Higher Structures in Homotopy Type Theory

Antoine Allioux

Université Paris Cité

Université Paris Cité

École de Sciences Mathématiques Paris Centre (ED 386)
Institut de Recherche en Informatique Fondamentale (IRIF)

Thèse de doctorat en informatique
Dirigée par Pierre-Louis Curien

Higher Structures in Homotopy Type Theory

par Antoine Allioux

Présentée et soutenue publiquement le 17 *juillet* 2023
devant un jury composé de :

Pierre-Louis Curien	DR émérite	Université Paris Cité	Directeur de thèse
Eric Finster	Lecturer	University of Birmingham	Co-encadrant de thèse
Matthieu Sozeau	CR	Centre Inria de l'Université de Rennes	Co-encadrant de thèse
Marcelo Fiore	Prof.	University of Cambridge	Rapporteur
Samuel Mimram	Prof.	École Polytechnique	Rapporteur
Martín H. Escardó	Prof.	University of Birmingham	Examinateur
Nicolai Kraus	Assoc. prof.	University of Nottingham	Examinateur
François Métayer	MCF	Université Paris Nanterre	Examinateur
Paige R. North	Asst. prof.	Utrecht University	Examinatrice
Emily Riehl	Prof.	Johns Hopkins University	Examinatrice

Higher Structures in Homotopy Type Theory

Antoine Allioux

PhD Thesis

Abstract

The definition of algebraic structures on arbitrary types in homotopy type theory (HoTT) has proven elusive so far. This is due to types being spaces instead of plain sets in general, and equalities of elements of a type behaving like homotopies. Equational laws of algebraic structures must therefore be stated coherently. However, in set-based mathematics, the presentation of this coherence data relies on set-level algebraic structures such as operads or presheaves which are thus not subject to additional coherence conditions. Replicating this approach in HoTT leads to a situation of circular dependency as these structures must be defined coherently from the beginning.

In this thesis, we break this situation of circular dependency by extending type theory with a universe of cartesian polynomial monads which, crucially, satisfy their laws definitionally. This extension permits the presentation of types and their higher structures as opetopic types: infinite collections of cells whose geometry is described by opetopes. Opetopes are geometric shapes originally introduced by Baez and Dolan in order to give a definition of n-categories.

We open this thesis by giving a purely type-theoretical definition of opetopes in a type theory similar to book HoTT in Chapter 1. Our definition essentially defines opetopes as sequences of well-founded trees satisfying some properties which are nicely captured by their typing. The opetopic approach particularly shines in the context of type theory as well-founded trees fall within the realm of inductive types. More specifically, our construction is based on a sequence of cartesian polynomial monads, a notion which becomes central in later chapters. We conclude this chapter with an inductive definition of the faces of an opetope. This self-contained chapter is the occasion for the reader to get familiar with opetopes in type theory which are sets and therefore elude considerations of coherences before broaching on opetopic types whose complexity may hide the conceptual simplicity of opetopes.

We then extend type theory with a universe of cartesian polynomial monads closed under some monad constructors in Chapter 2. We do so in the aim of defining opetopic types in Chapter 3. Contrary to our definition of opetopes which only involves sets, opetopic types are valued in arbitrary types. As a consequence, we can no longer state the equational laws of Chapter 1 in a coherent fashion, having no means to do so. We therefore define our universe of polynomial monads in order that these equational laws hold definitionally. The constructors under which our universe is closed then allow us to define, in particular, the Baez-Dolan slice construction on which is based our definition of opetopic types. We then define a number of extensions in order to establish some more advanced results in Chapter 3.

Finally, we take advantage of our universe of polynomial monads to define opetopic types in Chapter 3. This enables us to define coherent higher algebraic structures, among which ∞-groupoids and $(\infty, 1)$-categories. Crucially, their higher structure coincides with the one induced by their identity types. We

then establish some expected results in order to motivate our definitions. In particular, we compare our definition of fibrant opetopic types with Baez and Dolan definition of coherent algebras, and we show that they are equivalent under certain assumptions.

Keywords: homotopy type theory, higher algebra, higher category theory, polynomial monads, opetopes

Résumé

La définition de structures algébriques sur des types arbitraires en théorie des types homotopiques (HoTT) s'est révélée hors de portée jusqu'à présent. Cela est dû au fait que les types sont, en général, des espaces plutôt que de simples ensembles, et que les égalités d'éléments d'un type se comportent comme des homotopies. Les lois équationnelles des structures algébriques doivent donc être énoncées de manière cohérente. Cependant, en mathématiques ensemblistes, la présentation de ces données de cohérence se fait à l'aide de structures algébriques sur des ensembles, telles que les opérades ou les préfaisceaux, qui ne sont donc pas soumises à des conditions de cohérence supplémentaires. Reproduire cette approche en HoTT conduit à une situation de dépendance circulaire puisque ces structures doivent être définies de manière cohérente dès le départ.

Dans cette thèse, nous brisons cette circularité en étendant la théorie des types d'un univers de monades polynomiales cartésiennes qui, de manière cruciale, satisfont leurs lois définitionnellement. Cette extension permet de présenter les types et leurs structures supérieures sous forme de types opétopiques qui sont des collections infinies de cellules dont la géométrie est décrite par les opétopes. Les opétopes sont des formes géométriques introduites par Baez et Dolan afin de donner une définition des n-catégories.

Nous ouvrons cette thèse en donnant une définition des opétopes dans une théorie des types similaire à celle du livre HoTT au chapitre 1. Nous définissons les opétopes comme séquences d'arbres bien fondés satisfaisant certaines propriétés qui sont capturées par leur typage. L'approche opétopique est particulièrement adaptée au contexte de la théorie des types car ces arbres sont aisément définissables comme types inductifs. Plus précisément, notre construction est basée sur une séquence de monades polynomiales cartésiennes, une notion qui devient centrale dans les chapitres suivants. Nous concluons ce chapitre par une définition inductive des faces d'un opétope. Ce chapitre est l'occasion pour le lecteur de se familiariser avec les opétopes en théorie des types qui sont des ensembles et qui échappent donc aux considérations de cohérence avant d'aborder les types opétopiques dont la complexité peut obscurcir la simplicité conceptuelle des opétopes.

Nous étendons ensuite la théorie des types d'un univers de monades polynomiales cartésiennes clos sous certains constructeurs de monades au chapitre 2 qui nous servira à définir les types opétopiques au chapitre 3. Contrairement à notre définition des opétopes qui n'implique que des ensembles, les types opétopiques sont à valeurs dans des types arbitraires. Par conséquent, nous ne pouvons plus énoncer les lois équationnelles du chapitre 1 de façon cohérente. Nous définissons donc notre univers de monades polynomiales afin que ces lois soient satisfaites par définition. Les constructeurs sous lesquels notre univers est clos nous permettent alors de définir, en particulier, la construction tranche de

Baez et Dolan sur laquelle repose notre définition de type opétopique. Nous définissons finalement un certain nombre d'extensions afin d'établir des résultats plus avancés au chapitre 3.

Enfin, nous tirons parti de notre univers de monades polynomiales pour définir les types opétopiques au chapitre 3. Cela nous permet de définir des structures algébriques supérieures cohérentes, parmi lesquelles les ∞-groupoïdes et les $(\infty, 1)$-catégories. De manière cruciale, leur structure supérieure coïncide avec celle induite par leurs types d'identités. Nous établissons ensuite quelques résultats attendus afin de motiver nos définitions. En particulier, nous comparons notre définition de type opétopique fibrant avec la définition d'algèbre cohérente de Baez et Dolan, et nous montrons qu'elles sont équivalentes sous certaines hypothèses.

Mots clés : théorie des types homotopiques, algèbre supérieure, théorie des catégories supérieures, monades polynomiales, opétopes

Acknowledgements

I could not have undertaken this journey without the people who made it possible. I am deeply indebted to Matthieu Sozeau, my unofficial supervisor, for giving me the opportunity to do this PhD. Eric Finster for his pioneering work on opetopic approaches in type theory: my thesis owes a lot to Eric's work and it has been a pleasure to collaborate with him. Samuel Mimram and Marcelo Fiore, the "rapporteurs" as we say in French, for the time they dedicated to the review and the evaluation of this thesis, and for their helpful comments. The members of the jury: Emily Riehl, Paige North, Martín Escardó, Nicolai Kraus, and François Métayer for attending the defence and for their interesting questions. Pierre-Louis Curien, my official thesis supervisor for administrative reasons, for having coordinated the organisation of the defence. I also wish to thank Jamie Vicary, who supervised my Master's thesis at the University of Oxford in 2015 and with whom I had my first research experience, as well as Michele Pagani and Thomas Ehrhard who supervised my internship at IRIF in 2016.

I am grateful to my colleagues and friends for their support and for the many interesting discussions that we have had throughout these years and that, I hope, we will continue to have. In particular, I would like to thank Léonard for the numerous evenings spent at the Cuves de Fauve talking about the connections between type theory and higher category theory. Raphaël for his interesting perspectives on artificial intelligence, a recent interest of mine, and for our numerous adventures, notably in New York City. Ésaïe for his kindness, our discussions on artificial intelligence and finance, and for our nights out; I am glad to have found a fellow clubber! Daniel for his love for the formalisation of mathematics in proof assistants that I share with him and for the discussions that we have had on countless topics. Chait for his kindness and for his consistently thoughtful opinions on many topics including but not limited to maths. Sarah for our trips to the catacombs and, in particular, for this memorable orienteering we participated to down there. Jules for his kindness and for his interesting reading lists. Léa for her consistent support and for her interesting and singular perspectives, it is good not to be surrounded only by scientists! Adelin and Alexis for their friendship. Ben and Alexia for their hospitality. Pierre for staying true to himself in all circumstances. Sacha for our nights out. Doris for her hospitality and her gentleness. Hugo that I wish I had seen more often despite sharing the same office. Fanny for the hikes in the south of France which were a welcome break from Paris. Guyjean and Zandeck, that I have known since the "prépa", and with whom I have managed to stay in touch daily. The friends that I met at IRIF in 2016 while interning there: Ludovic, Théo, Cyrille, Maxime, Joey. The people that I have met at IRIF more recently: Félix, Colin, Gaëtan, Kostia, Moana. My friends from the weekly bar of iiens which took my mind off work: moïse, Cara, Amora, corn, Fériel.

Lastly, I am thankful to my parents and my brother, Julien, for their support. I never lacked anything and I realise how privileged I was in this respect.

Contents

Introduction

Algebra in type theory

More than fifty years after its inception, Martin-Löf type theory (MLTT) in its intensional form continues to be a source of wonder. This past decade has been marked by the development of homotopy type theory (HoTT), a field which stemmed from the discovery that Martin-Löf identity types provide a link between type theory and homotopy theory, a branch of algebraic topology. Simply put: equalities of elements of a type can be regarded as paths in a space that the type represents, with equalities *of* equalities corresponding to higher dimensional paths. Under this correspondence, we can use topological intuition when working with types and their proof-relevant equality types.

The notion of homotopy type first arose in algebraic topology from the desire to consider topological spaces up to a coarser notion of equivalence, namely *homotopy equivalence*, where we ask not that a function have an inverse "on the nose", but up to deformation. The homotopy type of a space is the structure which only contains the information of the algebra of paths in the space and higher paths between them.

Over time, the notion of homotopy type became an object of study in itself. The desire to axiomatise this algebra of paths led to the connection between algebraic topology and higher category theory. Alexander Grothendieck's *homotopy hypothesis* asserts exactly that the algebra of higher paths can be understood as an ∞-groupoid, that is, a weak higher-dimensional category in which all higher morphisms are invertible. More recently, it was discovered that identity types of HoTT give a precise axiomatisation of ∞-groupoids. Several fundamental results in the field have made this connection precise, showing that type theory gives rise to particular definitions of ∞-groupoids (Lumsdaine 2010; Van Den Berg and Garner 2011). This connection has been further generalised to establish that HoTT is the internal language of $(\infty, 1)$-topoï (Shulman 2019). Establishing the connection between types and ∞-groupoids allows regarding statements about types as statements about ∞-groupoids and vice versa. This has allowed the reformulation of a great body of homotopy theory results in a more conceptual way and this has led to the development of new proof techniques (Univalent Foundations Program 2013).

In light of these connections, it is possible to think of type theory as a foundation of mathematics based on strong principles, among which Voevodsky's *Univalence Principle* asserting that all statements about types are invariant under

equivalence (AWODEY 2014).

This foundation of mathematics should have its own theory of algebra. This, however has proven problematic. The reason is that defining algebraic structures on homotopy types requires that we do so *coherently*; otherwise, some expected constructions cannot be defined. So far, we have not been able to do so without assuming additional principles such as the uniqueness of identity proofs (UIP) which are not compatible with the homotopy interpretation of types that we subscribe to in HoTT. The difficulty lies in the description of the laws satisfied by the algebraic structure at hand. This problem is usually tackled in set-based mathematics — where spaces are defined in terms of sets — by using set-level algebraic structures such as operads or presheaves in order to present algebraic structures on spaces and their infinite data of coherences. However, this approach is denied to us in HoTT where types behave like spaces with equalities between elements of a type behaving like homotopies. This is a blessing as this allows to state constructions in a homotopy invariant way, but this is also a curse as this seems to prevent us from defining algebraic structures on arbitrary types. Related to this problem is the open question of defining semi-simplicial types. It was raised during the special year on Univalent Foundations at the Institute for Advanced Study in 2013 and all attempts to solve it have required modifying type theory in some way by reintroducing a strict equality (ALTENKIRCH, CAPRIOTTI and KRAUS 2016; ANNENKOV et al. 2017; CAPRIOTTI and KRAUS 2017; KRAUS and SATTLER 2017) thus defeating the foundational ambition of HoTT.

The objective of this thesis is therefore to provide the means to define a range of fully coherent and non-truncated higher algebraic structures in type theory while retaining the homotopy interpretation of types. The approach we adopt consists in extending type theory with a universe of polynomial monads. Crucially, the monad structure is defined in such a way that the laws hold definitionally. These structures are enough to define opetopic types: type-valued collections of cells whose geometry is governed by opetopes. Opetopes are many-to-one geometric shapes used in geometric approaches to higher algebra. A construction of opetopes based on polynomial monads known as the Baez-Dolan construction (BAEZ and DOLAN 1998) underlies our approach. Equipped with this structure of opetopic type, we are able to define a number of higher algebraic structures, among them being ∞-groupoids and $(\infty, 1)$-categories. We then develop opetopic methods that we apply to prove elementary facts about our higher algebraic structures.

We have implemented our system in the Agda proof assistant which has been an invaluable assistance in order to develop our ideas. We took advantage of its facilities allowing to postulate new constants and rewriting rules in order that our monads enjoy definitional laws.

Before introducing our approach in greater details, let us pause to see some of the subtleties of coherence definitions in type theory.

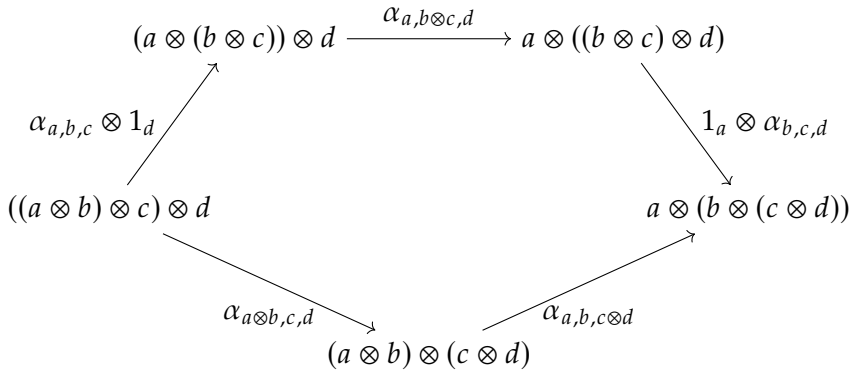


Figure 1: Stasheff polytope K_2

Coherence

A subtle consequence of the non-trivial nature of the higher structure of types is that, when one uses identity types to state algebraic laws, one has to do so *coherently* unless uniqueness of identity proofs (UIP) is assumed. Without these coherence laws, expected mathematical results simply do not hold. For instance, we cannot define the slice category of a category in general. Without UIP, type theory admits models in spaces and intuition of homotopy theory applies. Definitions of algebraic structures therefore become infinitary in their presentation to account for their coherence laws.

Let us illustrate this phenomenon with the definition of an associative magma on a type X. This consists in a binary operation $\otimes : X \times X \to X$ satisfying the associative law

$$\alpha_{a,b,c} : (a \otimes b) \otimes c = a \otimes (b \otimes c)$$

for all elements $a, b, c : X$. The definitions does not end here however, as $\alpha_{a,b,c}$ is a piece of data in a proof-relevant setting and its choice can not be arbitrary. It has to be coherent which means that any diagram made of α such as the one depicted on Figure 1 must commute. This condition is stated as an identity between the two paths of this diagram. In turn, this new identity can not be arbitrary and must satisfy its own coherence laws leading to an infinite tower of data whose geometry is described by Stasheff polytopes K_n.

As an example of the ill-behaved nature of a non-coherent associative law, consider the definition in type theory of the slice category C/c where c is an object of a category C. We witness a switch in dimension where the data of dimension n of C/c uses the $(n+1)$-dimensional data of C. In particular, when defining the composition in C/c, one has to make use of the associative law in C. And when it comes to proving that the composition in C/c is associative, we are stuck unless we have the data of the MacLane's pentagon.

Specifying the infinite tower of coherence data in type theory in terms of identity types has proved elusive without the ability to resort to a strict equality which is not subject to these coherence conditions. In particular, this is the

approach taken by Voevodsky's homotopy type system (VOEVODSKY 2013) and two-level type theory (ALTENKIRCH, CAPRIOTTI and KRAUS 2016). In this thesis, we pursue another approach consisting in extending type theory with a universe of polynomial monads satisfying their laws definitionally. This lets us specify the necessary coherence data while retaining the homotopy interpretation of types.

Opetopic types

Our approach will consist in defining a higher-dimensional presentation of types as opetopic types. The geometry of opetopic types is governed by opetopes that can also be seen as the terminal opetopic type. Opetopes are geometric shapes, like simplices or globes, that were introduced by Baez and Dolan (BAEZ and DOLAN 1998) in order to define n-categories. We chose opetopes as they can be defined using cartesian polynomial monads which lend themselves well to a definition in type theory.

Opetopes are many-to-one cells whose $(n+1)$-cells can be seen as relating a source formal composite of n-cells with a target n-cell. We understand $(n+1)$-opetopic cells as witnessing the composition of pasting diagrams of n-cells. $(n+2)$-cells then witness the laws satisfied by the composition of n-cells, $(n+3)$-cells witness some additional coherence data that the laws satisfy, and so forth. If enough of these cells exist, in a sense that we will make precise later, we say that the opetopic type is *fibrant* and we regard it as an infinite-dimensional collection of cells describing a fully coherent algebraic structure.

Let us depict some examples of opetopic cells in order to gain intuition.

0-cells They are the objects of our opetopic types, their depiction is therefore 0-dimensional:

$$a \qquad b \qquad c$$

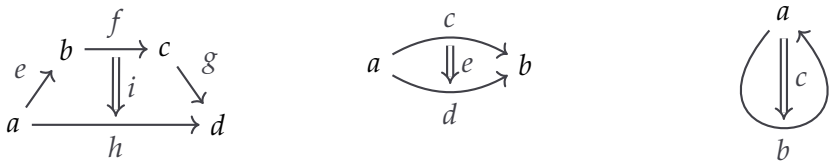
1-cells They relate a single source 0-cell to a single target 0-cell hence their depiction as arrows:

$$a \xrightarrow{c} b$$

Starting from this dimension we can talk about formal composites of n-cells named *pasting diagrams*. In dimension 1 they are just linear trees of arrows such as the following one:

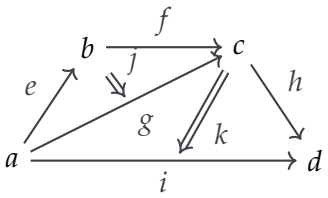
$$a \xrightarrow{e} b \xrightarrow{f} c \xrightarrow{g} d$$

2-cells They relate a source pasting diagram of 1-cells — a possibly empty linear tree — to a single arrow. Some examples of 2-cells are the following:



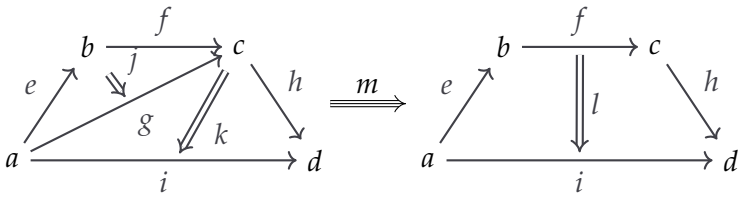
We understand these cells as witnessing that a certain pasting diagram of arrows composes to a particular arrow. The geometry of opetopes allows the expression of an unbiased notion of composition. The units will be given by the composition of empty pasting diagrams as depicted in the third case. Unary cells which relate a pair of parallel arrows (i.e., arrows sharing the same source and the same target) are particular cases of opetopic cells. Given that unary cells exist in all dimensions, opetopic types subsume globular types.

2-cells can be assembled into pasting diagrams by gluing the target of a 2-cell to a matching source of another 2-cell as illustrated in the next diagram:



The two cells j and k are glued along the common face g.

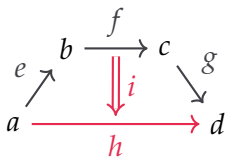
3-cells As a last example, consider the following 3-cell m:



Its source — the diagram on the left — is a pasting diagram of 2-cells; that is, a formal composite of the 2-cells j and k glued along a common face g. Its target is the 2-cell l whose frame — its source pasting diagram along with its target cell — matches the one of the source pasting diagram of m.

Fibrant opetopic types We are interested in opetopic types which satisfy the following property: for any pasting diagram of n-cells, there exists a unique composite n-cell sharing the same frame. We say that such an opetopic type is *fibrant*. They are the opetopic types which represent fully coherent algebras.

As an example, such an opetopic type would guarantee the existence of the red data displayed on the following diagram given the black source pasting diagram:



Moreover, the red data formed of the pair comprising the cell h and the cell i witnessing that h is a composite of its source pasting diagram is *unique*. By unique, we mean that the type of pairs, of which (i, h) is a member, is contractible. The notion of contractibility is fundamental in HoTT and is easily defined.

Polynomial monads

Cartesian polynomial monads form the backbone of our system; we will use them to define opetopes. They are at the heart of the so-called Baez-Dolan construction (BAEZ and DOLAN 1998) which was introduced in the setting of operads with the aim of defining n-categories. Cartesian polynomial monads can be understood as presentations for strongly regular algebraic theories. These are algebraic theories whose equations are constrained in such a way that variables have to appear in both sides without repetition and in the same order (LEINSTER 2004). From now on, we will not systematically specify that our polynomial monads are cartesian even though they always are in order to lighten the notation.

The endofunctor part of the polynomial monad structure is also known as an indexed container in the type theory literature (ABBOTT, ALTENKIRCH and GHANI 2005; ALTENKIRCH, GHANI et al. 2015). They provide a calculus of datatypes used to internalise a large class of inductive datatypes in type theory allowing, for example, the conception of generic algorithms on datatypes.

Polynomial monads lend themselves well to a definition in type theory as indexed families. The data underlying a polynomial endofunctor and that we refer to as a *polynomial* is described by the following type families and typing function where $\mathcal{U}$ stands for the universe of types:

$$
\begin{aligned}
&\mathsf{Idx} : \mathcal{U} \\
&\mathsf{Cns} : \mathsf{Idx} \to \mathcal{U} \\
&\mathsf{Pos} : \{i : \mathsf{Idx}\} \to \mathsf{Cns}\ i \to \mathcal{U} \\
&\mathsf{Typ} : \{i : \mathsf{Idx}\}\ (c : \mathsf{Cns}\ i) \to \mathsf{Pos}\ c \to \mathsf{Idx}
\end{aligned}
$$

This data can be regarded as a description of a signature of an algebraic theory: the elements of Idx, which we refer to as *indices* serve as the sorts of the theory, and for i : Idx, the type Cns i is the collection of operation symbols whose "output" sort is i. The type Pos c is then the collection of "input positions" of the operation c which are themselves assigned an index via the function Typ.

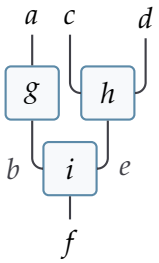
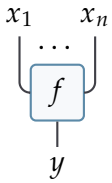


Figure 2: A P-tree

In this thesis, we will use a graphical language to depict the constructors of our polynomial monads. A constructor is depicted as a *corolla* whose output points downward and whose inputs point upward. In the following depiction of a corolla, the constructor is labelled f, the inputs are labelled x_i for $1 \leq i \leq n$, and the target is labelled y.



For any polynomial functor P, its constructors can be assembled into finite rooted trees named P-trees (Figure B.1) provided that for each inner edge, the index of the target and the index of the input of the corresponding constructors match.

The additional structure on a polynomial endowing the induced polynomial endofunctor with a structure of cartesian polynomial monad is specified, in part, by the following operations:

$$\eta : (i : \mathsf{Idx}) \longrightarrow \mathsf{Cns}\ i$$
$$\mu : \{i : \mathsf{Idx}\}\ (c : \mathsf{Cns}\ i)\ (d : (p : \mathsf{Pos}\ c) \longrightarrow \mathsf{Cns}\ (\mathsf{Typ}\ c\ p)) \longrightarrow \mathsf{Cns}\ i$$

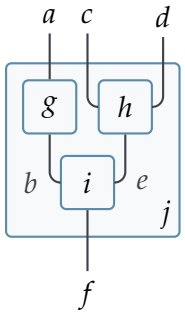
These operations have to satisfy certain laws that we do not detail in this section, but we will comment both operations.

For each index $i : \mathsf{Idx}$, there is a unary constructor $\eta\ i : \mathsf{Cns}\ i$ whose input's index is i. We can depict this constructor as follows:

As for the operation μ, its arguments comprise a constructor c as well as a family of constructors d indexed by the positions of c. We regard this data as specifying

a depth-2 tree such as the one displayed on the left of the following figure. The operation μ then sends this tree to a constructor sharing the same inputs and the same output such as the one depicted on the right. The fact that the input positions as well as their typing is preserved by the operation μ is due to the fact that our monads are cartesian.

An alternative depiction of a tree along with a constructor sharing its boundary consists in drawing the tree inside the constructor in question.



In addition, the operation μ has to be associative and unital with units specified by the operation η. We emphasize that we will implement our universe of polynomial monads by extending type theory in such a way that these laws hold definitionally.

