODEWSKI, K.-P. (Ed.). *Logic, Methodology and Philosophy of Science VI, Hannover, 1979.* [S.l.]: North-Holland, 1982, (Studies in Logic and the Foundations of Mathematics, v. 104). p. 153–175.

MARTIN-LÖF, P. *Intuitionistic Type Theory. Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980.* [S.l.]: Bibliopolis, Napoli, 1984. xii+92pp p.

MARTIN-LÖF, P. An intuitionistic theory of types. In: SAMBIN, G.; SMITH, J. (Ed.). *Twenty-five years of constructive type theory.* [S.l.]: Oxford University Press, 1998, (Oxford Logic Guides, v. 36). p. 127–172. ISBN 978-0198501275.

NLAB. *product.* 2014. Http://ncatlab.org/nlab/show/product (Versão 19). Last Review by Urs Schreiber in July 19, 2014. Access in March 24, 2015.

NLAB. *monoids.* 2017. Https://ncatlab.org/nlab/history/monoid (Versão 34). Last review by Urs Schreiber in May 26, 2017. Access in June 19, 2017.

OLIVEIRA, A. G. de. *Proof transformations for labelled natural deduction via term rewriting.* 1995. Master's thesis, Depto. de Informática, Universidade Federal de Pernambuco, Recife, Brazil, April 1995.

OLIVEIRA, A. G. de; QUEIROZ, R. J. G. B. de. Term rewriting with labelled deductive systems. In: *Proceedings of Brazilian Symposium on Artificial Intelligence (SBIA'94).* [S.l.: s.n.], 1994. p. 59–72.

OLIVEIRA, A. G. de; QUEIROZ, R. J. G. B. de. A normalization procedure for the equational fragment of labelled natural deduction. *Logic Journal of IGPL*, Oxford Univ Press, v. 3, n. (2–3), p. 243–290, 1999.

QUEIROZ, R. J. G. B. de; GABBAY, D. The functional interpretation of the existential quantifier. *Bulletin of the IGPL*, Oxford Univ Press, v. 7, n. 2, p. 173–215, 1995. (Special Issue on Deduction and Language, Guest Editor: Ruth Kempson). Full version of a paper presented at Logic Colloquium '91, Uppsala. Abstract in JSL 58(2):753–754, 1993.

QUEIROZ, R. J. G. B. de; GABBAY, D. M. Equality in labelled deductive systems and the functional interpretation of propositional equality. In: DEKKER, P.; STOCKHOF, M. (Ed.). *Proceedings of the 9th Amsterdam Colloquium*. [S.l.], 1994. p. 547–565.

QUEIROZ, R. J. G. B. de; OLIVEIRA, A. G. de. Natural deduction for equality: The missing entity. In: PEREIRA, L.; HAEUSLER, E.; PAIVA, V. de (Ed.). *Advances in Natural Deduction*. [S.l.]: Springer, 2014. p. 63–91.

QUEIROZ, R. J. G. B. de; OLIVEIRA, A. G. de. *Propositional Equality, Identity Types and Reversible Rewriting Sequences as Homotopies*. 2014. Propositional Equality, Identity Types and Reversible Rewriting Sequences as Homotopies, Talk at Logic Workshop, Universidade Federal do Ceará, Fortaleza, CE, Brazil.

QUEIROZ, R. J. G. B. de; OLIVEIRA, A. G. de; GABBAY, D. M. *The Functional Interpretation of Logical Deduction*. [S.l.]: World Scientific, 2011.

QUEIROZ, R. J. G. B. de; OLIVEIRA, A. G. de; RAMOS, A. F. Propositional equality, identity types, and direct computational paths. *South American Journal of Logic*, v. 2, n. 2, p. 245–296, 2016. Special Issue: A Festschrift for Francisco Miraglia.

RAMOS, A. F. *Identity Type as the Type of Computational Paths*. 2015. Master's thesis, Depto. de Informática, Universidade Federal de Pernambuco, Recife, Brazil, July, 2015.

RAMOS, A. F. *Computational paths, transport and the univalence axiom*. 2017. Talk at XVIII Brazilian Logic Conference, Pirenópolis, Goiás, May 2017.

RAMOS, A. F.; QUEIROZ, R. J. G. B. de; OLIVEIRA, A. G. de. On the identity type as the type of computational paths. *Logic Journal of IGPL*, Oxford Univ Press, v. 3, n. (2–3), p. 243–290, 2017.

RAMOS, A. F.; QUEIROZ, R. J. G. B. de; OLIVEIRA, A. G. de. *Explicit Computational Paths*. 2018. Https://arxiv.org/abs/1609.05079.

Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: <http://homotopytypetheory.org/book>, 2013.

VERAS, T. M. L. de; RAMOS, A. F.; QUEIROZ, R. J. G. B. de; OLIVEIRA, A. G. de. *On the Calculation of Fundamental Groups in Homotopy Type Theory by Means of Computational Paths*. 2018. Https://arxiv.org/abs/1804.01413.

VOEVODSKY, V. *Univalent Foundations and Set Theory*. 2014. Univalent Foundations and Set Theory, Lecture at IAS, Princeton, New Jersey, Mar 2014.