Plan

We open this thesis by first giving a purely type-theoretical definition of opetopes in a type theory similar to book HoTT in Chapter 1. Our definition essentially defines opetopes as sequences of well-founded trees satisfying some properties which are nicely captured by their typing. The opetopic approach particularly shines in the context of type theory as well-founded trees fall within the realm of inductive types. More specifically, our construction is based on a sequence of polynomial monads, a notion which becomes central in later chapters. We conclude this chapter with an inductive definition of the faces of an opetope. This self-contained chapter is the occasion for the reader to get familiar with opetopes before broaching on opetopic types whose complexity may hide the conceptual simplicity of opetopes.

Equipped with the understanding of opetopes, a simple example of polynomial monads, we then extend type theory with a universe of cartesian polynomial monads $\mathcal{M}$ closed under some monad constructors in Chapter 2. We

do so in the aim of defining opetopic types, collection of types whose combinatorics is described by opetopes, in Chapter 3. Contrary to our definition of opetopes which only involves sets, opetopic types are valued in arbitrary types. As a consequence, we can no longer state the equational laws of Chapter 1 in a coherent fashion, having no means to do so. We therefore define our universe of polynomial monads in order that these equational laws hold definitionally. The constructors under which our universe is closed then allow us to define, in particular, the sequence of polynomial monads defining opetopes. In addition to this universe which constitutes the core of our addition to type theory, we add two sorts of universes as further extensions. First, we define a family of universes $M \to_m N$ of cartesian monad morphisms from M to N. Then, we define a family of universes $\mathcal{M}\downarrow_M$ of polynomial monad families over on a monad M. These two extensions are developed in order to establish some more advanced results in Chapter 3.

Finally, we take advantage of our universe of polynomial monads to define opetopic types in Chapter 3. This enables us to define coherent higher algebraic structures, among which ∞-groupoids and $(\infty, 1)$-categories. Crucially, their higher structure is encoded in terms of identity types. We then establish expected results in order to motivate our definitions. We first compare our definition of fibrant opetopic type with Baez and Dolan definition of "coherent O-algebras" and show that they are equivalent if we require all morphisms to be invertible. Then, we establish that set-truncated fibrant M-opetopic types are equivalent to precategories as defined in the HoTT book for some monad M. We quickly reach for the dependent monads as alluded earlier in order to establish the remaining results. We start by defining fibrations of opetopic types which are a generalisation of fibrant opetopic types then we show that fibrant opetopic types are closed under dependent sums. We also define the universal fibration of types as an opetopic type. Finally, we define Grothendieck (op)fibrations that we use to define Lurie's definition of an adjunction (LURIE 2009, Definition 5.2.2.1) as a bifibration over the interval.

Formalisation

The vast majority of the results presented in this thesis have been formalised in the Agda proof assistant (AGDA DEVELOPMENT TEAM 2022) and can be found at the following address.

`https://github.com/allioux/thesis-formalisation`

Most of the types involved in this thesis fall within the scope of the types which can be defined in Agda except for the universe of polynomial monads $\mathcal{M}$ which is the subject of Chapter 2. We resorted to some features provided by Agda which allow to postulate new constants and new rewriting rules (COCKX 2020). This permits the definition of the different rules characterizing the universe $\mathcal{M}$ with the definitional equalities being defined as oriented rewriting rules.

Related work

As previously stated, we do not know if it is possible to express the coherences of arbitrary algebraic structures internally to homotopy type theory. Such phenomena are typically handled using a strict equality in set-based mathematics. The reintroduction of a strict equality in type theory in addition to the homotopical one forms the basis of two-level type theory (ALTENKIRCH, CAPRIOTTI and KRAUS 2016; ANNENKOV et al. 2017; CAPRIOTTI and KRAUS 2017; KRAUS and SATTLER 2017). Briefly, it consists in internalising homotopy type theory inside a second type theory satisfying the UIP. In such a type theory, we distinguish between the *fibrant* types belonging to the HoTT fragment and the non-fibrant types which live in the strict fragment. In addition, going back and forth between the two fragments is subject to some constraints. However, one drawback of this approach is that it leads us to abandon the unrestricted interpretation of all types as homotopy types since it does not apply to the non-fibrant fragment. Furthermore, introducing a second type theory in which univalence does not apply undermines the idea that mathematics can be based on homotopy theory. By contrast, our proposal preserves the unrestricted homotopy interpretation of types.

Chapter 1

Opetopes in type theory

In this chapter, we give a purely type-theoretical treatment of opetopes. Opetopes will be a recurring theme of this thesis and will constitute the backbone of an extension of type theory based on opetopic types that will be developed in the next chapters.

1.1 Introduction

Opetopes are geometric shapes intended to capture the combinatorics of many-to-one cells involved in certain definitions of weak n-categories and ∞-categories which are then defined as opetopic sets satisfying some properties. The 0-cells represent the objects of the categories, the 1-cells represent the morphisms, the 2-cells represent the composition of morphisms, the 3-cells represent the laws satisfied by the composition of morphisms, and so forth.

Opetopes were first introduced by Baez and Dolan (Baez and Dolan 1998) as a construction involving symmetric operads. Hermida, Makkai, and Power then gave their own definition based on generalised multicategories instead of operads and called them multitopes (Hermida, Makkai and Power 2000). Leinster defined opetopes in the more general setting of T-multicategories where T is a cartesian monad (Leinster 2001). All these definitions of opetopes were shown to be equivalent by Cheng (Cheng 2004a,b).

Finally, Kock et al. gave a definition of opetopes in the setting of cartesian polynomial monads (Kock et al. 2010) which naturally arise in locally cartesian closed categories (LCCC). They showed that their opetopes are equivalent to Leinster's ones. Our formulation of opetopes in type theory draws from the definition of Kock et al. which lends itself well to this task as type theory admits semantics in LCCC.

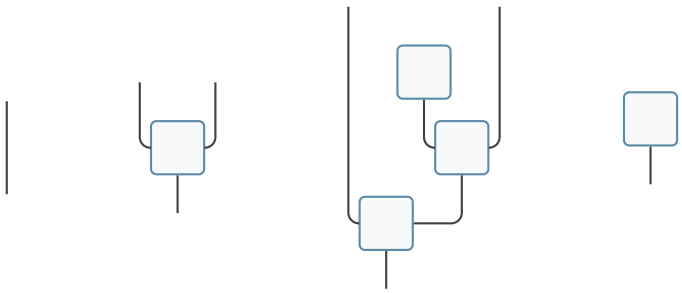
All these definitions have in common that opetopes in dimension $n+1$ are generated from opetopes in dimension n by the so-called *slice construction* which can be adapted to the different mentioned settings.

1.2 Diagrammatic depiction of opetopes

We present an informal and diagrammatic depiction of opetopes inspired from Kock's "5-minute definition" of opetopes (KOCK et al. 2010). This section aims at giving the reader an intuitive understanding of opetopes. We will give our own formal definition of opetopes in type theory in a further section.

1.2.1 Trees

Our definition will make use of directed finite rooted trees. This means that our trees have a single root that we draw at the bottom. Each node of a tree has a finite number of input edges and a single output edge. Edges which are both an input edge and an output edge are called internal edges. The other edges are drawn with a loose end corresponding to the absence of a source or a target. A single edge which is neither an input edge nor an output edge is a valid tree. The following are all examples of trees:



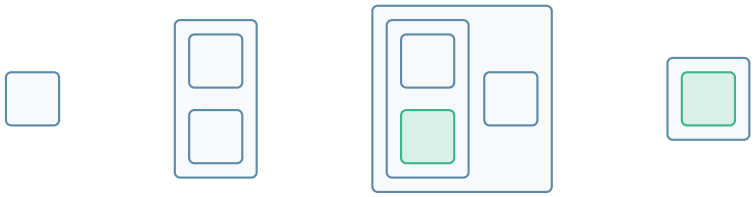
Blue boxes represent the nodes of our trees while black wires represent their edges. We will sometimes label the nodes and the edges so that we can refer to them. Any tree is endowed with a partial order on the collection of edges with the root being the maximal element and the leaves being minimal elements.

1.2.2 Nestings

We introduce an equivalent representation of trees named *nestings*. A nesting is a collection of non-intersecting boxes included in the plane endowed with a partial order induced by the inclusion relation. It must have a greatest element — there exists a box in which all the other boxes are included — and the minimal elements are the empty boxes. For the representation to be equivalent to trees, we also have to distinguish a subset of the minimal elements which correspond to the root edges of the nodes of the corresponding tree having no source.

The following nestings correspond to the four trees depicted in the previous

section in a sense that we will soon make precise.



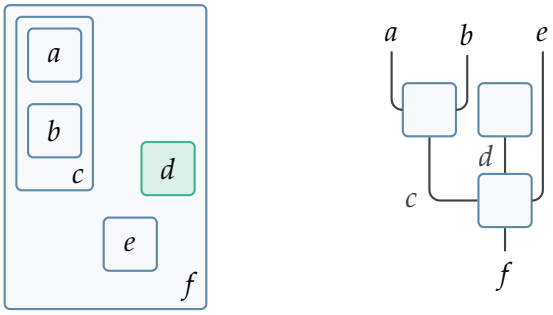
Here, we have chosen to draw the distinguished subset of empty boxes in green.

1.2.3 Correspondences

A *correspondence* between a tree and a nesting consists in an isomorphism between the boxes of the nesting and the edges of the tree. This isomorphism has to satisfy the following properties:

- The unmarked empty boxes are in bijection with the leaves of the tree.
- The isomorphism has to preserve the partial orders.

An example of correspondence between a nesting and a tree is the following:



Here, the data of the correspondence is visible through the labelling.

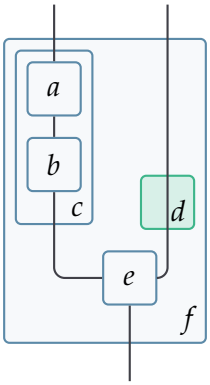
1.2.4 Subdivisions

We call *subdivision* the superposition of a nesting with a tree which satisfies the following properties:

- The unmarked empty boxes of the nesting are the nodes of the tree.
- The content of the other boxes must form a tree.

Note that these requirements force marked empty boxes to contain a single edge otherwise they would not be empty.

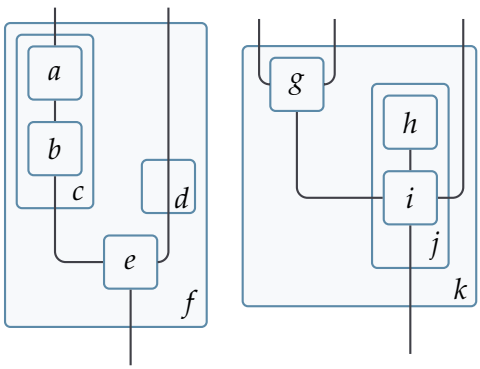
An example of such a subdivision is displayed in the following diagram:



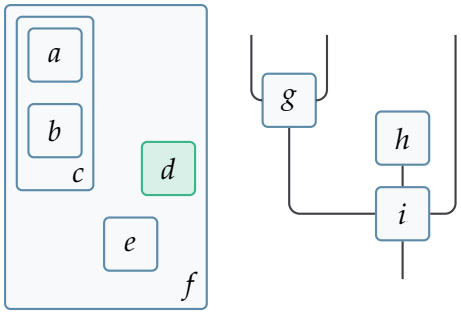
We will not mark the boxes of the nesting of a subdivision in the future as this information is already conveyed by the fact that these boxes are the ones containing a single edge. Subdivisions are called that way as they correspond to nested subdivisions of a tree.

1.2.5 Subdivision pairings

We define a *subdivision pairing* to be two subdivisions such that the nesting of the first subdivision and the tree of the second subdivision are in correspondence. Such a situation is depicted in the following diagram:



In the following diagram, we forget the tree of the first subdivision and the nesting of the second one to make the connection clearer. In doing so, we mark one node of the first nesting to remember that it used to contain a single edge.



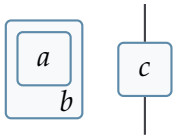
1.2.6 Opetopes

Opetopes are non-empty sequences of subdivisions such that two consecutive subdivisions form a subdivision pairing. Moreover, the first subdivision does not contain any tree, and is a single box if the sequence is of length 1 or is a linear nesting otherwise. The last subdivision of a sequence of length greater than one is a tree with a single node and no nesting. A sequence of n such subdivisions denotes an opetope of dimension $n-1$. We will also refer to opetopes of dimension n as n-opetopes. We name the leaves of the last tree in the sequence the *sources* of the opetope while its root is called the *target* of the opetope. We now illustrate this definition by presenting some examples of opetopes.

The object There is a single opetope of dimension 0 called the *object*.

The object has no source nor target. It is a nesting consisting of a single empty box.

The arrow There is a single opetope of dimension 1 called the *arrow*.

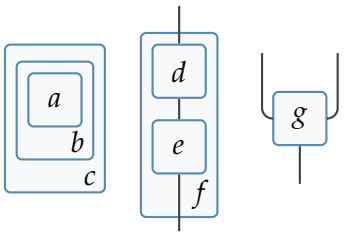


Boxes constituting an opetope are called *faces* and each corresponds to an opetope. For example, the arrow has three faces. Its top face c is the arrow itself while both its source a and its target b are objects. We will often conflate the face of an opetope with the opetope it denotes.

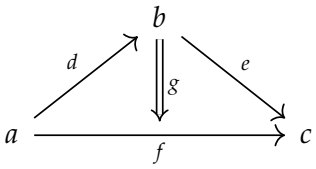
A more common depiction of the arrow is the following:

$$a \xrightarrow{\ c\ } b$$

The 2-simplex The 2-simplex (a triangle) is then defined as follows, taking the second tree to be a linear tree with two nodes:

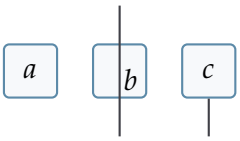


The 2-simplex is more commonly depicted as a triangle:

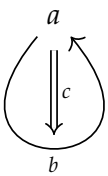


Note that the second tree will always be linear as the order of the first nesting is linear.

Loops Starting from dimension 2, we can have loops which are arrows having same source and target whose filler is witnessed by a higher-dimensional face with no source:

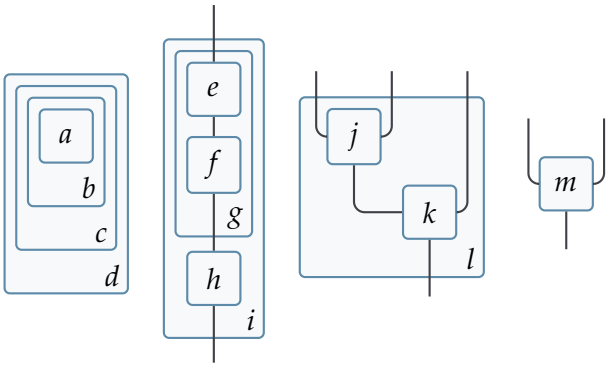


The loop is usually depicted as a loop whence its name:

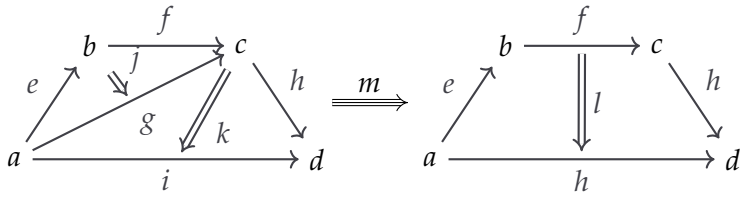


One last example Our interest in opetopes stems from the fact that opetopes capture the combinatorics of the higher dimensional laws that the composition of faces has to satisfy. We therefore expect to be able to denote a particular way of composing three unary faces for example. The following opetope of

dimension 3 can be seen as denoting a way of composing the faces e, f, and h:



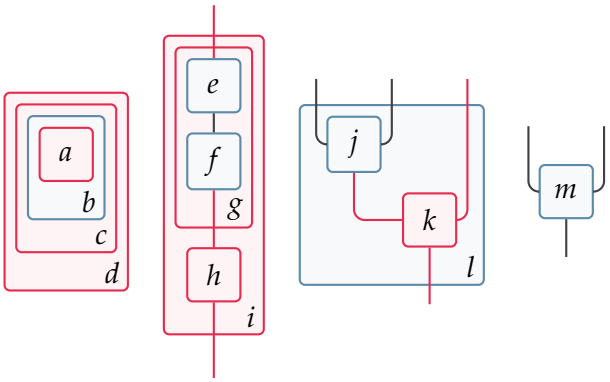
Once again, the previous diagram is more commonly depicted as follows:



We see that we are hitting the limits of the geometric depiction of opetopes, it is cumbersome to draw them in higher dimensions. The advantage of our notation is that it scales to any dimension due to the uniform nature of our trees: they are trees all the way up.

1.2.7 Faces of an opetope

We refer to the different boxes of a sequence of subdivisions forming an opetope as its *faces*. To each face corresponds an opetope that we can directly extract from the sequence. To illustrate this fact, we represent an opetope where we have highlighted the opetope corresponding to its face k:

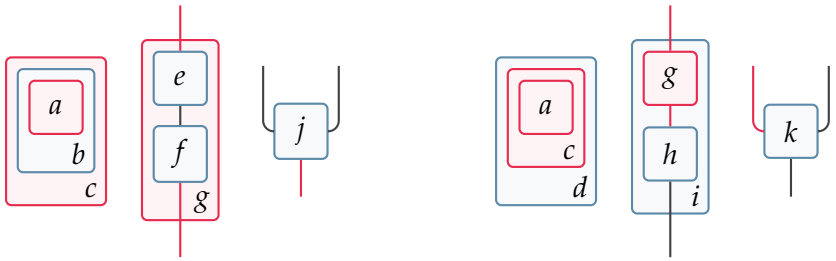


Note that the opetope corresponding to the face m is the whole opetope, we also say that m is its top face. We will often refer to an opetope using the name

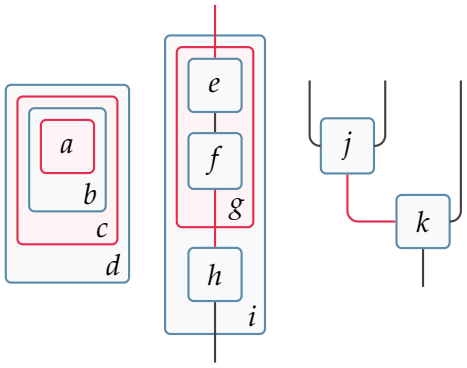
of its top face. We briefly describe how we obtained the highlighted opetope corresponding to the face k. First, we consider the corolla labelled k which will become the last subdivision of the new opetope we wish to create. Then, we proceed to determine the preceding subdivisions by decreasing dimension. Consider the edges touching k, they correspond to the faces g, h, and i. The second subdivision then forgets the faces e and f, and the face g becomes a node of a linear tree whose other node is h. Finally, consider the edges of this tree, they correspond to the faces a, c, and d forming a linear nesting. The sequence of subdivisions that we just obtained represents the opetope associated with the face k.

1.2.8 Grafting of opetopes

n-opetopes can be pasted together along a matching $(n-1)$-face — it is possible to paste along faces of lower dimension, but we will not cover this case here. We name *pasting diagram* the result of such a pasting. Consider the following two opetopes:



We want to graft j on k along the common face g whose corresponding opetope has been highlighted. The resulting pasting diagram is the following one:



The last diagram is the result of the graft of the corolla j on the corolla k along the common face g. The subdivision of dimension 1 has been obtained from the corresponding subdivision of the opetope k by substituting the corresponding subdivision of the opetope j for the face g. Finally, the subdivision of dimension 0 has been obtained from the corresponding subdivision of the opetope k by

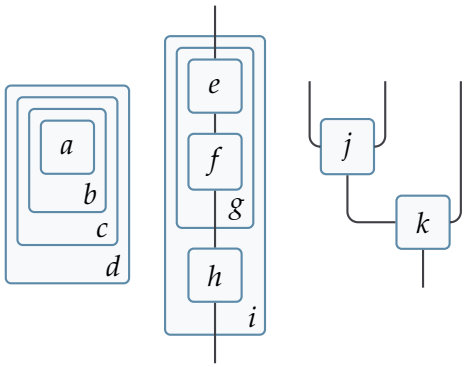
substituting the corresponding subdivision of the opetope j for the face c along with the content of its corresponding box.

Notably, any opetope can be regarded as a pasting diagram with a single node corresponding to its top face.

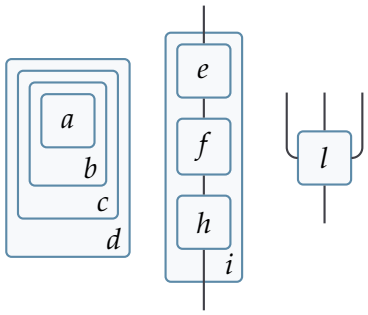
1.2.9 Composition of opetopes

Given a pasting diagram of opetopes, there exists a unique composite which is obtained by contracting the nodes of the tree in the last dimension and by deleting the boxes of the subdivision corresponding to the inner edges which have been removed. The composition is witnessed by a unique higher-dimensional face — its *filler* — whose sources are in correspondence with the nodes of the tree and whose target is the composite.

We illustrate the composition with the following pasting diagram:

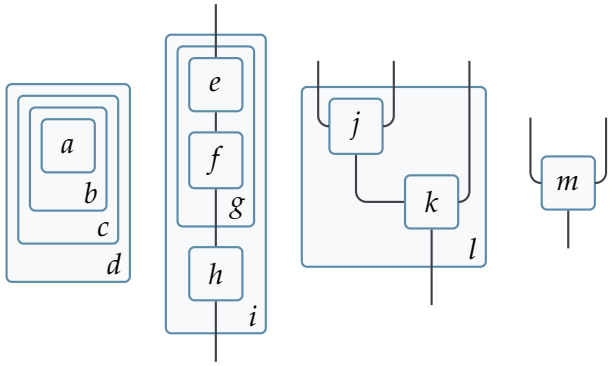


Its composite is:



The face l has been obtained by contracting the nodes of the tree. The face g corresponding to its inner edge has therefore been removed.

The opetope filler is witnessing that the tree composes to the composite:



Due to the fact that pasting diagrams of opetopes have a unique composite and a unique filler, opetopes are entirely determined by their source pasting diagram.

1.3 Opetopes in type theory

We now give a formal definition of opetopes in type theory. We will first introduce the different signatures before giving the precise definitions. We will define opetopes as the types of a sequence of cartesian polynomial monads. The data defining opetopes is summarised in Figure 1.1 while the operations of the monad structure are summarised in Figure 1.2. We will have more to say about polynomial monads in Chapter 2.

We briefly comment each type. The type $\mathcal{O}\ n$ is the type of opetopes of dimension n. The type $\mathcal{P}\ o$ is the type of pasting diagrams which compose to the opetope o. The type $\mathsf{Pos}\ x$ is the type of node locations of the pasting diagram x. The opetope $\mathsf{Typ}\ x\ p$ is then the opetope corresponding to the node of x located at position p.

Defining the different types and their operations is a little involved due to their mutual dependency. The definition of opetopes will then follow the following plan:

1. Definition of opetopes (Section 1.3.1).
2. Definition of pasting diagrams (Section 1.3.2).
3. Definition of positions (Section 1.3.3).
4. Informal presentation of the monad structure (Section 1.3.5).
5. Specification of positions operations (Section 1.3.6).
6. Definition of the monad structure (Section 1.3.7).
7. Definition of the positions operations (Section 1.3.8).

$$
\begin{aligned}
&\mathcal{O} : \mathbb{N} \to \mathcal{U} \\
&\mathcal{P} : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n)\ \to \mathcal{U} \\
&\mathsf{Pos} : \{n : \mathbb{N}\}\ \{o : \mathcal{O}\ n\} \to \mathcal{P}\ o \to \mathcal{U} \\
&\mathsf{Typ} : \{n : \mathbb{N}\}\ \{o : \mathcal{O}\ n\}\ (p : \mathcal{P}\ o) \to \mathsf{Pos}\ p \to \mathcal{O}\ n
\end{aligned}
$$

Figure 1.1: The data defining opetopes

$$
\begin{aligned}
&\eta : \{n : \mathbb{N}\}\ (x : \mathcal{O}\ n) \to \mathcal{P}\ x \\
&\mu : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ (y : \mathcal{P}\ x) \to \overrightarrow{\mathcal{P}}\ y \to \mathcal{P}\ x
\end{aligned}
$$

Figure 1.2: The monad structure

1.3.1 Definition of opetopes

We define opetopes of dimension n as sequences of trees of length $n + 1$ as we did in the informal presentation.

Definition 1.3.1 (Opetopes)**.** The family of opetopes

$$\mathcal{O} : \mathbb{N} \to \mathcal{U}$$

is defined as an inductive type with two constructors

$$
\begin{aligned}
&\mathsf{ob} : \mathcal{O}\ 0 \\
&\_ \triangleleft \_ : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n)\ \to \mathcal{P}\ o \to \mathcal{O}\ (n + 1)
\end{aligned}
$$

where the underscores indicate that $\_\triangleleft\_$ is a left-associative infix operator which, when applied to the arguments x and y, will be written $x \triangleleft y$.

Opetopes are either the object ob or an opetope of the form $x \triangleleft y$. The object has no source nor target while opetopes of the form $x \triangleleft y$ have a collection of sources specified by $\mathsf{Pos}\ y$ — the type of positions of the pasting diagram y — and a single target x. The typing ensures that y is *parallel* to x in that they share the same sources and the same target.

1.3.2 Definition of pasting diagrams

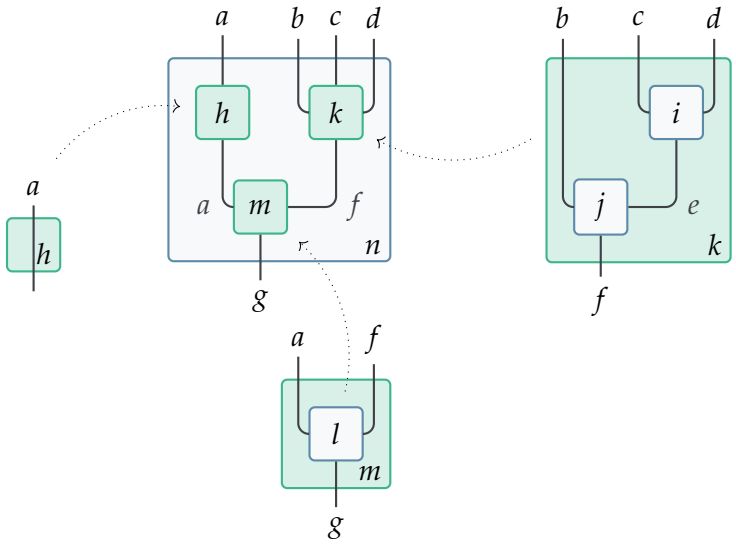
Before defining pasting diagrams, we introduce some notations to denote two different kind of families of pasting diagrams.

Notation. For any opetope $x : O\ n$ and any pasting diagram $y : \mathcal{P}\ x$, we introduce the notation

$$\overrightarrow{\mathcal{P}}\ y :\equiv (p : \mathsf{Pos}\ y) \to \mathcal{P}\ (\mathsf{Typ}\ y\ p)$$

denoting families of pasting diagrams indexed by the opetopes corresponding to the nodes of y.

Let $x : O\ n$ be an opetope and let $y : \mathcal{P}\ x$ be a pasting diagram. Consider the following figure:



Suppose that n is the top face of the opetope x and that the tree inside it is the parallel pasting diagram y. We regard a family $\overrightarrow{\mathcal{P}}\ y$ as specifying a family of pasting diagrams indexed by the nodes h, k, and m that we displayed here inside their indexing opetope surrounding the pasting diagram y.

We introduce a second notation for families of pasting diagrams in the next dimension.

Notation. For any opetope $x : O\ n$, any pasting diagram $y : \mathcal{P}\ x$, and any family of pasting diagrams $z : \overrightarrow{\mathcal{P}}\ y$, we introduce the notation

$$\overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z) :\equiv (p : \mathsf{Pos}\ y) \to \mathcal{P}\ (\mathsf{Typ}\ y\ p \triangleleft z\ p)$$

denoting families of pasting diagrams indexed by the opetopes $\mathsf{Typ}\ y\ p \triangleleft z\ p$ for any position $p : \mathsf{Pos}\ x$.

Consider again the previous figure and suppose that it displays the data of the family of pasting diagrams z indexed by the nodes of the pasting diagram

y. We think of a family of type $\overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)$ as specifying a family of pasting diagrams which, for each position $p : \mathsf{Pos}\ y$, specifies a pasting diagram of type $\mathcal{P}\ (\mathsf{Typ}\ y\ p \triangleleft z\ p)$. Consider the node k for example, a pasting diagram of the appropriate type for this position would be a pasting diagram having the pasting diagram made of the nodes i and j for source and the opetope k for target.

We are now ready to define pasting diagrams of opetopes.

Definition 1.3.2 (Pasting diagrams)**.** Pasting diagrams are inductively defined as the type family

$$\mathcal{P} : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n)\ \to \mathcal{U}$$

whose constructors are:

$$\begin{aligned}
&\mathsf{ob\text{-}pd} : \mathcal{P}\ \mathsf{ob} \\
&\mathsf{lf} \quad\ : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n) \to \mathcal{P}\ (o \triangleleft \eta\ o) \\
&\mathsf{nd} \quad\ : \{n : \mathbb{N}\}\ (x : \mathcal{O}\ n)\ (y : \mathcal{P}\ x)\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\
&\qquad \to \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z) \to \mathcal{P}\ (x \triangleleft \mu\ y\ z)
\end{aligned}$$

This definition involves the operations η and μ that will be explained later. We comment the role of each constructor. The constructor $\mathsf{ob\text{-}pd}$ corresponds to the unique pasting diagram made of the unique opetope ob of dimension 0. The leaf constructor $\mathsf{lf}\ o$ is the empty pasting diagram made of a single edge whose source and target are both the opetope o. Finally, $\mathsf{nd}\ x\ y\ t$ is the pasting diagram obtained from grafting the family of pasting diagrams t indexed by the positions of y on the sources of the opetope $x \triangleleft y$. We will write $\mathsf{nd}\ (x \triangleleft y)\ t$ instead of $\mathsf{nd}\ x\ y\ t$ to stress that the root node of this pasting diagram is the opetope $x \triangleleft y$.

1.3.3 Definition of positions

For any pasting diagram $t : \mathcal{P}\ o$, the type $\mathsf{Pos}\ t$ is the type of *paths* in t from its root to its different nodes.

Definition 1.3.3 (Positions of a pasting diagram)**.** The family of node positions of a pasting diagram

$$\mathsf{Pos} : \{n : \mathbb{N}\}\ \{o : \mathcal{O}\ n\}\ (t : \mathcal{P}\ o) \to \mathcal{U}$$

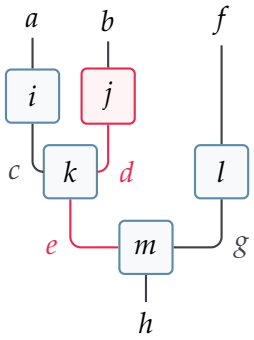
is defined by recursion on t:

$$\begin{aligned}
&\mathsf{Pos}\ \mathsf{ob\text{-}pd} &&:\equiv \top \\
&\mathsf{Pos}\ (\mathsf{lf}\ o) &&:\equiv \bot \\
&\mathsf{Pos}\ (\mathsf{nd}\ (x \triangleleft y)\ t) &&:\equiv \top + \textstyle\sum_{(p : \mathsf{Pos}\ y)} \mathsf{Pos}\ (t\ p)
\end{aligned}$$

To summarise, the unique pasting diagram made of the object has a single node. Leaves have no nodes therefore they have no node positions. Finally, a

node position of a tree built from the nd constructor is either the position of its root node or the position of one of the nodes specified by the family of trees grafted on the root node.

For example, the position of the node j in the following tree which has been highlighted in red is denoted by the term $\mathsf{inr}\ (p_e, \mathsf{inr}\ (p_d, \mathsf{inl}\ \star))$ where p_e is the position of the source of m corresponding to the edge e and where p_d is the position of the source of k corresponding to the edge d.



We are finally able to define the typing function Typ which simply projects out the opetope at a specified position of a pasting diagram.

Definition 1.3.4 (Typing of positions of a pasting diagram)**.** The function giving the typing of positions of a pasting diagram

$$\mathsf{Typ} : \{n : \mathbb{N}\}\ \{o : \mathcal{O}\ n\}\ (p : \mathcal{P}\ o) \to \mathsf{Pos}\ t \to \mathcal{O}\ n$$

is defined by induction on pasting diagrams and on their positions.

$$\begin{aligned}
&\mathsf{Typ}\ \mathsf{ob\text{-}pd}\ \star && :\equiv \mathsf{ob}\\
&\mathsf{Typ}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ (\mathsf{inl}\ \star) && :\equiv (x \triangleleft y)\\
&\mathsf{Typ}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ (\mathsf{inr}\ (p, q)) && :\equiv \mathsf{Typ}\ (t\ p)\ q
\end{aligned}$$

1.3.4 Inductive representation of opetopes

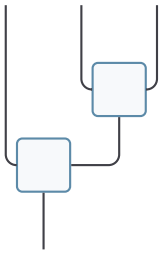
We now clarify the link between our type-theoretical definition of opetopes and their diagrammatic representation so that we can give some examples. We describe a depiction of opetopes by induction on opetopes and on their pasting diagrams.

We begin with the representation of pasting diagram by enumerating the three different possible cases.

- The object pasting diagram ob-pd is the unique pasting diagram composed of the object ob. We depict it as a box:

- Leaves $\mathsf{lf}\ o$ are depicted as a single edge:

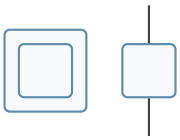
- Nodes $\mathsf{nd}\ (x \triangleleft y)\ t$ are depicted as trees that we recursively draw as follows. First, we draw a node standing for the root node $x \triangleleft y$. Then, we recursively graft the trees $t\ p$ on the root node for each position $p : \mathsf{Pos}\ y$ using the present algorithm. The following figure illustrates an example of this case with a root node and two recursively grafted trees on its two positions: a leaf and a single node.



Opetopes We depict opetopes as a sequence of diagrams by recursion on their dimension.

- The first base case is the object ob; it is depicted as a box.

- The second base case is the arrow $\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd}$. We first draw the object ob then we draw the pasting diagram $\mathsf{ob\text{-}pd}$ in the object resulting in two stacked boxes. Finally, we add a corolla with a single source corresponding to the unique position of $\mathsf{ob\text{-}pd}$.



- The inductive case corresponds to opetopes of shape $x \triangleleft y \triangleleft z$ where z is a pasting diagram parallel to the opetope $x \triangleleft y$. The induction hypothesis tells us how to depict $x \triangleleft y$. The last diagram of the depiction of $x \triangleleft y$ is a corolla. As z is parallel to $x \triangleleft y$, we can depict it *inside* that corolla as it shares its sources with the opetope it is parallel to. Then, we add a corolla to the sequence whose sources are in correspondence with the positions

of the pasting diagram z. Now, an alteration of the diagram representing y has to be performed. We start by erasing the tree while preserving its enclosing corolla then we distinguish two cases:

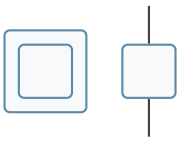
- If z is of the form $\mathsf{lf}\ o$ then we are done.
- If z is of the form $\mathsf{nd}\ (a \triangleleft b)\ t$, we draw the pasting diagram b in the empty corolla. Finally, we recursively fill each node of b at position p according to the data of $t\ p$ by considering the two cases presently listed.

The resulting tree is exactly y, but the diagram contains additional boxes forming a subdivision revealing where the successive substitutions took place according to the data of z.

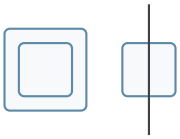
We illustrate this last case with a few examples. Consider the loop

$$\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd} \triangleleft \mathsf{lf}\ o$$

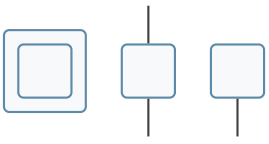
We start with the representation of the arrow $\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd}$.



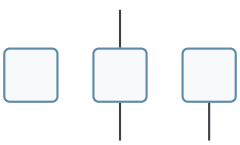
We add the leaf to the last corolla.



We then add a corolla with no source as leaves have no nodes.



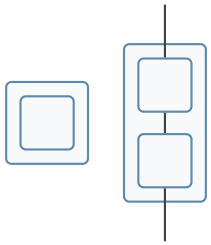
Finally, we alter the first diagram by removing its pasting diagram. We then reach the lf case of our algorithm and we stop.



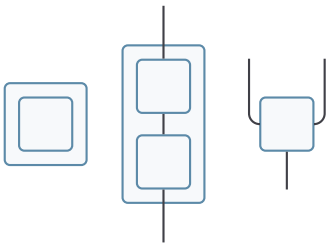
We consider another example: the 2-simplex $\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd} \triangleleft t$ where

$$t :\equiv \mathsf{nd}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ (\lambda p \longrightarrow \mathsf{nd}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ (\lambda q \longrightarrow \mathsf{lf}\ (\mathsf{Typ}\ \mathsf{ob\text{-}pd}\ q)))$$

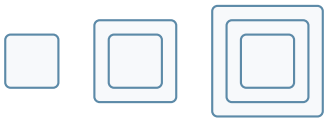
We start with the representation of the arrow $\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd}$ and we draw t in its last diagram which gives us:



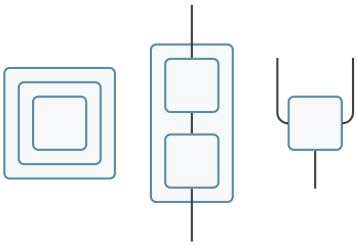
Then, we add a corolla whose sources are in correspondence with the nodes of t.



We finally have to alter the first diagram according to the data specified by t. We start by erasing $\mathsf{ob\text{-}pd}$ and keeping its enclosing box. The pasting diagram t being equal to $\mathsf{nd}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ t'$ for some tree t', we draw $\mathsf{ob\text{-}pd}$ in the empty box. Now, for each position of $p : \mathsf{Pos}\ \mathsf{ob\text{-}pd}$, we recursively draw the appropriate trees according to the data of $t'\ p$ by following the steps of our algorithm. There is only one such position which leads us to first add $\mathsf{ob\text{-}pd}$ again then we reach the lf case and we are done. We depict the changes of this first diagram over time:



The resulting diagram is therefore:



1.3.5 Informal presentation of the monad structure

We discuss the informal specification of the operations of Figure 1.2, and illustrate them in order that the reader understand their role in the definition of pasting diagrams. We will only define them after having introduced some further operations corresponding to the fact that the monad structure is cartesian.

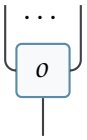
Note that opetopes being entirely determined by the last pasting diagram in their sequence, operations on pasting diagrams can be regarded as operations on opetopes too. We will stress this matter by illustrating the two perspectives using our graphical depiction of trees and opetopes.

The operation η

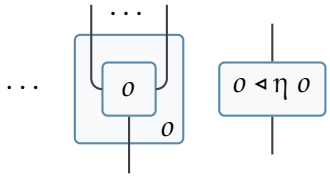
The operation η takes an opetope o and returns the tree whose sole node is the opetope o. Its signature is

$$\eta : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n) \to \mathcal{P}\ o$$

We display $\eta\ o$ as a corolla, just like the top face of the opetope o itself.



The pasting diagram $\eta\ o$ determines the opetope $o \triangleleft \eta\ o$.



Here we chose to label the faces with the opetopes they correspond to. Also, we have only drawn the last two diagrams of the opetope, omitting the first ones.

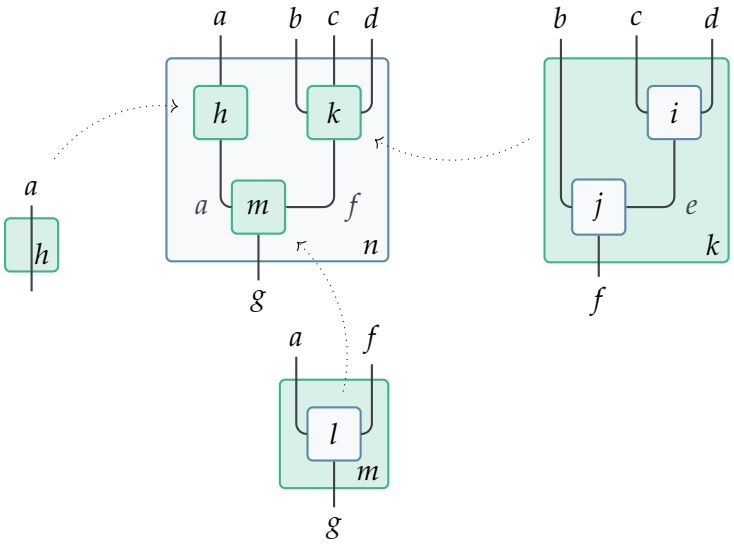
The operation μ

The operation μ substitutes a family of pasting diagrams for the nodes of another pasting diagram. It is associative with unit η, but we delay the precise statement of the laws as they require the introduction of some further operations. We recall its signature

$$\mu : \{n : \mathbb{N}\}\ \{o : \mathcal{O}\ n\}\ (t : \mathcal{P}\ o) \to \overrightarrow{\mathcal{P}}\ t \to \mathcal{P}\ o$$

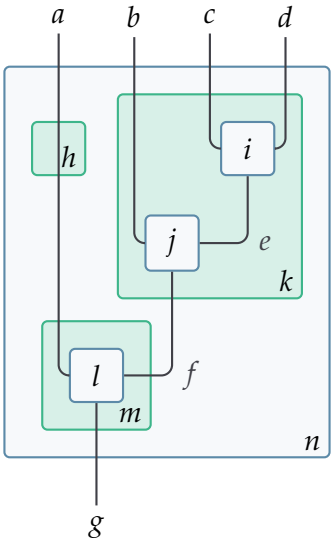
The arguments of the operation μ comprise a pasting diagram $t : \mathcal{P}\ o$ and a family of pasting diagrams $u : \overrightarrow{\mathcal{P}}\ t$ indexed by the positions of t. An example of

such data is depicted in the figure



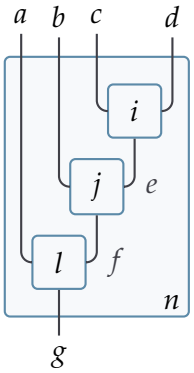
where we suppose that n is the top face of the opetope o, where t is the tree enclosed in the face labelled n, and where the family u is represented as the three trees indexed by the opetopes h, k, and m corresponding to the three nodes of t. Here, we have chosen to draw each tree along with their indexing opetope enclosing them.

Another way to depict this data is to draw the trees specified by u in the nodes of t:



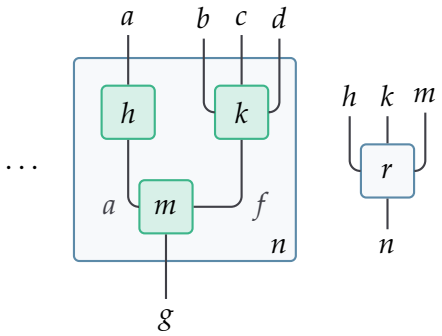
The operation μ then returns the pasting diagram resulting from the substitution

of the trees specified by u for the nodes of t:

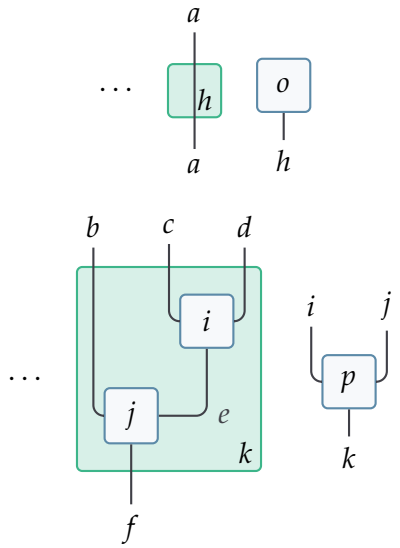


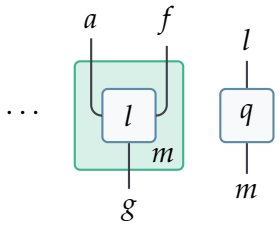
Note how the pasting diagram t and the resulting pasting diagram share the same indexing opetope n.

If, instead, we consider the opetopes determined by t and u, the arguments of μ, we can regard them as specifying a depth-2 tree of opetopes whose root opetope is $o \triangleleft t$ and whose opetopes grafted on $o \triangleleft t$ at position p are $\mathsf{Typ}\ t\ p \triangleleft u\ p$. Illustrating this perspective with our running example, the opetope $o \triangleleft t$ is the following one:

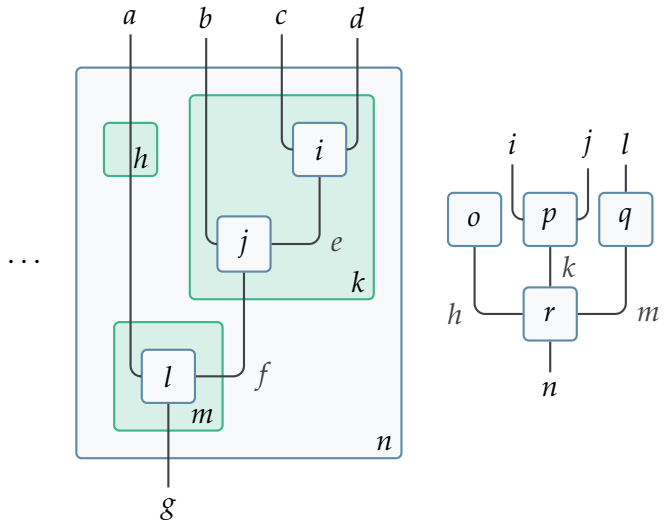


As for the opetopes specified by the family u, they are the following ones:

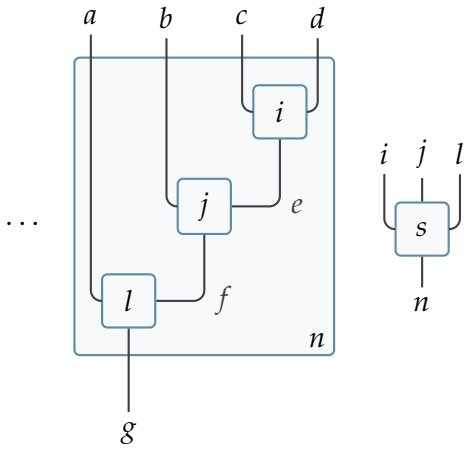




These opetopes assemble into the following depth-2 tree depicted on the right:

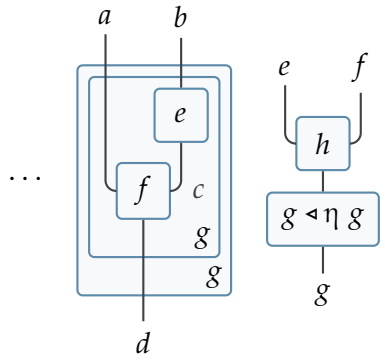


The operation μ then contracts this pasting diagram of opetopes into the following opetope:

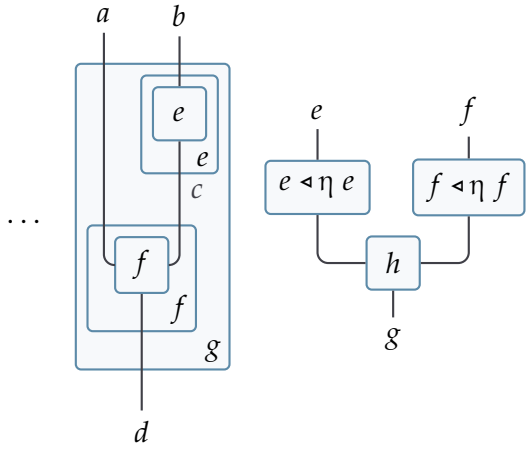


This operation preserves the sources and the target. Finally, the operation μ is associative and unital with units given by η. Being unital means that substituting the nodes of a tree with corollas made of these same nodes does nothing. Being associative means that the order in which we apply different substitutions does not matter. We will only illustrate the left unitality and the right unitality of μ respectively.

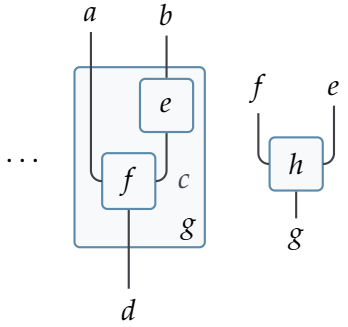
For instance, both this pasting diagram



and this pasting diagram



are sent to the following pasting diagram under the operation μ:



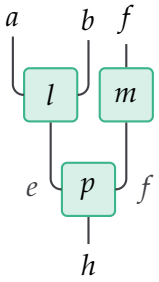
The operation γ

The definition of the operation μ will involve a further operation:

$$\begin{aligned}\gamma : &\{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\}\\ &\to (t : \mathcal{P}\ (x \triangleleft y))\ (u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z))\\ &\to \mathcal{P}\ (x \triangleleft \mu\ y\ z)\end{aligned}$$

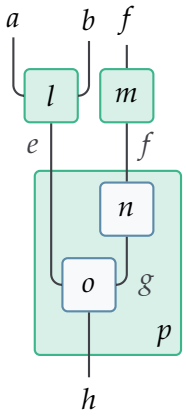
The operation γ takes a tree t and a family of trees u whose targets match the sources of the former and grafts the latter on the leaves of the former.

We consider the opetope $x \triangleleft y$ and the family of opetopes $\mathsf{Typ}\ y\ r \triangleleft z\ r$ for any position $r : \mathsf{Pos}\ y$ as specifying a depth-2 tree of opetopes constituting the indexing data for the trees we want to graft. We illustrate the action of γ on an example and represent the depth-2 tree as follows:



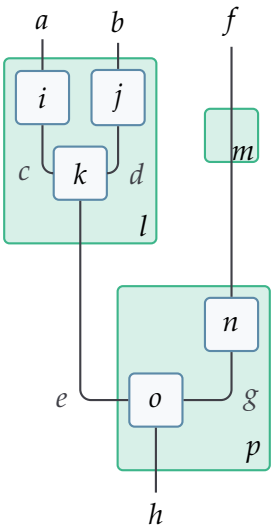
We suppose that p is the top face of the opetope $x \triangleleft y$ and that the two other nodes are the top faces of the opetopes $\mathsf{Typ}\ y\ r \triangleleft z\ r$ for each position r.

Second, the tree t on which the grafting will take place is indexed by the opetope $x \triangleleft y$. Therefore, we depict this tree in its indexing opetope:

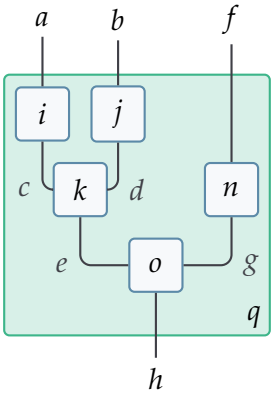


Next, the family u specifies a family of trees to graft on the leaves of t indexed by the opetopes specified by $\mathsf{Typ}\ y\ r \triangleleft z\ r$ for any position r of y. We therefore

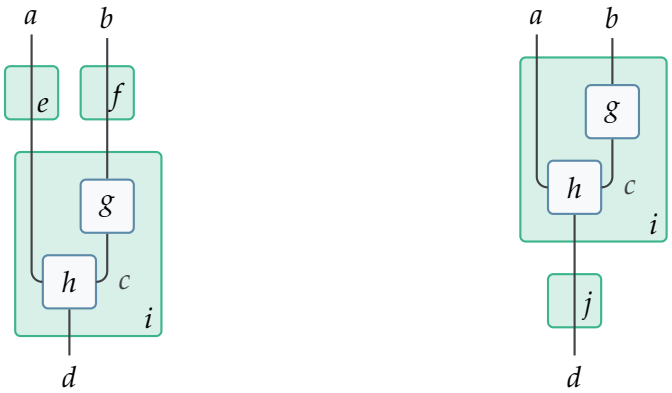
complete our example with this new data:



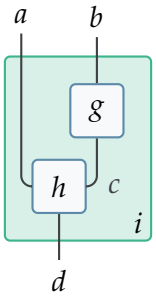
Finally, the operation γ defines a tree resulting from the grafting of the family u on the tree t. The opetope indexing this new tree is then obtained by taking the image of the depth-2 indexing tree — in green — under the operation μ, hence the type of the resulting tree $\mathcal{P}\ (x \triangleleft \mu\ y\ z)$:



As we will see, the operation γ satisfies some laws. It is unital and its units are leaves indexed by opetopes of the form $o \triangleleft \eta\ o$. Consider the following two situations:



The operation γ being right and left unital, the image under γ of these two configurations is therefore



Note that we used the fact that the operation μ is itself left and right unital as the opetopes indexing the leaves are of the form $o \triangleleft \eta\ o$. Finally, the operation γ is also associative in that the order in which we perform multiple graftings does not matter.

1.3.6 Specification of positions operations

We will now introduce some operations witnessing that positions along with their typing information are compatible with the algebraic structure of pasting diagrams. This is due to the fact that we are defining cartesian monads. Once again we will only present the signatures of these operations along with their laws and delay their definitions until later.

Operations relative to $\mathsf{Pos}\ (\eta\ x)$

Given an opetope x, the pasting diagram $\eta\ x$ has a single node which is x itself. Its type of positions, $\mathsf{Pos}\ (\eta\ x)$, should therefore be equivalent to the unit type. This leads us to state the following operations reminiscent of the introduction and elimination rules for the unit type:

$$\mathsf{pos}^{\eta} : \{n : \mathbb{N}\}\ (x : \mathcal{O}\ n) \to \mathsf{Pos}\ (\eta\ x)$$
$$\mathsf{Pos\text{-}\eta\text{-}elim} : \{n : \mathbb{N}\}\ (x : \mathcal{O}\ n)\ (A : \mathsf{Pos}\ (\eta\ x) \to \mathcal{U})$$
$$\to (a_{\mathsf{pos}^{\eta}} : A\ (\mathsf{pos}^{\eta}\ x))\ (p : \mathsf{Pos}\ (\eta\ x)) \to A\ p$$

The position $\mathsf{pos}^\eta\ x$ is the unique position of the pasting diagram composed of the single opetope x. As for the elimination rule for $\mathsf{Pos}\ (\eta\ x)$, it states that $\mathsf{pos}^\eta\ x$ is its unique element. The elimination rule applied to the introduction rule satisfies the following identity:

$$\mathsf{Pos\text{-}\eta\text{-}elim}\ x\ A\ a_{\mathsf{pos}^\eta}\ (\mathsf{pos}^\eta\ x) = a_{\mathsf{pos}^\eta} \qquad (\mathsf{Pos}\text{-}\eta\text{-}\mathsf{elim}\text{-}\beta)$$

Finally, the typing information is compatible with η and it is the case that $\mathsf{Typ}\ (\eta\ x)\ (\mathsf{pos}^\eta\ x) = x$. As the type of positions of $\eta\ x$ is contractible, we have the more general rule:

$$\mathsf{Typ}\ (\eta\ x)\ p = x \qquad (\mathsf{pos}^\eta\text{-}\mathsf{typ})$$

Operations relative to $\mathsf{Pos}\ (\mu\ x\ y)$

The positions operations relative to trees of the form $\mu\ x\ y$ implement the cartesian structure of our monads. The operation μ substitutes compatible trees specified by a family of trees y for the nodes of a given tree x. The type of positions of the resulting tree $\mu\ x\ y$ should therefore be equivalent to the sum of the positions of the trees specified by y. Therefore, we want the positions of a tree of the form $\mu\ x\ y$ to be characterised by the following requirements:

1. Any position $p : \mathsf{Pos}\ x$ along with a position $q : \mathsf{Pos}\ (y\ p)$ should determine a unique position of $\mu\ x\ y$.
2. Any position of $\mu\ x\ y$ should correspond to a position $p : \mathsf{Pos}\ x$ along with a position $\mathsf{Pos}\ (y\ p)$.
3. Moreover, the induced functions should be inverse to each other.

These desiderata are captured by the following operations:

$$\begin{aligned}
&\mathsf{pair}^\mu : \{n : \mathbb{N}\}\ \{x : O\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\
&\quad \to (p : \mathsf{Pos}\ y)\ (q : \mathsf{Pos}\ (z\ p)) \\
&\quad \to \mathsf{Pos}\ (\mu\ y\ z) \\
&\mathsf{pr}_1^\mu : \{n : \mathbb{N}\}\ \{x : O\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\
&\quad \to \mathsf{Pos}\ (\mu\ y\ z) \to \mathsf{Pos}\ y \\
&\mathsf{pr}_2^\mu : \{n : \mathbb{N}\}\ \{x : O\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\
&\quad \to (p : \mathsf{Pos}\ (\mu\ y\ z)) \to \mathsf{Pos}\ (z\ (\mathsf{pr}_1^\mu\ p))
\end{aligned}$$

Moreover, these operations obey the following laws:

$$\mathsf{pair}^\mu\ (\mathsf{pr}_1^\mu\ p)\ (\mathsf{pr}_2^\mu\ p) = p \qquad (\mathsf{pair}^\mu\text{-}\eta)$$

$$\mathsf{pr}_1^\mu\ (\mathsf{pair}^\mu\ p\ q) = p \qquad (\mathsf{pr}_1^\mu\text{-}\beta)$$

$$\mathsf{transport}^{\lambda p \to \mathsf{Pos}\ (z\ p)}\ \mathsf{pr}_1^\mu\text{-}\beta\ (\mathsf{pr}_2^\mu\ (\mathsf{pair}^\mu\ p\ q)) = q \qquad (\mathsf{pr}_2^\mu\text{-}\beta)$$

where we refer to $\mathsf{pr}_1^\mu\text{-}\beta$ in the last equation without making explicit how we instance it, a practice we will adopt throughout this thesis.

Note how these operations along with their laws mirror the introduction and elimination rules of sigma types to the difference that the laws only hold propositionally. We have chosen to grey the transport out in order not to obfuscate the meaning of the law. We also state how the typing operation behave with regard to the substitution:

$$\mathsf{Typ}\ (\mu\ x\ y)\ p = \mathsf{Typ}\ (y\ (\mathsf{pr}_1^\mu\ p))\ (\mathsf{pr}_2^\mu\ p) \qquad (\mu\text{-pos-typ})$$

In other words, the correspondence between positions under the substitution operation extends to their typing information. In future proofs we will often need the following identity that can be deduced from μ-pos-typ, pr_1^μ-β, and pr_2^μ-β:

$$\mathsf{Typ}\ (\mu\ x\ y)\ (\mathsf{pair}^\mu\ p\ q) = \mathsf{Typ}\ (y\ p)\ q \qquad (\mu\text{-pos-typ-aux})$$

Operations relative to $\mathsf{Pos}\ (\gamma\ t\ u)$

The operation γ grafts a family of trees $u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)$ on the leaves of a given tree $t : \mathcal{P}\ (x \triangleleft y)$. The type of positions of the resulting tree should be equivalent to the sum of the positions of t with the sum of the positions of the trees specified by the family u. Therefore, we want the positions of a tree of the form $\gamma\ t\ u$ to be characterised by the following requirements:

1. For any position $p : \mathsf{Pos}\ t$, we can obtain a position of $\gamma\ t\ u$.
2. For any position $p : \mathsf{Pos}\ y$ along with a position $q : \mathsf{Pos}\ (u\ p)$, we can obtain a position of $\gamma\ t\ u$.
3. For any position $p : \mathsf{Pos}\ (\gamma\ t\ u)$, we can either deduce a position of t or we can deduce a position of $u\ p$ for some position $p : \mathsf{Pos}\ y$.

Formally, these desiderata are captured by the following operations:

$$\begin{aligned}
&\mathsf{inl}^\gamma : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\
&\quad \to \{t : \mathcal{P}\ (x \triangleleft y)\}\ \{u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)\} \\
&\quad \to \mathsf{Pos}\ t \to \mathsf{Pos}\ (\gamma\ t\ u) \\
&\mathsf{inr}^\gamma : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\
&\quad \to \{t : \mathcal{P}\ (x \triangleleft y)\}\ \{u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)\} \\
&\quad \to (p : \mathsf{Pos}\ y) \to \mathsf{Pos}\ (u\ p) \to \mathsf{Pos}\ (\gamma\ t\ u)
\end{aligned}$$

A further function, akin to an elimination rule, states that the only positions of $\gamma\ t\ u$ are obtained with the operations inl^γ and inr^γ:

$$\begin{aligned}
&\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim} : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\
&\quad \to \{t : \mathcal{P}\ (x \triangleleft y)\}\ \{u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)\} \\
&\quad \to (A : \mathsf{Pos}\ (\gamma\ t\ u) \to \mathcal{U}) \\
&\quad \to (a_{\mathsf{inl}^\gamma} : (p : \mathsf{Pos}\ t) \to A\ (\mathsf{inl}^\gamma\ p)) \\
&\quad \to (a_{\mathsf{inr}^\gamma} : (p : \mathsf{Pos}\ y)\ (q : \mathsf{Pos}\ (u\ p)) \to A\ (\mathsf{inr}^\gamma\ p\ q)) \\
&\quad \to (p : \mathsf{Pos}\ (\gamma\ t\ u)) \to A\ p
\end{aligned}$$

Moreover, it satisfies the following laws:

$$\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim}\ A\ a_{\mathsf{inl}}\ a_{\mathsf{inr}}\ (\mathsf{inl}^{\gamma}\ p) = a_{\mathsf{inl}}\ p \qquad (\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim}\text{-}\mathsf{inl}\text{-}\beta)$$
$$\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim}\ A\ a_{\mathsf{inl}}\ a_{\mathsf{inr}}\ (\mathsf{inr}^{\gamma}\ p\ q) = a_{\mathsf{inr}}\ p\ q \qquad (\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim}\text{-}\mathsf{inr}\text{-}\beta)$$

Finally, the correspondence between positions extends to the typing information:

$$\mathsf{Typ}\ (\gamma\ t\ u)\ (\mathsf{inl}^{\gamma}\ p) = \mathsf{Typ}\ t\ p$$
$$\mathsf{Typ}\ (\gamma\ t\ u)\ (\mathsf{inr}^{\gamma}\ p\ q) = \mathsf{Typ}\ (u\ p)\ q$$

1.3.7 Definition of the monad structure

At last, we can define the monad structure. We start with the unit of the substitution operation.

Definition 1.3.5 (The unit η)**.** The unit η has type

$$\eta : \{n : \mathbb{N}\}\ (x : \mathcal{O}\ n) \to \mathcal{P}\ x$$

It is defined by induction on x.

- If x is of the form ob, we simply return $\mathsf{ob}\text{-}\mathsf{pd} : \mathcal{P}\ \mathsf{ob}$, the object pasting diagram.
- If x is of the form $x \triangleleft y$, we need to return a tree of type $\mathcal{P}\ (x \triangleleft y)$. We denote
$$t :\equiv \mathsf{nd}\ (x \triangleleft y)\ (\lambda p \to \mathsf{lf}\ (\mathsf{Typ}\ y\ p))$$
the tree with opetope $x \triangleleft y$ for sole node. It has type
$$\mathcal{P}\ (x \triangleleft \mu\ y\ (\lambda p \to \eta\ (\mathsf{Typ}\ y\ p)))$$
therefore we transport it along a path
$$\mu\ y\ (\lambda p \to \eta\ (\mathsf{Typ}\ y\ p)) = y$$
which holds by μ-unit-r.

We then define the substitution operation.

Definition 1.3.6 (The substitution operation μ)**.** The operation μ has type

$$\mu : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ (y : \mathcal{P}\ x)\ (z : \overrightarrow{\mathcal{P}}\ y) \to \mathcal{P}\ x$$

The operation μ substitutes the opetopes of a pasting diagram y — its nodes — with compatible pasting diagrams provided by the family z. It is defined by induction on y:

- If y is $\mathsf{ob}\text{-}\mathsf{pd}$, we substitute the tree $z\ \star$ for the only position of $\mathsf{ob}\text{-}\mathsf{pd}$ therefore we simply return $z\ \star$.

- If y is of the form $\mathsf{lf}\ x$, leaves have no node therefore there is nothing to substitute, and we just return the tree $\mathsf{lf}\ x$ intact.

- If y is of the form $\mathsf{nd}\ (x \triangleleft y)\ u$ with $u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft t)$ for some family $t : \overrightarrow{\mathcal{P}}\ y$, we have to define a tree of type $\mathcal{P}\ (x \triangleleft \mu\ y\ t)$ which should be the tree $\mathsf{nd}\ (x \triangleleft y)\ u$ whose nodes have been recursively substituted with compatible trees specified by the family z. Therefore, the resulting tree should be the result of the grafting of the family

$$v\ p :\equiv \mu\ (u\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q)))$$

— which is our induction hypothesis — on the tree $z\ (\mathsf{inl}\ \star)$; that is, the tree

$$\gamma\ (z\ (\mathsf{inl}\star))\ v$$

We finally state the laws obeyed by the operation μ which is an associative and unital operation with unit η,

$$\mu\ y\ (\lambda\ p \to \eta\ (\mathsf{Typ}\ y\ p)) = y \qquad (\mu\text{-unit-r})$$

$$\mathsf{transport}^{\mathcal{P}}\ \mathsf{pos}^{\eta}\text{-}\mathsf{typ}^{-1}\ (\mu\ (\eta\ x)\ y) = y\ (\mathsf{pos}^{\eta}\ x) \qquad (\mu\text{-unit-l})$$

$$\mu\ (\mu\ y\ z)\ w = \mu\ y\ (\lambda\ p \to \mu\ (z\ p)\ (w'\ p)) \qquad (\mu\text{-assoc})$$

$$\text{where } w'\ p\ q :\equiv \mathsf{transport}^{\mathcal{P}}\ \mu\text{-pos-typ-aux}\ (w\ (\mathsf{pair}^{\mu}\ p\ q))$$

We define the grafting operation γ.

Definition 1.3.7 (The grafting operation γ)**.** The operation γ has type

$$\begin{aligned}\gamma : &\{n : \mathbb{N}\}\ \{x : O\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\ &\to (t : \mathcal{P}\ (x \triangleleft y))\ (u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)) \\ &\to \mathcal{P}\ (x \triangleleft \mu\ y\ z)\end{aligned}$$

The operation γ grafts the family of trees u on the leaves of the tree t which are determined by the positions of the tree y. This operation is defined inductively on t.

- In the case $\gamma\ (\mathsf{lf}\ x)\ t$, where $t : \overrightarrow{\mathcal{P}}\ (\eta\ x \blacktriangleleft y)$, the result of grafting the sole tree specified by t at position $\mathsf{pos}^{\eta}\ x$ on the leaf $\mathsf{lf}\ x$ should precisely be the tree $t\ (\mathsf{pos}^{\eta}\ x)$. We conclude the proof by transporting it along a path

$$\mathcal{P}\ (\mathsf{Typ}\ (\eta\ x)\ (\mathsf{pos}^{\eta}\ x) \triangleleft y\ (\mathsf{pos}^{\eta}\ x)) = \mathcal{P}\ (x \triangleleft \mu\ (\eta\ x)\ y)$$

deduced from $\mathsf{pos}^{\eta}\text{-}\mathsf{typ}$ and μ-unit-l:

$$\mathsf{transport}^{\lambda(x,y)\to\mathcal{P}\ (x\triangleleft y)}\ (\mathsf{pair}{=}\ \mathsf{pos}^{\eta}\text{-}\mathsf{typ}^{-1}\ \mu\text{-unit-l})^{-1}\ (t\ (\mathsf{pos}^{\eta}\ x))$$

- In the case $\gamma\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ u$, with $t : \mathcal{P}\ (y \triangleleft z)$ and $u : \mathcal{P}\ (\mu\ y\ z \blacktriangleleft w)$, the root node of the resulting tree will still be $x \triangleleft y$. We are therefore looking for some tree $\mathsf{nd}\ (x \triangleleft y)\ v$ where v is a family resulting from the grafting of the family u on the family t.

  In order to graft on the family t, we need a family of families $\overrightarrow{\mathcal{P}}\ (t\ p \blacktriangleleft w'\ p)$ indexed by positions $p : \mathsf{Pos}\ y$ where the family w' has yet to be defined. Such a family of families can be defined from u as

  $$u'\ p\ q :\equiv \mathsf{transport}^{\lambda(x,y)\to\mathcal{P}\ (x\triangleleft y)}\ (\mathsf{pair{=}}\ \mu\text{-pos-typ-aux refl})\ (u\ (\mathsf{pair}^{\mu}\ p\ q))$$

  where $u\ (\mathsf{pair}^{\mu}\ p\ q)$ is transported along a path

  $$\mathcal{P}\ (\mathsf{Typ}\ (\mu\ y\ z)\ (\mathsf{pair}^{\mu}\ p\ q) \triangleleft w\ (\mathsf{pair}^{\mu}\ p\ q)) = \mathcal{P}\ (\mathsf{Typ}\ (z\ p)\ q \triangleleft w'\ p\ q)$$

  using μ-pos-typ-aux. This forces the definition of w' to be

  $$w'\ p\ q :\equiv \mathsf{transport}^{\mathcal{P}}\ \mu\text{-pos-typ-aux}\ (w\ (\mathsf{pair}^{\mu}\ p\ q))$$

  We therefore obtain the family

  $$v : \overrightarrow{\mathcal{P}}(y \blacktriangleleft (\lambda p \to \mu\ (z\ p)\ (w'\ p)))$$
  $$v\ p :\equiv \gamma\ (t\ p)\ (u'\ p)$$

  The resulting tree $\mathsf{nd}\ (x \triangleleft y)\ v$ has type $\mathcal{P}\ (x \triangleleft \mu\ y\ (\lambda p \to \mu\ (z\ p)\ (w'\ p)))$ where we need it to have type $\mathcal{P}\ (x \triangleleft \mu\ (\mu\ y\ z)\ w)$. We therefore conclude the proof by performing a last transport along a proof of associativity of μ

  $$\mathsf{transport}^{\lambda y\to\mathcal{P}\ (x\triangleleft y)}\ \mu\text{-assoc}^{-1}\ (\mathsf{nd}\ (x \triangleleft y)\ v)$$

Finally, we state the laws obeyed by the operation γ which is associative and unital with unit lf,

$\mathsf{transport}^{\lambda y\to\mathcal{P}\ (x\triangleleft y)}\ \mu\text{-unit-r}$ (γ-unit-r)
$(\gamma\ t\ (\lambda p \to \mathsf{lf}\ (\mathsf{Typ}\ y\ p))) = t$
for all $t : \mathcal{P}\ (x \triangleleft y)$

$\mathsf{transport}^{\lambda(x,y)\to\mathcal{P}\ (x\triangleleft y)}\ (\mathsf{pair{=}}\ \mathsf{pos}^{\eta}\text{-}\mathsf{typ}^{-1}\ \mu\text{-unit-l})$ (γ-unit-l)
$(\gamma\ (\mathsf{lf}\ x)\ t) = t\ (\mathsf{pos}^{\eta}\ x)$
for all $t : \overrightarrow{\mathcal{P}}\ (\eta\ x \blacktriangleleft y)$

$\mathsf{transport}^{\lambda y\to\mathcal{P}\ (x\triangleleft y)}\ \mu\text{-assoc}$ (γ-assoc)
$(\gamma\ (\gamma\ t\ u)\ v) = \gamma\ t\ (\lambda p \to \gamma\ (u\ p)\ (v'\ p))$
for all $t : \mathcal{P}\ (x \triangleleft y), u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z), v : \overrightarrow{\mathcal{P}}\ (\mu\ y\ z \blacktriangleleft w)$, and
where $v'\ p\ q :\equiv \mathsf{transport}^{\lambda(x,y)\to\mathcal{P}\ (x\triangleleft y)}\ (\mathsf{pair{=}}\ \mu\text{-pos-typ-aux refl})\ (v\ (\mathsf{pair}^{\mu}\ p\ q))$

1.3.8 Definition of the positions operations

Now that we have defined the different operations relative to pasting diagrams as well as the type of their positions, we can define the operations on their positions. We start with two simple lemmas that will be needed throughout this chapter and which essentially state that positions of trees are invariable under transport. For any tree $t : \mathcal{P}\ x$ and any identity $p : x = y$,

$$\mathsf{Pos}\ t = \mathsf{Pos}\ (\mathsf{transport}^{\mathcal{P}}\ p\ t) \qquad (\mathsf{Pos\text{-}transp}^{\mathcal{O}})$$

For any tree $t : \mathcal{P}\ (x \triangleleft y)$ and any identity $p : y = z$,

$$\mathsf{Pos}\ t = \mathsf{Pos}\ (\mathsf{transport}^{\lambda y \to \mathcal{P}\ (x \triangleleft y)}\ p\ t) \qquad (\mathsf{Pos\text{-}transp}^{\mathcal{P}})$$

They are both proven by a simple induction on p.

We also define two functions akin to elimination rules for types of the form $\mathsf{Pos}\ (\mathsf{transport}^{\lambda y \to \mathcal{P}\ (x \triangleleft y)}\ p\ t)$ and $\mathsf{Pos}\ (\mathsf{transport}^{\lambda (x,y) \to \mathcal{P}\ (x \triangleleft y)}\ p\ t)$ in order to reduce the need for annoying transports.

$$\begin{aligned}
&\mathsf{Pos\text{-}transp}^{\mathcal{O}}\text{-}\mathsf{elim} : \{n : \mathbb{N}\}\ \{x_0\ x_1 : \mathcal{O}\ n\}\ \{y_0 : \mathcal{P}\ x_0\}\ \{y_1 : \mathcal{P}\ x_1\} \\
&\quad \to (e : (x_0, y_0) = (x_1, y_1)) \\
&\quad \to (t : \mathcal{P}\ (x_0 \triangleleft y_0)) \\
&\quad \to (A : \mathsf{Pos}\ (\mathsf{transport}^{\lambda (x,y) \to \mathcal{P} (x \triangleleft y)}\ e\ t) \to \mathcal{U}) \\
&\quad \to (f : (p : \mathsf{Pos}\ t) \to A\ (\mathsf{transport}\ \mathsf{Pos\text{-}transp}^{\mathcal{O}}\ p)) \\
&\quad \to (p : \mathsf{Pos}\ (\mathsf{transport}^{\lambda (x,y) \to \mathcal{P} (x \triangleleft y)}\ e\ t)) \\
&\quad \to A\ p
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Pos\text{-}transp}^{\mathcal{P}}\text{-}\mathsf{elim} : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y\ z : \mathcal{P}\ x\}\ (e : y = z) \\
&\quad \to (t : \mathcal{P}\ (x \triangleleft y)) \\
&\quad \to (A : \mathsf{Pos}\ (\mathsf{transport}^{\lambda y \to \mathcal{P} (x \triangleleft y)}\ e\ t) \to \mathcal{U}) \\
&\quad \to (f : (p : \mathsf{Pos}\ t) \to A\ (\mathsf{transport}\ \mathsf{Pos\text{-}transp}^{\mathcal{P}}\ p)) \\
&\quad \to (p : \mathsf{Pos}\ (\mathsf{transport}^{\lambda y \to \mathcal{P} (x \triangleleft y)}\ e\ t)) \\
&\quad \to A\ p
\end{aligned}$$

They are both trivially defined by an induction on the path e in which case it suffices to return $f\ p$.

Operations relative to $\mathsf{Pos}\ (\eta\ x)$

We start by showing that the type of positions of $\eta\ x$ is contractible for any opetope $x : \mathcal{O}\ n$ which will allow us to derive the introduction and elimination rules for $\mathsf{Pos}\ (\eta\ x)$.

Lemma 1.3.8. *For any natural number $n : \mathbb{N}$ and any opetope $x : \mathcal{O}\ n$, the type* $\mathsf{Pos}\ (\eta\ x)$ *is contractible.*

Proof. We prove this property by induction on the opetope x. If x is of the form ob, $\eta\ \mathsf{ob}$ is the object pasting diagram $\mathsf{ob\text{-}pd}$ and its type of positions is the unit type $\mathbf{1}$ which is contractible. If x is of the form $x \triangleleft y$, the unit $\eta\ (x \triangleleft y)$ is the tree

$$t :\equiv \mathsf{nd}\ (x \triangleleft y)\ (\lambda\ p \to \mathsf{lf}\ (\mathsf{Typ}\ y\ p))$$

up to a transport along the path μ-unit-r. It is easy to show that the type $\mathsf{Pos}\ t$ is equivalent to $\mathbf{1}$ and is therefore contractible. It suffices to transport this result along a path $\mathsf{Pos}\ t = \mathsf{Pos}\ (\eta\ (x \triangleleft y))$ to conclude that $\mathsf{Pos}\ (\eta\ (x \triangleleft y))$ is contractible as well. □

We use the previous result to define $\mathsf{pos}^{\eta}\ x$ to be the centre of contraction of $\mathsf{Pos}\ (\eta\ x)$. As for the elimination rule, we first recall its signature:

$$\begin{aligned}\mathsf{Pos\text{-}\eta\text{-}elim} : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n)\ (A : \mathsf{Pos}\ (\eta\ o) \to \mathcal{U})\\ \to (a_{\mathsf{pos}^{\eta}} : A\ (\mathsf{pos}^{\eta}\ o))\ (p : \mathsf{Pos}\ (\eta\ o)) \to A\ p\end{aligned}$$

We simply define it by transporting $a_{\mathsf{pos}^{\eta}}$ from $A\ (\mathsf{pos}^{\eta}\ o)$ to $A\ p$ using the path from the centre of contraction of $\mathsf{Pos}\ (\eta\ o)$ to any position $p : \mathsf{Pos}\ (\eta\ o)$. Also, the operation $\mathsf{Pos\text{-}\eta\text{-}elim}$ satisfies the law

$$\mathsf{Pos\text{-}\eta\text{-}elim}\ x\ A\ a_{\mathsf{pos}^{\eta}}\ (\mathsf{pos}^{\eta}\ x) = a_{\mathsf{pos}^{\eta}} \qquad (\eta\text{-pos-elim-}\beta)$$

Indeed, the specified path from the centre of contraction to itself is propositionally equal to refl which is a property of contractible types.

Operations relative to $\mathsf{Pos}\ (\mu\ x\ y)$

We define the three operations pair^{μ}, pr_1^{μ}, and pr_2^{μ} relative to positions of trees of the form $\mu\ x\ y$. The operation pair^{μ} takes a position p of a tree $y : \mathcal{P}\ x$ and a position q of the tree $z\ p$ where $z : \overrightarrow{\mathcal{P}}\ y$ in order to return a position of the resulting tree $\mu\ x\ y$.

Definition 1.3.9 (pair^{μ})**.**

$$\begin{aligned}\mathsf{pair}^{\mu} : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\}\\ \to (p : \mathsf{Pos}\ y)\ (q : \mathsf{Pos}\ (z\ p))\\ \to \mathsf{Pos}\ (\mu\ y\ z)\end{aligned}$$

We define pair^{μ} by induction on the tree y and on the position p:

- If y is $\mathsf{ob\text{-}pd}$, $\mu\ \mathsf{ob\text{-}pd}\ z$ is equal to $z\ \star$, and we are looking for a term of type $\mathsf{Pos}\ (z\ \star)$ but p is of type $\mathsf{Pos\ ob\text{-}pd}$ and must be equal to $\star$. We therefore return q which is of the required type.
- If y is of the form $\mathsf{lf}\ x$, $\mathsf{Pos}\ (\mathsf{lf}\ x)$ is the empty type and we are done.

- If y is of the form $\mathsf{nd}\ (x \triangleleft y)\ u$, $\mu\ (\mathsf{nd}\ (x \triangleleft y)\ u)\ z$ is equal to $\gamma\ (z\ (\mathsf{inl}\ \star))\ v$ where
$$v\ p :\equiv \mu\ (u\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q)))$$
In order to determine a position of $\gamma\ (z\ (\mathsf{inl}\ \star))\ v$, we have to proceed by induction on p:
  - If p is $\mathsf{inl}\ \star$, q is a position of $z\ (\mathsf{inl}\ \star)$ which is the tree on which the grafting takes place. The position that we are looking for is therefore $\mathsf{inl}^\gamma\ q$.
  - If p is $\mathsf{inr}\ (r, s)$ with $r : \mathsf{Pos}\ y$ and $s : \mathsf{Pos}\ (u\ r)$, the position that we are looking for is a position of one of the trees specified by the family v injected into $\mathsf{Pos}\ (\gamma\ (z\ (\mathsf{inl}\ \star))\ v)$. This family v is indexed by $\mathsf{Pos}\ y$ and r is precisely such a position. We now need a position of $v\ r$, that is a position of $\mu\ (u\ r)\ (\lambda q \to z\ (\mathsf{inr}\ (r, q)))$, which is readily obtained as $\mathsf{pair}^\mu\ s\ q$. In the end, the position that we are looking for is $\mathsf{inr}^\gamma\ p\ (\mathsf{pair}^\mu\ s\ q)$.

Conversely, the operation pr_1^μ takes a position of $\mu\ x\ y$ and returns a position of the first tree x.

Definition 1.3.10 (pr_1^μ)**.**
$$\begin{aligned}\mathsf{pr}_1^\mu : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\}\\ \to (p : \mathsf{Pos}\ (\mu\ y\ z)) \to \mathsf{Pos}\ y\end{aligned}$$

The operation pr_1^μ is defined by induction on the tree y:

- If y is $\mathsf{ob\text{-}pd}$, we simply return the only position of $\mathsf{ob\text{-}pd}$, that is $\star$.
- If y is of the form $\mathsf{lf}\ x$, we simply eliminate from the empty type $\mathsf{Pos}\ (\mathsf{lf}\ x)$.
- If y is of the form $\mathsf{nd}\ (x \triangleleft y)\ u$, p is a position of the tree $\gamma\ (z\ (\mathsf{inl}\ \star))\ v$ where
$$v\ p :\equiv \mu\ (u\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q)))$$
We will therefore have to resort to the elimination rule for γ. We are looking for a position of $\mathsf{nd}\ (x \triangleleft y)\ u$ therefore we set the motive to the constant
$$A\ p :\equiv \mathsf{Pos}\ (\mathsf{nd}\ (x \triangleleft y)\ u)$$
and we treat the following two cases:
  - If p is a position of the base tree $z\ (\mathsf{inl}\ \star)$, we return the position of the root node of $\mathsf{nd}\ (x \triangleleft y)\ u$, that is $\mathsf{inl}\ \star$.
  - If p is a position of one of the trees grafted on the base tree, we can break it down into a position $r : \mathsf{Pos}\ y$ and a position $s : \mathsf{Pos}\ (v\ r)$. From s, we get a position of $u\ r$ as the position $\mathsf{pr}_1^\mu\ s$. We finally obtain the wanted position of $\mathsf{nd}\ (x \triangleleft y)\ u$ as the position $\mathsf{inr}\ (r, \mathsf{pr}_1^\mu\ s)$.

Finally, the operation pr_2^μ takes a position of $\mu\ x\ y$ and returns a position of the tree $y\ (\mathsf{pr}_1^\mu\ p)$.

Definition 1.3.11 (pr_2^μ)**.**

$$\begin{aligned}\mathsf{pr}_2^\mu : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\}\\ \to (p : \mathsf{Pos}\ (\mu\ y\ z)) \to \mathsf{Pos}\ (z\ (\mathsf{pr}_1^\mu\ p))\end{aligned}$$

The operation pr_2^μ is defined by induction on the tree y:

- If y is $\mathsf{ob\text{-}pd}$, we need to return a position of type $\mathsf{Pos}\ (z\ (\mathsf{inl}\ \star))$ which is precisely the type of p that we therefore just return.
- If y is of the form $\mathsf{lf}\ x$, we simply eliminate from the empty type $\mathsf{Pos}\ (\mathsf{lf}\ x)$.
- If y is of the form $\mathsf{nd}\ (x \triangleleft y)\ u$, p is a position of the tree $\gamma\ (z\ (\mathsf{inl}\ \star))\ v$ where

  $$v\ p :\equiv \mu\ (u\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q)))$$

  We will therefore have to resort to the elimination rule for γ. We are looking for a position of $z\ (\mathsf{pr}_1^\mu\ p)$ therefore we set the motive to

  $$B\ p :\equiv \mathsf{Pos}\ (z\ (\mathsf{pr}_1^\mu\ p))$$

  It remains to treat the two cases that we are facing:

  – If p is a position of the base tree $z\ (\mathsf{inl}\ \star)$, we have to return a position of type $\mathsf{Pos}\ (z\ (\mathsf{pr}_1^\mu\ (\mathsf{inl}^\gamma\ p)))$ that can be obtained from $p : \mathsf{Pos}\ (z\ (\mathsf{inl}\ \star))$ by transporting it along the path $\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim\text{-}inl}\text{-}\beta^{-1}$ remembering that $\mathsf{pr}_1^\mu\ (\mathsf{inl}^\gamma\ p)$ is defined as an application of $\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim}$ in the present case.

  – If p is a position of one of the trees grafted on the base tree, we can break it down into a position $r : \mathsf{Pos}\ y$ and a position $s : \mathsf{Pos}\ (v\ r)$. We are looking for a position of type $\mathsf{Pos}\ (z\ (\mathsf{pr}_1^\mu\ (\mathsf{inr}^\gamma\ r\ s)))$ that can be obtained from $\mathsf{pr}_2^\mu\ s : \mathsf{Pos}\ (z\ (\mathsf{inr}\ (r, \mathsf{pr}_1^\mu\ s)))$ by transporting it along the path $\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim\text{-}inr}\text{-}\beta^{-1}$ remembering that $\mathsf{pr}_1^\mu\ (\mathsf{inr}^\gamma\ r\ s)$ is defined as an application of $\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim}$ in the present case.

Operations relative to $\mathsf{Pos}\ (\gamma\ t\ u)$

We now define the functions relative to positions of the form $\mathsf{Pos}\ (\gamma\ t\ u)$.

Definition 1.3.12 (inl^γ)**.**

$$\begin{aligned}\mathsf{inl}^\gamma : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\}\\ \to \{t : \mathcal{P}\ (x \triangleleft y)\}\ \{u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)\}\\ \to (p : \mathsf{Pos}\ t) \to \mathsf{Pos}\ (\gamma\ t\ u)\end{aligned}$$

The operation inl^γ is defined by induction on the tree t and on the position p:

- If t is of the form $\mathsf{lf}\ x$ then we eliminate the empty type $\mathsf{Pos}\ (\mathsf{lf}\ x)$.
- If t is of the form $\mathsf{nd}\ (x \triangleleft y)\ t$, we have to determine a position of the tree

$$\mathsf{transport}^{\lambda y \to \mathcal{P}\ (x \triangleleft y)}\ \mu\text{-assoc}^{-1}\ (\mathsf{nd}\ (x \triangleleft y)\ v)$$

  with v defined as

$$v\ p :\equiv \gamma\ (t\ p)\ (u'\ p)$$

  and u' defined as

$$u'\ p\ q :\equiv \mathsf{transport}^{\lambda (x,y) \to \mathcal{P}\ (x \triangleleft y)}\ (\mathsf{pair}{=}\ \mu\text{-pos-typ-aux refl})\ (u\ (\mathsf{pair}^{\mu}\ p\ q))$$

  We proceed by induction on p:

  - If p is $\mathsf{inl}\ \star$; that is, if p is the root node of the base tree then we need to determine the root node of the tree $\mathsf{nd}\ (x \triangleleft y)\ v$ up to the required transport which is

$$\mathsf{transport\ Pos\text{-}transp}^{\mathcal{P}}\ (\mathsf{inl}\ \star)$$

  - If p is of the form $\mathsf{inr}\ (p, q)$ where $p : \mathsf{Pos}\ y$ and $q : \mathsf{Pos}\ (t\ p)$, we use our induction hypothesis $\mathsf{inl}^{\gamma}\ q$ to obtain a position of $v\ p$, and we deduce that $\mathsf{inr}\ (p, \mathsf{inl}^{\gamma}\ q)$ is the position of $\mathsf{nd}\ (x \triangleleft y)\ v$ that we are looking for. Finally, we return the position

$$\mathsf{transport\ Pos\text{-}transp}^{\mathcal{P}}\ (\mathsf{inr}\ (p, \mathsf{inl}^{\gamma}\ q))$$

Definition 1.3.13 (inr^{γ})**.**

$$\begin{aligned}\mathsf{inr}^{\gamma} : {} & \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\ & \to \{t : \mathcal{P}\ (x \triangleleft y)\}\ \{u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)\} \\ & \to (p : \mathsf{Pos}\ y)\ (q : \mathsf{Pos}\ (u\ p)) \to \mathsf{Pos}\ (\gamma\ t\ u)\end{aligned}$$

The operation inr^{γ} is defined by induction on the tree t:

- If t is of the form $\mathsf{lf}\ x$, p is a position of $\eta\ x$, q is a position of $u\ p$, and we have to determine a position of the tree

$$\mathsf{transport}^{\lambda (x,y) \to \mathcal{P}\ (x \triangleleft y)}\ (\mathsf{pair}{=}\ \mathsf{pos}^{\eta}\text{-}\mathsf{typ}^{-1}\ \mu\text{-unit-l})^{-1}\ (u\ (\mathsf{pos}^{\eta}\ x))$$

  We start by establishing a position of $u\ (\mathsf{pos}^{\eta}\ x)$. In order to do so, we eliminate the position p using the elimination rule for η with the motive

$$A\ p :\equiv (q : \mathsf{Pos}\ (u\ p)) \to \mathsf{Pos}\ (u\ (\mathsf{pos}^{\eta}\ x))$$

  and with clause $a_{\eta}\ p :\equiv p$. This elimination rule applied to p and q yields the required position of type $\mathsf{Pos}\ (u\ (\mathsf{pos}^{\eta}\ x))$. It remains to perform the required transport:

$$\mathsf{transport\ Pos\text{-}transp}^{\mathcal{P}}\ (\mathsf{Pos}\text{-}\eta\text{-}\mathsf{elim}\ x\ A\ a_{\eta}\ p\ q)$$

- If t is of the form $\mathsf{nd}\ (x \triangleleft y)\ t$ with $t : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)$ for some $z : \overrightarrow{\mathcal{P}}\ y$, p is a position of $\mu\ y\ z$, q is a position of $u\ p$, and we have to determine a position of the tree

$$\mathsf{transport}^{\lambda y \to \mathcal{P}\ (x \triangleleft y)}\ \mu\text{-assoc}^{-1}\ (\mathsf{nd}\ (x \triangleleft y)\ v)$$

with v defined as

$$v\ p :\equiv \gamma\ (t\ p)\ (u'\ p)$$

and u' defined as

$$u'\ p\ q :\equiv \mathsf{transport}^{\lambda(x,y) \to \mathcal{P}\ (x \triangleleft y)}\ (\mathsf{pair}{=}\ \mu\text{-pos-typ-aux refl})\ (u\ (\mathsf{pair}^{\mu}\ p\ q))$$

We start by establishing a position of $\mathsf{nd}\ (x \triangleleft y)\ v$. In order to do that, we first define $p_0 :\equiv \mathsf{pr}_1^{\mu}\ p$ of type $\mathsf{Pos}\ y$ and $p_1 :\equiv \mathsf{pr}_2^{\mu}\ p$ of type $\mathsf{Pos}\ (z\ p_0)$ then we establish a position of $v\ p_0$ by injecting a position q' of the tree $u'\ p_0\ p_1$ into $\mathsf{Pos}\ (v\ p_0)$. We define q' from q by first transporting it along the path $\mathsf{pair}^{\mu}\text{-}\eta^{-1}$ to obtain a position of the tree $u\ (\mathsf{pair}^{\mu}\ p_0\ p_1)$ then we transport that position along the path $\mathsf{Pos\text{-}transp}^{\mathcal{O}}$ to obtain a position of $u'\ p_0\ p_1$. To summarise,

$$q' :\equiv \mathsf{transport}\ \mathsf{Pos\text{-}transp}^{\mathcal{O}}\ (\mathsf{transport}^{\lambda p \to \mathcal{P}\ (u\ p)}\ \mathsf{pair}^{\mu}\text{-}\eta^{-1}\ q)$$

The position of $v\ p_0$ that we are looking for is therefore the position $r :\equiv \mathsf{inr}^{\gamma}\ p_1\ q'$. We conclude by performing the needed transport to obtain the final position $\mathsf{transport}\ \mathsf{Pos\text{-}transp}^{\mathcal{P}}\ r$.

Finally, we define the elimination principle for positions of type $\mathsf{Pos}\ (\gamma\ t\ u)$.

Definition 1.3.14 (Pos-γ-elim)**.**

$$\begin{aligned}
&\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim} : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\ \{z : \overrightarrow{\mathcal{P}}\ y\} \\
&\quad \to \{t : \mathcal{P}\ (x \triangleleft y)\}\ \{u : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z)\} \\
&\quad \to (A : \mathsf{Pos}\ (\gamma\ t\ u) \to \mathcal{U}) \\
&\quad \to (a_{\mathsf{inl}} : (p : \mathsf{Pos}\ t) \to A\ (\mathsf{inl}^{\gamma}\ p)) \\
&\quad \to (a_{\mathsf{inr}} : (p : \mathsf{Pos}\ y)\ (q : \mathsf{Pos}\ (u\ p)) \to A\ (\mathsf{inr}^{\gamma}\ p\ q)) \\
&\quad \to (p : \mathsf{Pos}\ (\gamma\ t\ u)) \to A\ p
\end{aligned}$$

We define $\mathsf{Pos}\text{-}\gamma\text{-}\mathsf{elim}$ by induction on t:

- If t is of the form $\mathsf{lf}\ x$ then p is a position of the tree

$$\mathsf{transport}^{\lambda(x,y) \to \mathcal{P}\ (x \triangleleft y)}\ (\mathsf{pair}{=}\ \mathsf{pos}^{\eta}\text{-}\mathsf{typ}^{-1}\ \mu\text{-unit-l})^{-1}\ (u\ (\mathsf{pos}^{\eta}\ x))$$

We use the elimination rule $\mathsf{Pos\text{-}transp}^{\mathcal{O}}\text{-}\mathsf{elim}$ with motive A and we are reduced to defining a term of type $A\ (\mathsf{transport}\ \mathsf{Pos\text{-}transp}^{\mathcal{O}}\ p)$ for any position $p : \mathsf{Pos}\ (u\ (\mathsf{pos}^{\eta}\ x))$. The term $a_{\mathsf{inr}}\ (\mathsf{pos}^{\eta}\ x)\ p$ has type $A\ (\mathsf{inr}^{\gamma}\ (\mathsf{pos}^{\eta}\ x)\ p)$. However, $\mathsf{inr}^{\gamma}\ (\mathsf{pos}^{\eta}\ x)\ p$ is equal to $\mathsf{transport}\ \mathsf{Pos\text{-}transp}^{\mathcal{O}}\ p'$ where

$$p' :\equiv \mathsf{Pos}\text{-}\eta\text{-}\mathsf{elim}\ x\ (\lambda p \to \mathsf{Pos}\ (u\ p) \to \mathsf{Pos}\ (u\ (\mathsf{pos}^{\eta}\ x)))\ (\lambda p \to p)\ (\mathsf{pos}^{\eta}\ x)\ p$$

but p' is equal to p according to the β-law for Pos-η-elim, we therefore conclude with this last transport:

$$\mathsf{transport}^{\lambda f \to A\ (\mathsf{transport}\ \text{Pos-transp}^{\mathcal{O}}\ (f\ p))}\ \text{Pos-}\eta\text{-elim-}\beta\ (a_{\mathsf{inr}}\ (\mathsf{pos}^{\eta}\ x)\ p)$$

- If t is of the form $\mathsf{nd}\ (x \triangleleft y)\ \{z\}\ t$ then p is a position of the tree

$$\mathsf{transport}^{\lambda y \to \mathcal{P}\ (x \triangleleft y)}\ \mu\text{-assoc}^{-1}\ (\mathsf{nd}\ (x \triangleleft y)\ v)$$

where

$$\begin{aligned} v\ p &:\equiv \gamma\ (t\ p)\ (u'\ p) \\ u'\ p\ q &:\equiv \mathsf{transport}^{\lambda (x,y) \to \mathcal{P}\ (x \triangleleft y)}\ (\text{pair=}\ \mu\text{-pos-typ-aux refl})\ (u\ (\mathsf{pair}^{\mu} p\ q)) \end{aligned}$$

We use the elimination principle Pos-transp$^{\mathcal{P}}$-elim with motive A, and we are reduced to defining a term of type

$$A\ (\mathsf{transport}\ \text{Pos-transp}^{\mathcal{P}}\ p)$$

for any position $p : \mathsf{Pos}\ (\mathsf{nd}\ (x \triangleleft y)\ \{z\}\ v)$.

We start by defining the shorthand $A'\ p :\equiv A\ (\mathsf{transport}\ \text{Pos-transp}^{\mathcal{P}}\ p)$ then we proceed by induction on p.

  - If p is of the form $\mathsf{inl}\ \star$, we choose $a_{\mathsf{inl}}\ (\mathsf{inl}\ \star)$ which is of the required type $A\ (\mathsf{inl}^{\gamma}\ (\mathsf{inl}\ \star))$ considering that there is a definitional equality

$$\mathsf{inl}^{\gamma}\ (\mathsf{inl}\ \star) \equiv \mathsf{transport}\ \text{Pos-transp}^{\mathcal{P}}\ (\mathsf{inl}\ \star)$$

  - If p is of the form $\mathsf{inr}\ (p, q)$, we are looking for some term of type $A'\ (\mathsf{inr}\ (p, q))$ for positions $p : \mathsf{Pos}\ y$ and $q : \mathsf{Pos}\ (\gamma\ (t\ p)\ (u'\ p))$.
    Our induction hypothesis allows applying Pos-γ-elim to q with motive $\lambda q \to A'\ (\mathsf{inr}\ (p, q))$ and we are reduced to defining two families covering the two possible cases for q.
    We start with the definition of the family of type

$$A'\ (\mathsf{inr}\ (p, \mathsf{inl}^{\gamma}\ q))$$

    for any position $q : \mathsf{Pos}\ (t\ p)$ that is simply obtained as $a_{\mathsf{inl}}\ (\mathsf{inr}\ (p, q))$.
    The second family to define has type

$$A'\ (\mathsf{inr}\ (p, \mathsf{inr}^{\gamma}\ q\ r))$$

    for any positions $q : \mathsf{Pos}\ (z\ p)$ and $r : \mathsf{Pos}\ (u'\ p\ q)$.
    We use the remaining hypothesis and arrive at the following term:

$$a_{\mathsf{inr}}\ (\mathsf{pair}^{\mu}\ p\ q)\ (\mathsf{transport}\ \text{Pos-transp}^{\mathcal{O}^{-1}}\ r)$$

However, the type is not quite right as it is equal to

$$A'\ (\mathsf{inr}\ (p', \mathsf{inr}^{\gamma}\ q'\ r'))$$

with

$$\begin{aligned}
p' &:\equiv \mathsf{pr}_1^{\mu}\ (\mathsf{pair}^{\mu}\ p\ q)\\
q' &:\equiv \mathsf{pr}_2^{\mu}\ (\mathsf{pair}^{\mu}\ p\ q)\\
r' &:\equiv \mathsf{transport}\ \mathsf{Pos\text{-}transp}^{\mathcal{P}}\ r_2\\
r_2 &:\equiv \mathsf{transport}^{\lambda p \to \mathsf{Pos}\ (u\ p)}\ \mathsf{pair}^{\mu}\text{-}\eta^{-1}\ r_3\\
r_3 &:\equiv \mathsf{transport}\ \mathsf{Pos\text{-}transp}^{\mathcal{P}^{-1}}\ r
\end{aligned}$$

It then remains to perform a transport along an identity $(p, q, r) = (p', q', r')$. An identity $(p, q) = (p', q')$ is readily established using both μ-pos-fst-β and μ-pos-snd-β. In order to prove the identity

$$\mathsf{transport}^{\lambda(p,q)\to\mathsf{Pos}\ (u'\ p\ q)}\ (\mathsf{pair}{=}\ \mu\text{-pos-fst-}\beta\ \mu\text{-pos-snd-}\beta)\ r' = r$$

we can show, albeit tediously, that the left-hand side is of the form $\mathsf{transport}^{\mathsf{Pos}}\ e\ r$ for some path e of type

$$\begin{aligned}
&\mathsf{transport}^{\lambda(x,y)\to\mathcal{P}\ (x \triangleleft y)}\ (\mathsf{pair}{=}\ \mu\text{-pos-typ-aux refl})\ (u\ (\mathsf{pair}^{\mu}\ p\ q))\\
&= \mathsf{transport}^{\lambda(x,y)\to\mathcal{P}\ (x \triangleleft y)}\ (\mathsf{pair}{=}\ \mu\text{-pos-typ-aux refl})\ (u\ (\mathsf{pair}^{\mu}\ p\ q))
\end{aligned}$$

But, the type of trees being a set, this identity must be equal to refl which permits to conclude the proof.

This concludes our definition of opetopes.

1.4 Faces of an opetope

Our inductive definition of opetopes allows to easily characterise the faces which compose them. We will define the type family

$$\mathcal{F} : \{n : \mathbb{N}\} \to \mathcal{O}\ n \to \mathbb{N} \to \mathcal{U}$$

For any opetope $o : \mathcal{O}\ n$, the elements of $\mathcal{F}\ o\ m$ will be the $(m-1)$-dimensional faces of o. We will explain why this dimension shift later on. Let us pause and think what should the faces of an opetope be. Any opetope has a top face whose dimension is the one of the opetope; therefore for any opetope $o : \mathcal{O}\ n$, there will be a constructor $\mathsf{top} : \mathcal{F}\ o\ (n+1)$. In particular, this will be the only face of the object ob. Now, what could the other faces of an opetope be? Any $(n+1)$-opetope has some source faces and a single target face as n-dimensional faces. The target face is unique so, given any opetope $(x \triangleleft y) : \mathcal{O}\ (n+1)$ there will be a constructor $\mathsf{target} : \mathcal{F}\ (x \triangleleft y)\ (n+1)$. As for the source faces, they are in correspondence with the node positions of y. So, for each position $p : \mathsf{Pos}\ y$, there will be a

constructor $\mathsf{src}\ p : \mathcal{F}\ (x \triangleleft y)\ (n+1)$. The difficulty is now to characterise the faces of lower dimension. The definition that we propose consists in identifying the remaining faces — if any — with the *edges* of the trees whose sequence defines an opetope.

We now present our definition of the faces of an opetope which will be followed with some examples.

1.4.1 Definitions

We first define the type of edges of a tree.

Definition 1.4.1 (Edges of a tree)**.** The family of edges of a tree has signature

$$\mathcal{E} : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ (y : \mathcal{P}\ x) \to \mathcal{U}$$

It is inductively defined with constructors:

$$\begin{aligned}
&\mathsf{lf\text{-}edge} : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n) \to \mathcal{E}\ (\mathsf{lf}\ o)\\
&\mathsf{root\text{-}edge} : \{n : \mathbb{N}\}\ (x : \mathcal{O}\ n)\ (y : \mathcal{P}\ x)\ \{z : \overrightarrow{\mathcal{P}}\ y\}\\
&\quad \to (t : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z))\\
&\quad \to \mathcal{E}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\\
&\mathsf{nd\text{-}edge} : \{n : \mathbb{N}\}\ (x : \mathcal{O}\ n)\ (y : \mathcal{P}\ x)\ \{z : \overrightarrow{\mathcal{P}}\ y\}\\
&\quad \to (t : \overrightarrow{\mathcal{P}}\ (y \blacktriangleleft z))\\
&\quad \to (p : \mathsf{Pos}\ y) \to \mathcal{E}\ (t\ p)\\
&\quad \to \mathcal{E}\ (\mathsf{nd}\ (x \triangleleft y)\ t)
\end{aligned}$$

We characterise the edges of the two possible forms of trees. In the case of a leaf, there is a single edge corresponding to the leaf itself. While in the case of a tree of the form $\mathsf{nd}\ (x \triangleleft y)\ t$, an edge is either the root edge or an edge of one of the trees grafted on the root node at some position $p : \mathsf{Pos}\ y$.

We want to collect the edges of the sequence of trees forming an opetope, we therefore need a further type collecting these edges corresponding to the lower faces of an opetope.

Definition 1.4.2. The family of lower faces of an opetope has type

$$\underline{\mathcal{F}} : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n) \to \mathbb{N} \to \mathcal{U}$$

and is defined as an inductive type with two constructors

$$\begin{aligned}
&\mathsf{edge} : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\}\\
&\quad \to \mathcal{E}\ y \to \underline{\mathcal{F}}\ (x \triangleleft y)\ n\\
&\mathsf{lower} : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ (y : \mathcal{P}\ x)\ \{k : \mathbb{N}\}\\
&\quad \to \underline{\mathcal{F}}\ x\ k \to \underline{\mathcal{F}}\ (x \triangleleft y)\ k
\end{aligned}$$

A lower face of a $(n+1)$-opetope of the form $x \triangleleft y$ is therefore either an edge of y or a lower face of the opetope x. We can finally explain the dimension shift in the indexing. The edges of y correspond to the $(n-1)$-dimensional faces of $x \triangleleft y$ but in order to avoid subtracting indices, we choose to index n-dimensional faces with the index $n+1$.

We are finally in position to define the type of faces of an opetope.

Definition 1.4.3 (Faces of an opetope)**.** The family of faces of an opetope has type

$$\mathcal{F} : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n) \to \mathbb{N} \to \mathcal{U}$$

and is defined as an inductive type whose constructors are

$$\begin{array}{ll}
\mathsf{top} & : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ n) \to \mathcal{F}\ o\ (n+1) \\
\mathsf{target} & : \{n : \mathbb{N}\}\ (o : \mathcal{O}\ (n+1)) \to \mathcal{F}\ o\ (n+1) \\
\mathsf{src} & : \{n : \mathbb{N}\}\ \{x : \mathcal{O}\ n\}\ \{y : \mathcal{P}\ x\} \to \mathsf{Pos}\ y \to \mathcal{F}\ (x \triangleleft y)\ (n+1) \\
\mathsf{lower\text{-}face} & : \{n : \mathbb{N}\}\ \{o : \mathcal{O}\ n\}\ \{k : \mathbb{N}\} \to \underline{\mathcal{F}}\ o\ k \to \mathcal{F}\ o\ k
\end{array}$$

The first three constructors correspond to the ones discussed in the introduction of this section and the last one corresponds to the lower faces.

The faces of an opetopes actually assemble into an opetopic type. Opetopic types will be the subject of Chapter 3; however, the precise definition of the opetopic type of the faces of an opetope is still a work in progress.

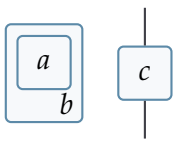
1.4.2 Examples

We revisit a few elementary opetopes and list their faces in order to illustrate our definition.

The object The object has a single face $\mathsf{top}\ \mathsf{ob}$.

The arrow The arrow $\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd}$ has three faces:

$$\begin{aligned}
a &:\equiv \mathsf{src}\ \star \\
b &:\equiv \mathsf{target}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd}) \\
c &:\equiv \mathsf{top}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})
\end{aligned}$$



The 2-simplex The 2-simplex (ob ◃ ob-pd ◃ t) where

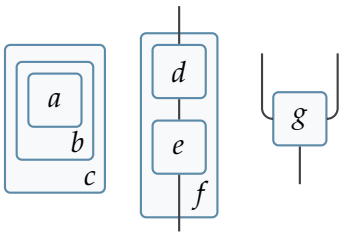
$$t :\equiv \mathsf{nd}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ (\lambda p \longrightarrow \mathsf{nd}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ (\lambda q \longrightarrow \mathsf{lf}\ (\mathsf{Typ}\ \mathsf{ob\text{-}pd}\ q)))$$

has seven faces:

$$\begin{aligned}
a &:\equiv \mathsf{lower\text{-}face}\ (\mathsf{edge}\ (\mathsf{nd\text{-}edge}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ t' \star \\
&\qquad (\mathsf{nd\text{-}edge}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ (\lambda p \longrightarrow \mathsf{lf}\ (\mathsf{Typ}\ \mathsf{ob\text{-}pd}\ p))\ \star\ (\mathsf{lf\text{-}edge}\ \mathsf{ob}))))\\
b &:\equiv \mathsf{lower\text{-}face}\ (\mathsf{edge}\ (\mathsf{nd\text{-}edge}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ t' \star \\
&\qquad (\mathsf{root\text{-}edge}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ (\lambda p \longrightarrow \mathsf{lf}\ (\mathsf{Typ}\ \mathsf{ob\text{-}pd}\ p)))))\\
c &:\equiv \mathsf{lower\text{-}face}\ (\mathsf{edge}\ (\mathsf{root\text{-}edge}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ t'))\\
d &:\equiv \mathsf{src}\ (\mathsf{inr}\ (\star, \mathsf{inl}\ \star))\\
e &:\equiv \mathsf{src}\ (\mathsf{inl}\ \star)\\
f &:\equiv \mathsf{target}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd} \triangleleft t)\\
g &:\equiv \mathsf{top}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd} \triangleleft t)
\end{aligned}$$

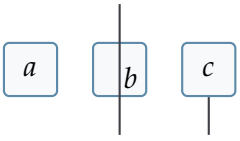
where

$$t' :\equiv \mathsf{nd}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd})\ (\lambda p \longrightarrow \mathsf{lf}\ (\mathsf{Typ}\ \mathsf{ob\text{-}pd}\ p))$$



Loops The loop ob ◃ ob-pd ◃ lf ob has three faces:

$$\begin{aligned}
a &:\equiv \mathsf{lower\text{-}face}\ (\mathsf{edge}\ (\mathsf{lf\text{-}edge}\ \mathsf{ob}))\\
b &:\equiv \mathsf{target}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd} \triangleleft \mathsf{lf}\ \mathsf{ob})\\
c &:\equiv \mathsf{top}\ (\mathsf{ob} \triangleleft \mathsf{ob\text{-}pd} \triangleleft \mathsf{lf}\ \mathsf{ob})
\end{aligned}$$



Leaves having no node, the only 1-dimensional face in this example is b which is the target of the opetope ob ◃ ob-pd ◃ lf ob.

Chapter 2

Polynomial monads

In this chapter, we extend type theory with a universe of cartesian polynomial monads closed under a number of monad constructors. We will take advantage of these structures to define opetopic types, collections of types whose geometry is governed by opetopes, in Chapter 3. Opetopes are then obtained as the terminal opetopic type. Most importantly, we require the laws of polynomial monads to hold definitionally in order to avoid making the coherence laws explicit as our constructions are not sets any more in this chapter. We will finally add two extensions to our system: morphisms of monads and monad families.

2.1 Cartesian polynomial monads

Cartesian polynomial monads form the backbone of our system. They are at the heart of the so-called Baez-Dolan construction (Baez and Dolan 1998) which was introduced in the setting of operads for the purpose of defining n-categories. Cartesian polynomial monads can be understood as presentations for strongly regular algebraic theories. These are algebraic theories whose equations are constrained in such a way that variables have to appear in both sides without repetition and in the same order (Leinster 2004). We will not systematically specify that our polynomial monads are cartesian even though they always are in order to lighten the notation.

The endofunctor part of the polynomial monad structure is also known as an indexed container in the type theory literature (Abbott, Altenkirch and Ghani 2005; Altenkirch, Ghani et al. 2015). They provide a calculus of datatypes used to internalise a large class of inductive datatypes in type theory allowing, for example, the conception of generic algorithms on datatypes.

2.1.1 Polynomials

We start by postulating a universe of polynomial monads $\mathcal{M} : \mathcal{U}$ whose elements are regarded as codes for our polynomial monads. Contrary to a regular universe à la Tarski there will not be a single decoding function but four of them

$$\begin{aligned}
&\mathsf{Idx} : \mathcal{M} \to \mathcal{U} \\
&\mathsf{Cns} : (M : \mathcal{M}) \to \mathsf{Idx}\ M \to \mathcal{U} \\
&\mathsf{Pos} : (M : \mathcal{M})\ \{i : \mathsf{Idx}\ M\} \to \mathsf{Cns}\ M\ i \to \mathcal{U} \\
&\mathsf{Typ} : (M : \mathcal{M})\ \{i : \mathsf{Idx}\ M\}\ (c : \mathsf{Cns}\ M\ i) \to \mathsf{Pos}\ M\ c \to \mathsf{Idx}\ M
\end{aligned}$$

Figure 2.1: The data defining a polynomial

which will describe the different data defining a *polynomial* (Figure 2.1). For any monad $M : \mathcal{M}$, we can regard this data as describing the signature of an algebraic theory: the elements of $\mathsf{Idx}\ M$, which we refer to as *indices* serve as the sorts of the theory, and for $i : \mathsf{Idx}\ M$, the type $\mathsf{Cns}\ M\ i$ is the collection of operation symbols whose output sort is i. The type $\mathsf{Pos}\ M\ c$ then assigns to each operation a collection of input positions which are themselves assigned an index via the function $\mathsf{Typ}\ M$. We will write the monad indexing the decoding functions in subscript and write $\mathsf{Cns}_M\ i$ instead of $\mathsf{Cns}\ M\ i$ for example.

For any monad M, the data of its polynomial gives rise to an endomorphism called its *extension*

$$[\![M]\!] : (X : \mathsf{Idx}_M \to \mathcal{U}) \to \mathsf{Idx}_M \to \mathcal{U}$$

whose defining equation is

$$[\![M]\!]\ X\ i :\equiv \textstyle\sum_{(c:\mathsf{Cns}_M\ i)} (p : \mathsf{Pos}_M\ c) \to X\ (\mathsf{Typ}_M\ c\ p)$$

This is the type of constructors of M whose inputs are decorated with elements in X. Polynomials are definable in any locally cartesian closed category, and their extension is then a functor on some slice category (GAMBINO and KOCK 2013).

2.1.2 Monads

In this section, we specify some additional structure on polynomials which, in a categorical setting, endows their extension with a structure of a monad. We therefore refer to the elements of $\mathcal{M}$ as polynomial monads by abuse of language.

$$\eta : (M : \mathcal{M})\ (i : \mathsf{Idx}_M) \to \mathsf{Cns}_M\ i$$
$$\mu : (M : \mathcal{M})\ \{i : \mathsf{Idx}_M\}\ (c : \mathsf{Cns}_M\ i)\ (d : \overrightarrow{\mathsf{Cns}_M}\ c) \to \mathsf{Cns}_M\ i$$

Figure 2.2: The monad structure

Notation. We denote Fam_M the type of families of types indexed by the indices of a monad M:

$$\mathsf{Fam}_M :\equiv \mathsf{Idx}_M \to \mathcal{U}$$

We also introduce a notation for the type of sections of type families of the form $A \circ \mathsf{Typ}_M\ c$ which are indexed by the positions of a constructor c. Given a monad M, a type family $A : \mathsf{Fam}_M$, and a constructor $c : \mathsf{Cns}_M\ i$, we define the notation

$$\overrightarrow{A}\ c :\equiv (p : \mathsf{Pos}_M\ c) \to A\ (\mathsf{Typ}_M\ c\ p)$$

A particularly recurrent family will be $\overrightarrow{\mathsf{Cns}_M}\ c$ for some constructor c. A constructor $c : \mathsf{Cns}_M\ i$ along with a family $\overrightarrow{\mathsf{Cns}_M}\ c$ denote a depth-2 tree of constructors.

The structure of a polynomial monad comprises two operations (Figure 2.2). Our monads being cartesian, they will satisfy the following equivalences:

$$\mathsf{Pos}_M\ (\mu_M\ c\ d) \simeq \textstyle\sum_{(p:\mathsf{Pos}_M\ c)} \mathsf{Pos}_M\ (d\ p)$$
$$\mathsf{Pos}_M\ (\eta_M\ i) \simeq \mathbf{1}$$

Moreover, their typing function Typ_M is compatible with these equivalences. The unit $\eta_M\ i$ is therefore a *unary* constructor whose both input and output are labelled by the index i. As for the operation μ_M, it composes a depth-2 tree of constructors to a constructor sharing its index with the one of the root node. In addition, μ_M preserves the type of positions of its input tree as well as their typing.

These operations have to satisfy a number of definitional laws which we will detail after having introduced some further operations on positions with regard to the cartesian monad structure. Indeed, the mentioned equivalences will rarely be equal to the identity. That is why we will characterise the type of positions of constructors obtained from the operations η and μ using operations that can be seen as witnessing these equivalences but which will obey definitional laws. These operations on positions correspond exactly to those introduced in Chapter 1.

Operations relative to $\mathsf{Pos}_M\ (\eta_M\ i)$

The type of positions of a unit constructor is contractible and can be characterised by the existence of a centre of contraction along with an elimination rule equivalent to the one of the unit type which leads us to postulate the following principles:

$$\begin{aligned}
&\mathsf{pos}^{\eta} : (M : \mathcal{M})\ (i : \mathsf{Idx}_M)\ \to \mathsf{Cns}_M\ i \\
&\mathsf{Pos\text{-}\eta\text{-}elim} : (M : \mathcal{M})\ (i : \mathsf{Idx}_M)\ (X : \mathsf{Pos}_M\ (\eta_M\ i) \to \mathcal{U}) \\
&\quad \to (x_{\mathsf{pos}^{\eta}} : X\ (\mathsf{pos}^{\eta}_M\ i)) \\
&\quad \to (p : \mathsf{Pos}_M\ (\eta_M\ i)) \to X\ p
\end{aligned}$$

The elimination rule satisfies the following *definitional* computation rule:

$$\mathsf{Pos\text{-}\eta\text{-}elim}_M\ i\ X\ x_{\mathsf{pos}^{\eta}}\ (\mathsf{pos}^{\eta}_M\ i) \equiv x_{\mathsf{pos}^{\eta}} \qquad (\mathsf{Pos\text{-}\eta\text{-}elim}\text{-}\beta)$$

The typing function is compatible with the operation η_M:

$$\mathsf{Typ}_M\ (\eta_M\ i)\ p \equiv i \qquad (\mathsf{pos}^{\eta}\text{-}\mathsf{typ})$$

Operations relative to $\mathsf{Pos}_M\ (\mu_M\ x\ y)$

The type of positions of a constructor obtained through the application of the operation μ_M is characterised by a pairing function along with two projection functions similarly to sigma types:

$$\begin{aligned}
&\mathsf{pair}^{\mu} : (M : \mathcal{M})\ \{i : \mathsf{Idx}_M\}\ (c : \mathsf{Cns}_M\ i)\ (d : \overrightarrow{\mathsf{Cns}_M}\ c) \\
&\quad \to (p : \mathsf{Pos}_M\ c)\ (q : \mathsf{Pos}_M\ (d\ p)) \\
&\quad \to \mathsf{Pos}_M\ (\mu_M\ c\ d) \\
&\mathsf{pr}^{\mu}_1 : (M : \mathcal{M})\ \{i : \mathsf{Idx}_M\}\ \{c : \mathsf{Cns}_M\ i\}\ \{d : \overrightarrow{\mathsf{Cns}_M}\ c\} \\
&\quad \to \mathsf{Pos}_M\ (\mu_M\ c\ d) \to \mathsf{Pos}_M\ c \\
&\mathsf{pr}^{\mu}_2 : (M : \mathcal{M})\ \{i : \mathsf{Idx}_M\}\ \{c : \mathsf{Cns}_M\ i\}\ \{d : \overrightarrow{\mathsf{Cns}_M}\ c\} \\
&\quad \to (p : \mathsf{Pos}_M\ (\mu_M\ c\ d)) \to \mathsf{Pos}_M\ (d\ (\mathsf{pr}^{\mu}_1\ p))
\end{aligned}$$

These functions are inverse to each other which is witnessed by the following definitional equalities:

$$\mathsf{pair}^{\mu}_M\ (\mathsf{pr}^{\mu}_{1\,M}\ p)\ (\mathsf{pr}^{\mu}_{2\,M}\ p) \equiv p \qquad (\mathsf{pair}^{\mu}\text{-}\eta)$$

$$\mathsf{pr}^{\mu}_{1\,M}\ (\mathsf{pair}^{\mu}_M\ p\ q) \equiv p \qquad (\mathsf{pr}^{\mu}_1\text{-}\beta)$$

$$\mathsf{pr}^{\mu}_{2\,M}\ (\mathsf{pair}^{\mu}_M\ p\ q) \equiv q \qquad (\mathsf{pr}^{\mu}_2\text{-}\beta)$$

Note how $\mathsf{pr}^{\mu}_2\text{-}\beta$ is stated without having to transport the left-hand side along $\mathsf{pr}^{\mu}_1\text{-}\beta$ now that the laws are definitional. The typing function is compatible with the operation μ_M

$$\mathsf{Typ}_M\ (\mu_M\ c\ d)\ p = \mathsf{Typ}_M\ (d\ (\mathsf{pr}^{\mu}_{1\,M}\ p))\ (\mathsf{pr}^{\mu}_{2\,M}\ p) \qquad (\mu\text{-}\mathsf{pos}\text{-}\mathsf{typ})$$

Finally, we postulate the laws obeyed by the operations η_M and μ_M:

$$\mu_M\ c\ (\lambda p \to \eta_M\ (\mathsf{Typ}_M\ c\ p)) \equiv c$$
$$\mu_M\ (\eta_M\ i)\ c \equiv c\ (\mathsf{pos}^{\eta}_M\ i)$$
$$\mu_M\ (\mu_M\ c\ d)\ e \equiv \mu_M\ c\ (\lambda p \to \mu_M\ (d\ p)\ (\lambda q \to e\ (\mathsf{pair}^{\mu}_M\ p\ q)))$$

That is, the operation μ_M is associative and unital with unit η_M.

You might have noticed that we have not mentioned the operation γ so far. This is because it is not part of the definition of a polynomial monad. Instead, it will be an operation concerning a specific monad, the slice monad $M/$ of a monad M, and which will be used in order to define the operation $\mu_{M/}$.

2.2 The universe of polynomial monads

We now populate the universe with some monad constructors. In order to do so, we postulate a new code for each monad that we want to add, then we define its corresponding decoding functions specifying the data of its polynomial along with the operations characterising its cartesian monad structure.

2.2.1 The identity monad

The first monad we introduce is the trivial monad with a single constructor which is unary and whose type of sorts is the unit type. We start by postulating the new code $\mathsf{Id} : \mathcal{M}$. Its polynomial part is predictably defined as follows:

$$\begin{aligned}
\mathsf{Idx}_{\mathsf{Id}} &:\equiv \mathbf{1} \\
\mathsf{Cns}_{\mathsf{Id}}\ i &:\equiv \mathbf{1} \\
\mathsf{Pos}_{\mathsf{Id}}\ c &:\equiv \mathbf{1} \\
\mathsf{Typ}_{\mathsf{Id}}\ c\ p &:\equiv \star
\end{aligned}$$

We do not detail the definitions of the different operations relative to the identity monad as they are all trivial, having the unit type as codomain. The extension of the identity monad $[\![\mathsf{Id}]\!]$ is then equivalent to the identity on $\mathsf{Fam}_{\mathsf{Id}}$ hence its name.

2.2.2 The pullback monad

Given a monad M and a type family $X : \mathsf{Fam}_M$, we define a new code for the pullback monad $X^*\,M : \mathcal{M}$ which reindexes the monad M. Its constructors are constructors of M along with a decoration of both their inputs and output with elements of the family X. Its polynomial part is defined as follows:

$$\begin{aligned}
\mathsf{Idx}_{X^*M} &:\equiv \textstyle\sum_{(i:\mathsf{Idx}_M)} X\ i \\
\mathsf{Cns}_{X^*M}\ (i, x) &:\equiv \textstyle\sum_{(c:\mathsf{Cns}_M\ i)} \overrightarrow{X}\ c \\
\mathsf{Pos}_{X^*M}\ (c, x) &:\equiv \mathsf{Pos}_M\ c \\
\mathsf{Typ}_{X^*M}\ (c, x)\ p &:\equiv \mathsf{Typ}_M\ c\ p
\end{aligned}$$

The operations of the pullback monad $X^* M$ simply use the ones of the underlying monad M.

$$\eta_{X^* M}\ (i, x) :\equiv (\eta_M\ i, \eta\text{-dec}_M\ x)$$

where $\eta\text{-dec}_M\ x$ is defined as

$$\eta\text{-dec}_M\ x :\equiv \mathsf{Pos}\text{-}\eta\text{-}\mathsf{elim}_M\ i\ (\lambda p \to X\ (\mathsf{Typ}_M\ (\eta_M\ i)\ p))\ x$$

The unit then takes an index $i : \mathsf{Idx}\ M$ as well as an element $x : X\ i$ and returns the unit of the underlying monad $\eta_M\ i$ whose only position is decorated with the element x.

The multiplication is defined as

$$\mu_{X^* M}\ (c, x)\ d :\equiv (\mu_M\ c\ (\lambda p \to \mathsf{pr}_1\ (d\ p)), \lambda p \to \mathsf{pr}_2\ (d\ (\mathsf{pr}^{\mu}_{1\,M}\ p))\ (\mathsf{pr}^{\mu}_{2\,M}\ p))$$

The multiplication multiplies the M-constructors given by c and d using μ_M, then decorates the positions of the resulting constructor using the decoration specified by d while forgetting the inner decoration specified by x.

As for the operations on the positions, they are straightforward since the pullback monad shares its type of positions with its underlying monad.

$$\begin{aligned}
&\mathsf{pos}^{\eta}_{X^* M}\ (i, x) &&:\equiv \mathsf{pos}^{\eta}_{M}\ i\\
&\mathsf{Pos}\text{-}\eta\text{-}\mathsf{elim}_{X^* M}\ (i, x)\ X\ x_{\mathsf{pos}^{\eta}}\ p &&:\equiv \mathsf{Pos}\text{-}\eta\text{-}\mathsf{elim}_M\ i\ X\ x_{\mathsf{pos}^{\eta}}\ p
\end{aligned}$$

Finally, the positions operations with regard to the operation μ are defined as follows:

$$\begin{aligned}
\mathsf{pair}^{\mu}_{X^* M}\ p\ q &:\equiv \mathsf{pair}^{\mu}_{M}\ p\ q\\
\mathsf{pr}^{\mu}_{1\,X^* M}\ p &:\equiv \mathsf{pr}^{\mu}_{1\,M}\ p\\
\mathsf{pr}^{\mu}_{2\,X^* M}\ p &:\equiv \mathsf{pr}^{\mu}_{2\,M}\ p
\end{aligned}$$

2.2.3 The slice monad

The slice monad of a monad M, denoted $M/$, is at the heart of the Baez-Dolan construction (BAEZ and DOLAN 1998). It is the monad which materialises the algebraic structure of a monad M, encoding it into data as constructors of the monad $M/$. Iterating this construction, we capture the laws governing the multiplication of M, then the coherences they satisfy, and so forth. This is the monad constructor which describes how to raise the dimension of opetopes.

The slice polynomial

We start by postulating a new code $M/ : \mathcal{U}$ for any monad M. Its indices are the constructors of M:

$$\mathsf{Idx}_{M/} :\equiv \textstyle\sum_{(i : \mathsf{Idx}_M)} \mathsf{Cns}_M\ i$$

Notation. We will adopt the notation of Chapter 1 and denote the elements of $\mathsf{Idx}_{M/}$ as $y \triangleleft x$ in place of (y, x). This is meant to convey the idea that $y \triangleleft x$ is the data of a configuration of inputs given by x along with the data of an output given by y. A type family $\mathsf{Fam}_{M/}$ is then regarded as a relation over this data. As the type of x depends on y, we will often reverse the naming order and write $x \triangleleft y$ instead.
We also extend our notation to families. For any type family $X : \mathsf{Fam}_{M/}$, any constructor $x : \mathsf{Cns}_M\ i$, and any family $y : \overrightarrow{\mathsf{Cns}_M}\ x$, we define the following notation:

$$\overrightarrow{X}\ (x \blacktriangleleft y) :\equiv (p : \mathsf{Pos}_M\ x) \to X\ (\mathsf{Typ}_M\ x\ p \triangleleft y\ p)$$

We now define the type of constructors of the slice monad of M

$$\mathsf{Cns}_{M/} : \mathsf{Idx}_{M/} \to \mathcal{U}$$

Constructors of the monad $M/$ indexed by $x \triangleleft y$ are well-founded trees of constructors of M which multiply to y — a x-indexed constructor of M — using the monad structure of M. The type family $\mathsf{Cns}_{M/}$ is therefore defined as the following inductive type:

$$\begin{aligned}
&\mathsf{lf} : (x : \mathsf{Idx}_M) \to \mathsf{Cns}_{M/}\ (x \triangleleft \eta_M\ x)\\
&\mathsf{nd} : (x : \mathsf{Idx}_M)\ (y : \mathsf{Cns}_M\ x)\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\}\\
&\quad \to (t : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z))\\
&\quad \to \mathsf{Cns}_{M/}\ (x \triangleleft \mu_M\ y\ z)
\end{aligned}$$

Once, again we will write $\mathsf{nd}\ (x \triangleleft y)\ t$ instead of $\mathsf{nd}\ x\ y\ t$. This type is very similar to $\mathcal{P}$ from Chapter 1 and it helps to understand it pictorially. The presentation given in the simpler context of opetopes transposes to this new setting. The leaf $\mathsf{lf}\ x$ represents the empty tree with one input and one output sharing the same index x. The node $\mathsf{nd}\ (x \triangleleft y)\ t$ represents a tree whose root node is the constructor y on which are recursively grafted the trees specified by t on the positions of y. As trees are indexed by constructors, one has to first specify a family of constructors z indexed by the positions of y. The tree $t\ p$ grafted at position p of y is therefore indexed by $\mathsf{Typ}\ y\ p \triangleleft z\ p$. The resulting tree, $\mathsf{nd}\ (x \triangleleft y)\ t$, is indexed by $x \triangleleft \mu_M\ y\ z$ where $\mu_M\ y\ z$ is indeed its image under the operation μ_M since this operation is associative.

The type of positions of a constructor of the slice monad is the type of paths from the root of the tree to its nodes:

$$\begin{aligned}
&\mathsf{Pos}_{M/}\ (\mathsf{lf}\ x) &&:\equiv \bot\\
&\mathsf{Pos}_{M/}\ (\mathsf{nd}\ (x \triangleleft y)\ t) &&:\equiv \mathbf{1} + \textstyle\sum_{(p:\mathsf{Pos}_M\ y)} \mathsf{Pos}_{M/}\ (t\ p)
\end{aligned}$$

The leaf has no node and therefore no position. A position of a tree made of a root node is either this root node or a position of one of the trees grafted on this node.

The typing function then projects out the index corresponding to the node indicated by a position p:

$$\begin{aligned}\mathsf{Typ}_{M/}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ (\mathsf{inl}\ \star) &:\equiv x \triangleleft y\\ \mathsf{Typ}_{M/}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ (\mathsf{inr}\ (p, q)) &:\equiv \mathsf{Typ}_{M/}\ (t\ p)\ q\end{aligned}$$

The monad structure

We now define the monad structure of the slice polynomial. The operations will be similar to the ones acting on $\mathcal{P}$ but, considering that their laws are now taken to be definitional, their definition will be greatly simplified.

The unit takes an index $x \triangleleft y : \mathsf{Idx}_{M/}$ and returns the tree whose sole node is $x \triangleleft y$:

$$\eta_{M/}\ (x \triangleleft y) :\equiv \mathsf{nd}\ (x \triangleleft y)\ (\lambda p \to \mathsf{lf}\ (\mathsf{Typ}_M\ y\ p))$$

The multiplication substitutes a family of compatible trees for the nodes of a given tree. It requires the definition of the grafting operation γ that we already encountered in Chapter 1:

$$\begin{aligned}\gamma : (M : \mathcal{M})\ \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\}\\ \to (t : \mathsf{Cns}_{M/}\ (x \triangleleft y))\ (u : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z))\\ \to \mathsf{Cns}_{M/}\ (x \triangleleft \mu_M\ y\ z)\end{aligned}$$

This operation takes a tree t and a family of trees u indexed by the positions of t and returns the result of the graft of the family u on t. It is defined by induction on t:

- If t is of the form $\mathsf{lf}\ x$, we simply return $u\ (\mathsf{pos}^{\eta}_M\ x)$. Note that this typechecks due to μ-unit-l being definitional.
- If t is of the form $\mathsf{nd}\ (x \triangleleft y)\ t$, we recursively graft the trees specified by t and u on the root node $x \triangleleft y$:

$$\mathsf{nd}\ (x \triangleleft y)\ (\lambda p \to \gamma_M\ (t\ p)\ (\lambda q \to u\ (\mathsf{pair}^{\mu}_M\ p\ q)))$$

We are now in position to define $\mu_{M/}\ c\ d$ by induction on c:

- If c is of the form $\mathsf{lf}\ x$, there is nothing to substitute and we return $\mathsf{lf}\ x$ intact.
- If c is of the form $\mathsf{nd}\ (x \triangleleft y)\ t$, the root note will be substituted with the tree $d\ (\mathsf{inl}\ \star)$, in consequence we recursively graft on this tree the result of the substitution of the nodes of t with the corresponding trees specified by d:

$$\gamma_M\ (d\ (\mathsf{inl}\ \star))\ (\lambda p \to \mu_{M/}\ (t\ p)\ (\lambda q \to d\ (\mathsf{inr}\ (p, q))))$$

Operations on positions

We now define the position operations of the slice monad.

η position operations Let an index $i : \mathsf{Idx}_{M/}$, we define $\mathsf{pos}^{\eta}_{M/}\ i$ to be the only position of $\eta_{M/}\ i$:

$$\mathsf{pos}^{\eta}_{M/}\ i :\equiv \mathsf{inl}\ \star$$

As for the elimination rule, $\mathsf{Pos\text{-}\eta\text{-}elim}_{M/}\ i\ X\ x_\eta\ p$ is defined by induction on p of type $\mathsf{Pos}_{M/}\ (\eta_{M/}\ i)$ whose sole case is when p is equal to $\mathsf{pos}^{\eta}_{M/}\ i$:

$$\mathsf{Pos\text{-}\eta\text{-}elim}_{M/}\ i\ X\ x_\eta\ (\mathsf{pos}^{\eta}_{M/}\ i) :\equiv x_\eta$$

γ position operations We need some operations characterising the positions of constructors obtained through γ before moving on to the μ case. The positions of trees of the form $\gamma_M\ t\ u$ fall into two categories.

In the first case, inl^{γ} injects a position of a tree t into the type of positions of the tree $\gamma_M\ t\ u$:

$$\begin{aligned}\mathsf{inl}^{\gamma} : (M : \mathcal{M})\ \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\}\\ \to \{t : \mathsf{Cns}_{M/}\ (x \triangleleft y)\}\ \{u : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z)\}\\ \to (p : \mathsf{Pos}_{M/}\ t) \to \mathsf{Pos}_{M/}\ (\gamma_M\ t\ u)\end{aligned}$$

It is defined by induction on t and on p,

- If t is of the form $\mathsf{lf}\ x$, $\mathsf{Pos}\ t$ is the empty type and there is nothing to do.
- If t is of the form $\mathsf{nd}\ (x \triangleleft y)\ t$, we have to return a position of the tree $\mathsf{nd}\ (x \triangleleft y)\ v$ where $v\ p :\equiv \gamma_M\ (t\ p)\ (\lambda q \to u\ (\mathsf{inr}\ (p, q)))$. There are two cases to cover for p:
  - If p is $\mathsf{inl}\ \star$, that is, the position of the root node of the base tree, we return the position of the root node of the resulting tree which is $\mathsf{inl}\ \star$.
  - If p is of the form $\mathsf{inr}\ (p, q)$ with $p : \mathsf{Pos}\ y$ and $q : \mathsf{Pos}\ (t\ p)$, we obtain a position of $v\ p$ as $\mathsf{inl}^{\gamma}_M\ q$ and finally obtain a position of $\mathsf{nd}\ (x \triangleleft y)\ v$ as $\mathsf{inr}\ (p, \mathsf{inl}^{\gamma}_M\ q)$.

In the second case, given a tree $t : \mathsf{Cns}_{M/}\ (x \triangleleft y)$ and a family $u : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z)$, inr^{γ} injects a position of the tree $u\ p$ for some position $p : \mathsf{Pos}_M\ y$ into the type of positions of $\gamma_M\ t\ u$.

$$\begin{aligned}\mathsf{inr}^{\gamma} : (M : \mathcal{M})\ \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\}\\ \to \{t : \mathsf{Cns}_{M/}\ (x \triangleleft y)\}\ \{u : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z)\}\\ \to (p : \mathsf{Pos}_M\ y)\ (q : \mathsf{Pos}_{M/}\ (u\ p)) \to \mathsf{Pos}_{M/}\ (\gamma_M\ t\ u)\end{aligned}$$

It is defined by induction on t,

- If t is of the form $\mathsf{lf}\ x$, we have to return a position of $u\ (\mathsf{pos}^{\eta}_M\ x)$ but q is a position of $u\ p$ where $p : \mathsf{Pos}_M\ (\eta_M\ x)$. We therefore have to use the corresponding elimination principle with motive

$$A\ p :\equiv \mathsf{Pos}_{M/}\ (u\ p) \to \mathsf{Pos}_{M/}\ (u\ (\mathsf{pos}^{\eta}_M\ x))$$

and return the position

$$\mathsf{Pos\text{-}\eta\text{-}elim}_M\ x\ A\ (\lambda p \to p)\ p\ q$$

- If t is of the form $\mathsf{nd}\ (x \triangleleft y)\ \{z\}\ t$, we have to return a position of the tree $\mathsf{nd}\ (x \triangleleft y)\ v$ where $v\ p :\equiv \gamma_M\ (t\ p)\ (\lambda q \to u\ (\mathsf{inr}\ (p, q)))$. In this context, p has type $\mathsf{Pos}_M\ (\mu_M\ y\ z)$. We first obtain a position of $v\ (\mathsf{pr}_1^\mu\ p)$ as $\mathsf{inr}^\gamma\ (\mathsf{pr}_2^\mu\ p)\ q$ and we finally get a position of $\mathsf{nd}\ (x \triangleleft y)\ v$ as $\mathsf{inr}\ (\mathsf{pr}_1^\mu\ p, \mathsf{inr}^\gamma\ (\mathsf{pr}_2^\mu\ p)\ q)$.

We conclude with the elimination rule for positions of grafted trees:

$$\begin{aligned}
&\mathsf{Pos\text{-}\gamma\text{-}elim} : (M : \mathcal{M})\ \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\} \\
&\quad \to (t : \mathsf{Cns}_{M/}\ (x \triangleleft y))\ (u : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z)) \\
&\quad \to (A : \mathsf{Pos}_{M/}\ (\gamma_M\ t\ u) \to \mathcal{U}) \\
&\quad \to (a_{\mathsf{inl}} : (p : \mathsf{Pos}_{M/}\ t) \to A\ (\mathsf{inl}^\gamma\ p)) \\
&\quad \to (a_{\mathsf{inr}} : (p : \mathsf{Pos}_M\ y)\ (q : \mathsf{Pos}_{M/}\ (u\ p)) \to A\ (\mathsf{inr}^\gamma\ p\ q)) \\
&\quad \to (p : \mathsf{Pos}_{M/}\ (\gamma_M\ t\ u)) \to A\ p
\end{aligned}$$

It is defined by induction on t and on p:

- If t is of the form $\mathsf{lf}\ x$, p is a position of $u\ (\mathsf{pos}_M^\eta\ x)$. We return the term $a_{\mathsf{inr}}\ (\mathsf{pos}_M^\eta\ x)\ p$ which has type $A\ (\mathsf{inr}^\gamma\ (\mathsf{pos}_M^\eta\ x)\ p)$ which is definitionally equal to $A\ p$.
- If t is of the form $\mathsf{nd}\ (x \triangleleft y)\ t$, p is a position of the tree $\mathsf{nd}\ (x \triangleleft y)\ v$ with $v\ p :\equiv \gamma_M\ (t\ p)\ (\lambda q \to u\ (\mathsf{inr}\ (p, q)))$ there are two cases to cover for p:
  - If p is $\mathsf{inl}\ \star$, we return $a_{\mathsf{inl}}\ (\mathsf{inl}\ \star)$ which has the required type.
  - If p is of the form $\mathsf{inr}\ (p, q)$, p is a position of y and q is a position of $v\ p$ which leads us to eliminate q. We set the motive to $B\ q :\equiv A\ (\mathsf{inr}\ (p, q))$ and it remains to define the two cases of the elimination with the two defining equations:

    $$\begin{aligned} b_{\mathsf{inl}}\ q &:\equiv a_{\mathsf{inl}}\ (\mathsf{inr}\ (p, q)) \\ b_{\mathsf{inr}}\ q\ r &:\equiv a_{\mathsf{inr}}\ (\mathsf{pair}^\mu\ p\ q)\ r \end{aligned}$$

    Packaging all this data together, we obtain the following expression:

    $$\mathsf{Pos\text{-}\gamma\text{-}elim}_M\ (t\ p)\ (\lambda q \to u\ (\mathsf{inr}\ (p, q)))\ B\ b_{\mathsf{inl}}\ b_{\mathsf{inr}}\ q$$

μ position operations We conclude the definition of the slice monad with the definition of operations relative to positions of constructors obtained from μ.

Let $x : \mathsf{Idx}_{M/}$, $y : \mathsf{Cns}_{M/}\ x$, and $z : \overrightarrow{\mathsf{Cns}_{M/}}\ y$. For any positions $p : \mathsf{Pos}_{M/}\ y$ and position $q : \mathsf{Pos}_{M/}\ (z\ p)$, there is a position $\mathsf{pair}_{M/}^\mu\ p\ q$ of the tree $\mu_{M/}\ y\ z$ that we define by induction on y and on p:

- If y is of the form $\mathsf{lf}\ x$, $\mathsf{Pos}\ (\mathsf{lf}\ x)$ is the empty type and we are done.

- If y is of the form $\mathsf{nd}\ (x \triangleleft y)\ t$, we are looking for a position of the tree $\gamma_M\ (z\ (\mathsf{inl}\ \star))\ u$ with $u\ p :\equiv \mu_{M/}\ (t\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q)))$. There are two cases to cover for p:
  - If p is $\mathsf{inl}\ \star$, q is a position of $z\ (\mathsf{inl}\ \star)$ and we inject it as $\mathsf{inl}^{\gamma}_{M}\ q$.
  - If p is of the form $\mathsf{inr}\ (p, r)$ where r is a position of $t\ p$ then q is a position of $z\ (\mathsf{inr}\ (p, r))$. We therefore obtain a position of $u\ p$ as $\mathsf{pair}^{\mu}_{M/}\ r\ q$ that we inject it as $\mathsf{inr}^{\gamma}_{M}\ p\ (\mathsf{pair}^{\mu}_{M/}\ r\ q)$.

Conversely, given a position $p : \mathsf{Pos}_{M/}\ (\mu_{M/}\ y\ z)$, we define $\mathsf{pr}^{\mu}_{1\,M/}\ p$ and $\mathsf{pr}^{\mu}_{2\,M/}\ p$ by induction on y. The case of the leaf is quickly eliminated and we are left with the node case of the form $\mathsf{nd}\ (x \triangleleft y)\ t$. The position p is a position of the tree $\gamma_M\ (z\ (\mathsf{inl}\ \star))\ u$ with $u\ p :\equiv \mu_{M/}\ (t\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q)))$ and we have to eliminate it using the elimination rule for γ.

In order to define $\mathsf{pr}^{\mu}_{1\,M/}\ p$, we set the motive to the constant $A\ p :\equiv \mathsf{Pos}_{M/}\ (\mathsf{nd}\ (x \triangleleft y)\ t)$ and define:

$$a_{\mathsf{inl}}\ p :\equiv \mathsf{inl}\ \star$$
$$a_{\mathsf{inr}}\ p\ q :\equiv \mathsf{inr}\ (p, \mathsf{pr}^{\mu}_{1\,M/}\ q)$$

Indeed, the first case concerns the substitution of the root node of $\mathsf{nd}\ (x \triangleleft y)\ t$ and the second one corresponds to a nested substitution somewhere in one of the trees specified by t; hence the recursive call to $\mathsf{pr}^{\mu}_{1\,M/}$.

Finally, to define $\mathsf{pr}^{\mu}_{2\,M/}\ p$, we set the motive to $A\ p :\equiv \mathsf{Pos}_{M/}\ (z\ (\mathsf{pr}^{\mu}_{1\,M/}\ p))$ and define:

$$a_{\mathsf{inl}}\ p :\equiv p$$
$$a_{\mathsf{inr}}\ p\ q :\equiv \mathsf{pr}^{\mu}_{2\,M/}\ q$$

This concludes the presentation of the core of our extension of type theory with polynomial monads. This is enough to internalise higher algebraic structures such as ∞-groupoids as we shall see in Chapter 3.

2.3 Morphisms of monads

The extension of type theory with polynomial monads would not be complete without also introducing their morphisms. More precisely, we will define cartesian morphisms of monads which preserve the type of positions of a constructor as well as their typing. We will only make a modest use of them as their only purpose in this thesis is to reindex opetopic types in Chapter 3.

Similarly to monads, morphisms of monads are a primitive feature of our system enjoying definitional laws. For each monads M and N, we postulate a universe of morphisms from M to N whose elements are seen as codes for morphisms of monads from M to N

$$M \to_m N : \mathcal{U}$$

$$\mathsf{idx}_f^{\rightarrow} : \mathsf{Idx}_M \to \mathsf{Idx}_N$$
$$\mathsf{cns}_f^{\rightarrow} : \{i : \mathsf{Idx}_M\} \to \mathsf{Cns}_M\ i \to \mathsf{Cns}_N\ (\mathsf{idx}_f^{\rightarrow}\ i)$$
$$\mathsf{pos}_f^{\rightarrow} : \{i : \mathsf{Idx}_M\}\ \{c : \mathsf{Cns}_M\ i\} \to \mathsf{Pos}_M\ c \to \mathsf{Pos}_N\ (\mathsf{cns}_f^{\rightarrow}\ c)$$
$$\mathsf{pos}_f^{\leftarrow} : \{i : \mathsf{Idx}_M\}\ \{c : \mathsf{Cns}_M\ i\} \to \mathsf{Pos}_N\ (\mathsf{cns}_f^{\rightarrow}\ c) \to \mathsf{Pos}_M\ c$$

Figure 2.3: The data defining a monad morphism

A morphism of monad $f : M \to_m N$ is characterised by a collection of decoding functions (Figure 2.3). Note that due to the fact our morphisms are cartesian, they have to preserve the positions of constructors as well as their typing. First, morphisms of monads respect the typing of constructors' positions:

$$\mathsf{idx}_f^{\rightarrow}\ (\mathsf{Typ}_M\ c\ p) \equiv \mathsf{Typ}_N\ (\mathsf{cns}_f^{\rightarrow}\ c)\ (\mathsf{pos}_f^{\rightarrow}\ p)$$

Second, the positions operations are inverse to each other:

$$\mathsf{pos}_f^{\leftarrow}\ (\mathsf{pos}_f^{\rightarrow}\ p) \equiv p$$
$$\mathsf{pos}_f^{\rightarrow}\ (\mathsf{pos}_f^{\leftarrow}\ p) \equiv p$$

Finally, monad morphisms respect the monad structure. We have a first set of laws regarding η:

$$\begin{aligned}
&\mathsf{cns}_f^{\rightarrow}\ (\eta_M\ x) && \equiv \eta_N\ (\mathsf{idx}_f^{\rightarrow} x)\\
&\mathsf{pos}_f^{\rightarrow}\ (\mathsf{pos}^{\eta}\ i) && \equiv \mathsf{pos}^{\eta}\ (\mathsf{idx}^{\rightarrow}\ i)\\
&\mathsf{pos}_f^{\leftarrow}\ (\mathsf{pos}^{\eta}\ (\mathsf{idx}^{\rightarrow}\ i)) && \equiv \mathsf{pos}^{\eta}\ i
\end{aligned}$$

There is a second set of laws regarding μ:

$$\begin{aligned}
&\mathsf{cns}_f^{\rightarrow}\ (\mu_M\ c\ d) && \equiv \mu_N\ (\mathsf{cns}_f^{\rightarrow}\ c)\ (\lambda p \to \mathsf{cns}_f^{\rightarrow}\ (d\ (\mathsf{pos}_f^{\leftarrow}\ p)))\\
&\mathsf{pos}_f^{\rightarrow}\ (\mathsf{pair}^{\mu}\ p\ q) && \equiv \mathsf{pair}^{\mu}\ (\mathsf{pos}_f^{\rightarrow}\ p)\ (\mathsf{pos}_f^{\rightarrow}\ q)\\
&\mathsf{pos}_f^{\rightarrow}\ (\mathsf{pr}_1^{\mu}\ p) && \equiv \mathsf{pr}_1^{\mu}\ (\mathsf{pos}_f^{\rightarrow}\ p)\\
&\mathsf{pos}_f^{\rightarrow}\ (\mathsf{pr}_2^{\mu}\ p) && \equiv \mathsf{pr}_2^{\mu}\ (\mathsf{pos}_f^{\rightarrow}\ p)\\
&\mathsf{pos}_f^{\leftarrow}\ (\mathsf{pair}^{\mu}\ p\ q) && \equiv \mathsf{pair}^{\mu}\ (\mathsf{pos}_f^{\leftarrow}\ p)\ (\mathsf{pos}_f^{\leftarrow}\ q)\\
&\mathsf{pos}_f^{\leftarrow}\ (\mathsf{pr}_1^{\mu}\ p) && \equiv \mathsf{pr}_1^{\mu}\ (\mathsf{pos}_f^{\leftarrow}\ p)\\
&\mathsf{pos}_f^{\leftarrow}\ (\mathsf{pr}_2^{\mu}\ p) && \equiv \mathsf{pr}_2^{\mu}\ (\mathsf{pos}_f^{\leftarrow}\ p)
\end{aligned}$$

We now define a monad morphism constructor for each of the monad constructors populating the universe of monads.

2.3.1 Identity morphism

The identity morphism behave as expected, it sends constructors of a monad to themselves. Given a monad M, we postulate a new code for the identity morphism on M:

$$\mathsf{id}_M : M \to_m M$$

Its components are all identities:

1. For any index $i : \mathsf{Idx}_M$,
$$\mathsf{idx}^{\to}_{\mathsf{id}_M}\ i :\equiv i$$
2. For any constructor $c : \mathsf{Cns}_M\ i$,
$$\mathsf{cns}^{\to}_{\mathsf{id}_M}\ c :\equiv c$$
3. For any constructor $c : \mathsf{Cns}_M\ i$ and position $p : \mathsf{Pos}_M\ c$,
$$\mathsf{pos}^{\to}_{\mathsf{id}_M}\ p :\equiv p$$
$$\mathsf{pos}^{\leftarrow}_{\mathsf{id}_M}\ p :\equiv p$$

2.3.2 Slice monad morphisms

A monad morphism $f : M \to_m N$ induces a morphism between their respective slice monads preserving the structure of trees. Given such a morphism f, we postulate a new code for the monad morphism between the slice monads $M/$ and $N/$:

$$f/ : M/ \to_m N/$$

Its components are defined as follows:

1. For any index $(i, c) : \mathsf{Idx}_{M/}$,
$$\mathsf{idx}^{\to}_{f/}\ (i, c) :\equiv (\mathsf{idx}^{\to}_f\ i, \mathsf{cns}^{\to}_f\ c)$$
2. $\mathsf{cns}^{\to}_{f/}$ is defined by induction on $c : \mathsf{Cns}_{M/}\ i$,
$$\begin{aligned}\mathsf{cns}^{\to}_{f/}\ (\mathsf{lf}\ i) &:\equiv \mathsf{lf}\ (\mathsf{idx}^{\to}_f\ i)\\ \mathsf{cns}^{\to}_{f/}\ (\mathsf{nd}\ (x \triangleleft y)\ t) &:\equiv \mathsf{nd}\ (\mathsf{idx}^{\to}_f\ x \triangleleft \mathsf{cns}^{\to}_f\ y)\ (\lambda p \to \mathsf{cns}^{\to}_{f/}\ (t\ (\mathsf{pos}^{\leftarrow}_f\ p)))\end{aligned}$$
3. $\mathsf{pos}^{\to}_{f/}$ is defined by induction on $c : \mathsf{Cns}_{M/}\ i$ and $p : \mathsf{Pos}_{M/}\ c$,
$$\begin{aligned}\mathsf{pos}^{\to}_{f/}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ (\mathsf{inl}\ \star) &:\equiv \mathsf{inl}\ \star\\ \mathsf{pos}^{\to}_{f/}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ (\mathsf{inr}\ (p, q)) &:\equiv \mathsf{inr}\ (\mathsf{pos}^{\to}_f\ p, \mathsf{pos}^{\to}_{f/}\ q)\end{aligned}$$
4. $\mathsf{pos}^{\leftarrow}_{/f}$ is defined by induction on $c : \mathsf{Cns}_{M/}\ i$ and $p : \mathsf{Pos}_{N/}\ (\mathsf{cns}^{\to}_{f/}\ c)$,
$$\begin{aligned}\mathsf{pos}^{\leftarrow}_{f/}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ (\mathsf{inl}\ \star) &:\equiv \mathsf{inl}\ \star\\ \mathsf{pos}^{\leftarrow}_{f/}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ (\mathsf{inr}\ (p, q)) &:\equiv \mathsf{inr}\ (\mathsf{pos}^{\leftarrow}_f\ p, \mathsf{pos}^{\leftarrow}_{f/}\ q)\end{aligned}$$

2.3.3 Pullback monad morphisms

Given a monad morphism $f : M \to_m N$, two type families $A : \mathsf{Fam}_M$ and $B : \mathsf{Fam}_N$ as well as a function $g : \{i : \mathsf{Idx}_M\} \to A\ i \to B\ (\mathsf{idx}^{\to}_f\ i)$, we postulate a new code for the induced morphism between the pullback monads $A^* M$ and $B^* N$:

$$\mathsf{Pb}_{f,g} : A^* M \to_m B^* N$$

This morphism acts by mapping constructors and their decoration. Its components are defined as follows:

1. For any index $(i, x) : \mathsf{Idx}_{A^* M}$,

$$\mathsf{idx}^{\to}_{\mathsf{Pb}_{f,g}}\ (i, x) :\equiv (\mathsf{idx}^{\to}_f\ i, g\ x)$$

2. For any constructor $(c, x) : \mathsf{Cns}_{A^* M}\ (i, y)$,

$$\mathsf{cns}^{\to}_{\mathsf{Pb}_{f,g}}\ (c, x) :\equiv (\mathsf{cns}^{\to}_f\ c, \lambda p \to g\ (x\ (\mathsf{pos}^{\leftarrow}_f\ p)))$$

3. For any constructor $c : \mathsf{Cns}_{A^* M}\ i$ and position $p : \mathsf{Pos}_{A^* M}\ c$,

$$\mathsf{pos}^{\to}_{\mathsf{Pb}_{f,g}}\ p :\equiv \mathsf{pos}^{\to}_f\ p$$

4. For any constructor $c : \mathsf{Cns}_{A^* M}\ i$ and position $p : \mathsf{Pos}_{B^* N}\ (\mathsf{cns}^{\to}_{\mathsf{Pb}_{f,g}}\ c)$,

$$\mathsf{pos}^{\leftarrow}_{\mathsf{Pb}_{f,g}}\ p :\equiv \mathsf{pos}^{\leftarrow}_f\ p$$

2.3.4 Composition of monad morphisms

We finish this section with the definition of the composition of two monad morphisms. Let $f : X \to_m Y$ and $g : Y \to_m Z$, we define their composition

$$g \circ_m f : X \to_m Z$$

Its components are defined as follows:

$$\begin{aligned}
\mathsf{idx}^{\to}_{g \circ_m f}\ i &:\equiv \mathsf{idx}^{\to}_g\ (\mathsf{idx}^{\to}_f\ i) \\
\mathsf{cns}^{\to}_{g \circ_m f}\ c &:\equiv \mathsf{cns}^{\to}_g\ (\mathsf{cns}^{\to}_f\ c) \\
\mathsf{pos}^{\to}_{g \circ_m f}\ p &:\equiv \mathsf{pos}^{\to}_g\ (\mathsf{pos}^{\to}_f\ p) \\
\mathsf{pos}^{\leftarrow}_{g \circ_m f}\ p &:\equiv \mathsf{pos}^{\leftarrow}_f\ (\mathsf{pos}^{\leftarrow}_g\ i)
\end{aligned}$$

2.4 Monad families

Working in dependent type theory, it is natural to generalise polynomial monads to families of polynomial monads indexed by a monad. Roughly, we regard monad families as specifying a collection of constructors indexed by a base

$$
\begin{aligned}
&\mathsf{Idx}{\downarrow_{M\downarrow}} : \mathsf{Idx}_M \to \mathcal{U} \\
&\mathsf{Cns}{\downarrow_{M\downarrow}} : \{i : \mathsf{Idx}_M\} \to \mathsf{Idx}{\downarrow_{M\downarrow}}\ i \to \mathsf{Cns}_M\ i \to \mathcal{U} \\
&\mathsf{Typ}{\downarrow_{M\downarrow}} : \{i : \mathsf{Idx}_M\}\ \{c : \mathsf{Cns}_M\ i\} \\
&\quad \to \mathsf{Cns}{\downarrow_{M\downarrow}}\ i{\downarrow}\ c \to (p : \mathsf{Pos}\ M\ c) \to \mathsf{Idx}{\downarrow_{M\downarrow}}\ (\mathsf{Typ}_M\ c\ p)
\end{aligned}
$$

Figure 2.4: The data of a monad family

constructor. The algebraic structure then grants the ability to compose these constructors resulting in a constructor indexed over the composite of their base constructors. We can also regard monad families as yet another axiomatisation of cartesian morphisms of monads.

We will introduce the dependent counterpart of all the constructions that have already been introduced in this chapter. This is a very laborious albeit straightforward process therefore we will gloss over the details. For any monad $M : \mathcal{M}$, we introduce a universe of monads dependent over M:

$$\mathcal{M}{\downarrow_M} : \mathcal{U}$$

Notation. We will postfix all our dependent constructions with the symbol $\downarrow$. Often, an expression named $x\downarrow$ will depend on some other expression x. For example, when talking about a monad $M\downarrow$ without having made explicit the type of $M\downarrow$, we will assume that it is a monad family indexed by an ordinary monad M.

The polynomial part of a family of polynomial monads $M\downarrow : \mathcal{M}{\downarrow_M}$ is characterised by a set of decoding functions (Figure 2.4) depending over their non-dependent counterpart. Note that the type of positions of a constructor of $M\downarrow$ is the one of the constructor of M it depends on. We understand this data as defining a cartesian morphism of polynomial monads preserving the type of positions of constructors by definition.

$$\begin{aligned}
&\eta{\downarrow_{M\downarrow}} : \{i : \mathsf{Idx}_M\}\ (i{\downarrow} : \mathsf{Idx}{\downarrow_{M\downarrow}}\ i) \to \mathsf{Cns}{\downarrow_{M\downarrow}}\ i{\downarrow}\ (\eta_M\ i)\\
&\mu{\downarrow_{M\downarrow}} : \{i : \mathsf{Idx}_M\}\ \{c : \mathsf{Cns}_M\ i\}\ \{d : \overrightarrow{\mathsf{Cns}_M}\ c\}\\
&\quad \to \{i{\downarrow} : \mathsf{Idx}{\downarrow_{M\downarrow}}\ i\}\ (c{\downarrow} : \mathsf{Cns}{\downarrow_{M\downarrow}}\ i{\downarrow}\ c) \to \overrightarrow{\mathsf{Cns}{\downarrow_{M\downarrow}}}\ c{\downarrow}\ d\\
&\quad \to \mathsf{Cns}{\downarrow_{M\downarrow}}\ i{\downarrow}\ (\mu_M\ c\ d)
\end{aligned}$$

Figure 2.5: The monad structure

Notation. We extend families over the type of indices of a monad to their dependent counterpart. Let $M{\downarrow} : \mathcal{M}{\downarrow_M}$ and $X : \mathsf{Fam}_M$, we define

$$\mathsf{Fam}{\downarrow^X_{M\downarrow}} :\equiv \{i : \mathsf{Idx}_M\} \to \mathsf{Idx}{\downarrow_{M\downarrow}}\ i \to X\ i \to \mathcal{U}$$

We extend the notation $\overrightarrow{X}\ c$ to families of elements of $\mathsf{Fam}{\downarrow^X_{M\downarrow}}$ dependent over the positions of a constructor. Let $M{\downarrow} : \mathcal{M}{\downarrow_M}$, let $X : \mathsf{Fam}_M$, let $X{\downarrow} : \mathsf{Fam}{\downarrow^X_{M\downarrow}}$ be a type family, let $c{\downarrow} : \mathsf{Cns}{\downarrow_{M\downarrow}}\ i{\downarrow}\ c$ be a constructor, and let $x : \overrightarrow{X}\ c$ be a family of elements of X indexed by the positions of c. We define the notation $\overrightarrow{X{\downarrow}}\ c{\downarrow}\ x$ as follows:

$$\overrightarrow{X{\downarrow}}\ c{\downarrow}\ x :\equiv (p : \mathsf{Pos}_M\ c) \to X{\downarrow}\ (\mathsf{Typ}{\downarrow_{M\downarrow}}\ c{\downarrow}\ p)\ (x\ p)$$

As expected, the multiplication and unit of monad families depend on their non dependent counterpart (Figure 2.5). Luckily for us, we do not have to introduce dependent versions of the operations on positions, so we can state the laws that have to be obeyed by the monad operations immediately.

$$\begin{aligned}
&\mu{\downarrow_{M\downarrow}}\ x{\downarrow}\ (\lambda p \to \eta{\downarrow_{M\downarrow}}\ (\mathsf{Typ}{\downarrow_{M\downarrow}}\ x{\downarrow}\ p)) \equiv x{\downarrow}\\
&\mu{\downarrow_{M\downarrow}}\ (\eta{\downarrow_{M\downarrow}}\ x{\downarrow})\ y{\downarrow} \equiv y{\downarrow}\ (\mathsf{pos}^{\eta}_M\ x)\\
&\mu{\downarrow_{M\downarrow}}\ (\mu{\downarrow_{M\downarrow}}\ x{\downarrow}\ y{\downarrow})\ z{\downarrow} \equiv \mu{\downarrow_{M\downarrow}}\ x{\downarrow}\ (\lambda p \to \mu{\downarrow_{M\downarrow}}\ (y{\downarrow}\ p)\ (\lambda q \to z{\downarrow}\ (\mathsf{pair}^{\mu}_M\ p\ q)))
\end{aligned}$$

We are now ready to define the dependent versions of the monad constructors we had introduced at the beginning of this chapter.

2.4.1 The identity monad

We start with the dependent identity monad defined for any type X:

$$\mathsf{Id}{\downarrow_X} : \mathcal{M}{\downarrow_{\mathsf{Id}}}$$

The polynomial part of its definition is straightforward.

$$\begin{aligned}
&\mathsf{Idx}{\downarrow}_{\mathsf{Id}{\downarrow}_X}\ i && :\equiv X\\
&\mathsf{Cns}{\downarrow}_{\mathsf{Id}{\downarrow}_X}\ i{\downarrow}\ c && :\equiv \mathbf{1}\\
&\mathsf{Typ}{\downarrow}_{\mathsf{Id}{\downarrow}_X}\ \{i{\downarrow} = x\}\ c{\downarrow}\ p && :\equiv x
\end{aligned}$$

We do not describe its trivial monad structure. This family has a single unary constructor for any element $x : X$. Moreover, both its input and its output have x for index. When we omit the type X and write $\mathsf{Id}{\downarrow}$, we assume that X is the unit type $\mathbf{1}$.

2.4.2 The pullback monad

Given a monad family $M{\downarrow} : \mathcal{M}{\downarrow}_M$ and a dependent family $X{\downarrow} : \mathsf{Fam}{\downarrow}^X_{M{\downarrow}}$, we introduce the dependent pullback monad $X{\downarrow}^* M{\downarrow} : \mathcal{M}{\downarrow}_{X^* M}$. Its polynomial part is defined as follows:

$$\begin{aligned}
&\mathsf{Idx}{\downarrow}_{X{\downarrow}^* M{\downarrow}}\ (i, x) && :\equiv \textstyle\sum_{(i{\downarrow}:\mathsf{Idx}{\downarrow}_{M{\downarrow}}\ i)}\ X{\downarrow}\ i{\downarrow}\ x\\
&\mathsf{Cns}{\downarrow}_{X{\downarrow}^* M{\downarrow}}\ (i{\downarrow}, x{\downarrow})\ (c, y) && :\equiv \textstyle\sum_{(c{\downarrow}:\mathsf{Cns}{\downarrow}_{M{\downarrow}}\ i{\downarrow}\ c)}\ \overrightarrow{X{\downarrow}}\ c{\downarrow}\ y\\
&\mathsf{Typ}{\downarrow}_{X{\downarrow}^* M{\downarrow}}\ (c{\downarrow}, x{\downarrow})\ p && :\equiv \mathsf{Typ}{\downarrow}_{M{\downarrow}}\ c{\downarrow}\ p
\end{aligned}$$

Once again, its monad structure is a straightforward generalisation of the non-dependent version.

$$\begin{aligned}
&\eta{\downarrow}_{X{\downarrow}^* M{\downarrow}}\ (i{\downarrow}, x{\downarrow}) && :\equiv (\eta{\downarrow}_{M{\downarrow}}\ i{\downarrow}, \eta{\downarrow}\text{-}\mathsf{dec}\ x{\downarrow})\\
&\mu{\downarrow}_{X{\downarrow}^* M{\downarrow}}\ (c{\downarrow}, y{\downarrow})\ d{\downarrow} && :\equiv (\mu{\downarrow}_{M{\downarrow}}\ c{\downarrow}\ (\lambda p \to \mathsf{pr}_1\ (d{\downarrow}\ p)), \lambda p \to \mathsf{pr}_2\ (d{\downarrow}\ (\mathsf{pr}^{\mu}_{1\,M}\ p))\ (\mathsf{pr}^{\mu}_{2\,M}\ p))
\end{aligned}$$

where $\eta{\downarrow}\text{-}\mathsf{dec}_M\ x$ is defined as

$$\eta{\downarrow}\text{-}\mathsf{dec}_M\ x{\downarrow} :\equiv \mathsf{Pos}\text{-}\eta\text{-}\mathsf{elim}_M\ i\ (\lambda p \to X{\downarrow}\ (\mathsf{Typ}{\downarrow}_{M{\downarrow}}\ (\eta{\downarrow}_{M{\downarrow}}\ i{\downarrow})\ p)\ (\eta\text{-}\mathsf{dec}\ x))\ x{\downarrow}$$

2.4.3 The slice monad

Given a monad family $M{\downarrow} : \mathcal{M}{\downarrow}_M$, we introduce the dependent slice monad $M{\downarrow}/ : \mathcal{M}{\downarrow}_{M/}$.

> **Notation.** We extend the notation $\overrightarrow{X}\ (c \blacktriangleleft d)$. Let $M{\downarrow} : \mathcal{M}{\downarrow}_M$, let $X{\downarrow} : \mathsf{Fam}{\downarrow}^X_{M{\downarrow}/}$ be a family indexed by the indices of the monad $M{\downarrow}/$, let $c{\downarrow} : \mathsf{Cns}{\downarrow}_{M{\downarrow}}\ i{\downarrow}\ c$, let $d{\downarrow} : \overrightarrow{\mathsf{Cns}{\downarrow}_{M{\downarrow}}}\ c{\downarrow}\ d$, and let $x : \overrightarrow{X}\ (c \blacktriangleleft d)$. We define the notation $\overrightarrow{X{\downarrow}}\ (c{\downarrow} \blacktriangleleft d{\downarrow})\ x$ as follows:
>
> $$\overrightarrow{X{\downarrow}}\ (c{\downarrow} \blacktriangleleft d{\downarrow})\ x :\equiv (p : \mathsf{Pos}_M\ c) \to X{\downarrow}\ (\mathsf{Typ}{\downarrow}_{M{\downarrow}}\ c{\downarrow}\ p \triangleleft d{\downarrow}\ p)\ (x\ p)$$

As expected, its indices are constructors of $M{\downarrow}$:

$$\mathsf{Idx}{\downarrow_{M\downarrow/}}\ (i, c) :\equiv \textstyle\sum_{(i\downarrow:\mathsf{Idx}_{M\downarrow}\ i)} \mathsf{Cns}{\downarrow_{M\downarrow}}\ i{\downarrow}\ c$$

Its type of constructors is an inductive type with two constructors, one for each constructor of its base type:

$$\begin{aligned}
&\mathsf{If}{\downarrow} : \{i : \mathsf{Idx}_M\}\ (i{\downarrow} : \mathsf{Idx}{\downarrow_{M\downarrow}}\ i) \to \mathsf{Cns}{\downarrow_{M\downarrow/}}\ (i{\downarrow} \triangleleft \eta{\downarrow_{M\downarrow}}\ i{\downarrow})\ (\mathsf{If}\ i)\\
&\mathsf{Nd}{\downarrow} : \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\}\ \{t : \overrightarrow{\mathsf{Cns}_M}\ (y \blacktriangleleft z)\}\\
&\quad \to (x{\downarrow} : \mathsf{Idx}{\downarrow_{M\downarrow}}\ x)\ (y{\downarrow} : \mathsf{Cns}{\downarrow_{M\downarrow}}\ x{\downarrow}\ y)\ \{z{\downarrow} : \overrightarrow{\mathsf{Cns}_{M\downarrow}}\ y{\downarrow}\ z\}\\
&\quad \to (t{\downarrow} : \overrightarrow{\mathsf{Cns}{\downarrow_{M\downarrow}}}\ (y{\downarrow} \blacktriangleleft z{\downarrow})\ t)\\
&\quad \to \mathsf{Cns}{\downarrow_{M\downarrow/}}\ (x{\downarrow} \triangleleft \mu{\downarrow_{M\downarrow}}\ y{\downarrow}\ z{\downarrow})\ (\mathsf{Nd}\ (x \triangleleft y)\ t)
\end{aligned}$$

The typing function projects out the data of the node specified by the position:

$$\begin{aligned}
&\mathsf{Typ}{\downarrow_{M\downarrow/}}\ (\mathsf{Nd}{\downarrow}\ (x{\downarrow} \triangleleft y{\downarrow})\ t{\downarrow})\ (\mathsf{inl}\ \star) &&:\equiv (x{\downarrow} \triangleleft y{\downarrow})\\
&\mathsf{Typ}{\downarrow_{M\downarrow/}}\ (\mathsf{Nd}{\downarrow}\ (x{\downarrow} \triangleleft y{\downarrow})\ t{\downarrow})\ (\mathsf{inr}\ (p, q)) &&:\equiv \mathsf{Typ}{\downarrow_{M\downarrow/}}\ (t{\downarrow}\ p)\ q
\end{aligned}$$

The unit returns a corolla whose sole node is specified by its argument:

$$\eta{\downarrow_{M\downarrow/}}\ (x{\downarrow} \triangleleft y{\downarrow}) :\equiv \mathsf{Nd}{\downarrow}\ (x{\downarrow} \triangleleft y{\downarrow})\ (\lambda p \to \mathsf{If}{\downarrow}\ (\mathsf{Typ}{\downarrow_{M\downarrow}}\ y{\downarrow}\ p))$$

In order to define $\mu_{M\downarrow/}$, we first have to define the grafting operation whose signature is

$$\begin{aligned}
&\gamma{\downarrow_{M\downarrow}} : \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\}\\
&\quad \to \{t : \mathsf{Cns}_{M/}\ (x \triangleleft y)\}\ \{u : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z)\}\\
&\quad \to \{x{\downarrow} : \mathsf{Idx}{\downarrow_{M\downarrow}}\}\ \{y{\downarrow} : \mathsf{Cns}{\downarrow_{M\downarrow}}\ x{\downarrow}\ y\}\ \{z{\downarrow} : \overrightarrow{\mathsf{Cns}{\downarrow_{M\downarrow}}}\ y{\downarrow}\ z\}\\
&\quad \to (t{\downarrow} : \mathsf{Cns}{\downarrow_{M\downarrow/}}\ (x{\downarrow} \triangleleft y{\downarrow})\ t)\ (u{\downarrow} : \overrightarrow{\mathsf{Cns}_{M/}}\ (y{\downarrow} \blacktriangleleft z{\downarrow})\ u)\\
&\quad \to \mathsf{Cns}{\downarrow_{M\downarrow/}}\ (x{\downarrow} \triangleleft \mu{\downarrow_{M\downarrow}}\ y{\downarrow}\ z{\downarrow})\ (\gamma_M\ t\ u)
\end{aligned}$$

It is defined by induction on $t{\downarrow}$:

- If $t{\downarrow}$ is of the form $\mathsf{If}{\downarrow}\ i{\downarrow}$, we simply return $u{\downarrow}\ (\mathsf{pos}^{\eta}\ i)$.
- If $t{\downarrow}$ is of the form $\mathsf{Nd}{\downarrow}\ (x{\downarrow} \triangleleft y{\downarrow})\ t{\downarrow}$, we return $\mathsf{Nd}{\downarrow}\ (x{\downarrow} \triangleleft y{\downarrow})\ t{\downarrow}'$ with
$$t{\downarrow}'\ p :\equiv \gamma{\downarrow_{M\downarrow}}\ (t{\downarrow}\ p)\ (\lambda q \to u\ (\mathsf{pair}^{\mu}\ p\ q))$$

We are now in position to define $\mu{\downarrow_{M\downarrow/}}\ c{\downarrow}\ d{\downarrow}$ by induction on $c{\downarrow}$:

- If $c{\downarrow}$ is of the form $\mathsf{If}{\downarrow}\ i{\downarrow}$, there is nothing to substitute and we return $\mathsf{If}{\downarrow}\ i{\downarrow}$ intact.
- If $c{\downarrow}$ is of the form $\mathsf{Nd}\ (x{\downarrow} \triangleleft y{\downarrow})\ t{\downarrow}$, the root note will be substituted with the tree $d{\downarrow}\ (\mathsf{inl}\ \star)$, in consequence we recursively graft on this tree the result of the substitution of the nodes of $t{\downarrow}$ with the corresponding trees specified by $d{\downarrow}$:
$$\gamma{\downarrow_{M\downarrow}}\ (d{\downarrow}\ (\mathsf{inl}\ \star))\ (\lambda p \to \mu{\downarrow_{M\downarrow/}}\ (t{\downarrow}\ p)\ (\lambda q \to d{\downarrow}\ (\mathsf{inr}\ (p, q))))$$

2.4.4 Dependent sums

We end this chapter with the definition of dependent sums of monads which will be used to define dependent sums of opetopic types. Let $M : \mathcal{M}$ be a monad and let $M{\downarrow} : \mathcal{M}{\downarrow}_M$ be a family of monads indexed by M, we form the new monad

$$\Sigma^m(M, M{\downarrow}) : \mathcal{M}$$

Its different components are defined by the following equations:

$$\begin{aligned}
&\mathsf{Idx}_{\Sigma^m(M,M{\downarrow})} && :\equiv \textstyle\sum_{(i:\mathsf{Idx}_M)} \mathsf{Idx}{\downarrow}_{M{\downarrow}}\ i \\
&\mathsf{Cns}_{\Sigma^m(M,M{\downarrow})}\ (i, i{\downarrow}) && :\equiv \textstyle\sum_{(c:\mathsf{Cns}_M\ i)} \mathsf{Cns}{\downarrow}_{M{\downarrow}}\ i{\downarrow}\ c \\
&\mathsf{Pos}_{\Sigma^m(M,M{\downarrow})}\ (c, c{\downarrow}) && :\equiv \mathsf{Pos}_M\ c \\
&\mathsf{Typ}_{\Sigma^m(M,M{\downarrow})}\ (c, c{\downarrow})\ p && :\equiv (\mathsf{Typ}_M\ c\ p, \mathsf{Typ}{\downarrow}_{M{\downarrow}}\ c{\downarrow}\ p)
\end{aligned}$$

Its monad structure is then defined as follows:

$$\begin{aligned}
&\eta_{\Sigma^m(M,M{\downarrow})}\ (i, i{\downarrow}) && :\equiv (\eta_M\ i, \eta{\downarrow}_{M{\downarrow}}\ i{\downarrow}) \\
&\mu_{\Sigma^m(M,M{\downarrow})}\ (c, c{\downarrow})\ d && :\equiv (\mu_M\ c\ (\lambda p \to \mathsf{pr}_1\ (d\ p)), \mu{\downarrow}_{M{\downarrow}}\ c{\downarrow}\ (\lambda p \to \mathsf{pr}_2\ (d\ p)))
\end{aligned}$$

We do not detail the definitions of the operations acting on positions which simply use the ones of the monad M.

We conclude with the definition of two monad morphisms witnessing that the slice monad and the pullback monad constructors both distribute over the dependent sum. We will need them in Chapter 2. We start with the slice monad. Let $M : \mathcal{M}$ and $M{\downarrow} : \mathcal{M}{\downarrow}_M$, we form the new monad morphism

$$\Sigma^m/(M, M{\downarrow}) : \Sigma^m(M, M{\downarrow})/ \to_m \Sigma^m(M/, M{\downarrow}/)$$

Its components are defined as follows:

1. $\mathsf{idx}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ i$ is defined by the equation

$$\mathsf{idx}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ ((i, i{\downarrow}), (c, c{\downarrow})) :\equiv ((i, c), (i{\downarrow}, c{\downarrow}))$$

2. $\mathsf{cns}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ c$ is defined by induction on c,

$$\begin{aligned}
&\mathsf{cns}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ (\mathsf{lf}\ (i, i{\downarrow})) :\equiv (\mathsf{lf}\ i, \mathsf{lf}{\downarrow}\ i{\downarrow}) \\
&\mathsf{cns}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ (\mathsf{nd}\ ((x, x{\downarrow}) \triangleleft (y, y{\downarrow}))\ t) :\equiv \\
&\quad (\mathsf{nd}\ (x \triangleleft y)\ (\lambda p \to \mathsf{pr}_1\ (\mathsf{cns}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ (t\ p))), \\
&\quad \mathsf{nd}{\downarrow}\ (x{\downarrow} \triangleleft y{\downarrow})\ (\lambda p \to \mathsf{pr}_2\ (\mathsf{cns}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ (t\ p))))
\end{aligned}$$

3. $\mathsf{pos}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ \{c\}\ p$ is defined by induction on c and p,

$$\begin{aligned}
&\mathsf{pos}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ \{\mathsf{nd}\ (x \triangleleft y)\ t\}\ (\mathsf{inl}\ \star) :\equiv \mathsf{inl}\ \star \\
&\mathsf{pos}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ \{\mathsf{nd}\ (x \triangleleft y)\ t\}\ (\mathsf{inr}\ (p, q)) :\equiv \mathsf{inr}\ (p, \mathsf{pos}^{\to}_{\Sigma^m/(M,M{\downarrow})}\ \{t\ p\}\ q)
\end{aligned}$$

4. $\mathsf{pos}^{\leftarrow}_{\Sigma^m/(M,M\downarrow)}\ \{c\}\ p$ is defined by induction on c and p,

$$\mathsf{pos}^{\leftarrow}_{\Sigma^m/(M,M\downarrow)}\ \{\mathsf{nd}\ (x \triangleleft y)\ t\}\ (\mathsf{inl}\ \star) :\equiv \mathsf{inl}\ \star$$
$$\mathsf{pos}^{\leftarrow}_{\Sigma^m/(M,M\downarrow)}\ \{\mathsf{nd}\ (x \triangleleft y)\ t\}\ (\mathsf{inr}\ (p,q)) :\equiv \mathsf{inr}\ (p, \mathsf{pos}^{\leftarrow}_{\Sigma^m/(M,M\downarrow)}\ \{t\ p\}\ q)$$

Similarly, we define the map witnessing that pullback monad constructors distribute over dependent sum constructors. Let $M : \mathcal{M}$ and $M\downarrow : \mathcal{M}\downarrow_M$ be two monads and let $X : \mathsf{Fam}_M$ and $X\downarrow : \mathsf{Fam}\downarrow^X_{M\downarrow}$ be two type families. We form the new monad map

$$\Sigma^{m*}(M, M\downarrow, X, X\downarrow) : \Sigma\downarrow(X, X\downarrow)^*\,(\Sigma^m(M, M\downarrow)) \to_m \Sigma^m(X^*\,M, X\downarrow^*\,M\downarrow)$$

where the family $\Sigma\downarrow(X, X\downarrow) : \mathsf{Fam}_{\Sigma^m(M,M\downarrow)}$ is defined by the equation

$$\Sigma\downarrow(X, X\downarrow)\ (i, i\downarrow) :\equiv \textstyle\sum_{(x:X\ i)} X\downarrow\ i\downarrow\ x$$

Its components are defined as follows:

1. $\mathsf{idx}^{\rightarrow}_{\Sigma^{m*}(M,M\downarrow,X,X\downarrow)}\ i$ is defined by the equation

$$\mathsf{idx}^{\rightarrow}_{\Sigma^{m*}(M,M\downarrow,X,X\downarrow)}\ ((i, i\downarrow), (x, x\downarrow)) :\equiv ((i, x), (i\downarrow, x\downarrow))$$

2. $\mathsf{cns}^{\rightarrow}_{\Sigma^{m*}(M,M\downarrow,X,X\downarrow)}\ c$ is defined by the equation

$$\mathsf{cns}^{\rightarrow}_{\Sigma^{m*}(M,M\downarrow,X,X\downarrow)}\ ((c, c\downarrow), x) :\equiv ((c, \lambda p \to \mathsf{pr}_1\ (x\ p)), (c\downarrow, \lambda p \to \mathsf{pr}_2\ (x\ p)))$$

3. $\mathsf{pos}^{\rightarrow}_{\Sigma^{m*}(M,M\downarrow,X,X\downarrow)}\ p$ is defined by the equation

$$\mathsf{pos}^{\rightarrow}_{\Sigma^{m*}(M,M\downarrow,X,X\downarrow)}\ p :\equiv p$$

4. $\mathsf{pos}^{\leftarrow}_{\Sigma^{m*}(M,M\downarrow,X,X\downarrow)}\ p$ is defined by the equation

$$\mathsf{pos}^{\leftarrow}_{\Sigma^{m*}(M,M\downarrow,X,X\downarrow)}\ p :\equiv p$$

Chapter 3

Opetopic methods in type theory

We leverage the extension of type theory with polynomial monads that we introduced in Chapter 2 to define opetopic types: collections of types whose geometry is governed by opetopes. We will use them to define a range of fully coherent higher algebraic structures such as ∞-groupoids and $(\infty, 1)$-categories. We will then apply opetopic methods in order to establish a number of elementary results about higher algebra in homotopy type theory.

In particular, we will prove in Section 3.3 that our definition of ∞-groupoid is equivalent to a particular instance of Baez and Dolan's definition of a coherent O-algebra (Baez and Dolan 1998).

3.1 Opetopic types

We introduce the notion of opetopic type which is a higher-dimensional collection of elements whose combinatorics is described by opetopes. Opetopic types are parametrised by a base monad M. The collection of 0-cells is given by a first family indexed by the indices of M, that is a type family $X_0 : \mathsf{Fam}_M$. We expect 1-cells to be relations between a configuration of source 0-cells and a single target 0-cell. Such a configuration is given by a constructor of M whose inputs and output are decorated with elements of X_0. These are precisely the constructors of the monad $X_0^* M$. Equivalently, these correspond to the indices of the monad $(X_0^* M)/$. A family of 1-cells is therefore given by a family $X_1 : \mathsf{Fam}_{(X_0^* M)/}$. We can continue this process indefinitely. $n+1$-cells are then relations between a configuration of source n-cells and a single target n-cell.

> **Notation.** The construction of a pullback monad followed by its slice monad will be pervasive in this chapter and we adopt the following notation:
>
> $$M/A :\equiv (A^* M)/$$

Notation. Similarly, we will write f/g instead of the more convoluted $\mathsf{Pb}_{f,g}/$ for the monad morphism of type $M/A \to_m N/B$.

The previous informal description is captured by the following definition.

Definition 3.1.1 (Opetopic type)**.** An opetopic type parametrised by a monad M — or M-opetopic type — is a coinductive sequence of type families which is defined by the following data:

- A type family $X : \mathsf{Fam}_M$.
- An opetopic type parametrised by the monad M/X.

We denote $\mathcal{O}_M$ the type of opetopic types parametrised by the monad M.

Notation. An opetopic type X is therefore an infinite sequence of families $(X_0, X_1, X_2, \dots)$ where X_n is its $(n+1)$th type family for $n : \mathbb{N}$. We denote $X_{>n}$ the opetopic type consisting of the families $(X_{n+1}, X_{n+2}, \dots)$. We call n-cells the elements of X_n.

Our first example of opetopic type is the terminal opetopic type for a monad M, denoted $\mathbf{1}^o_M$, whose type families are the trivial families with one element.

Definition 3.1.2 (Terminal M-opetopic type)**.** Let M be a monad, the terminal M-opetopic type $\mathbf{1}^o_M$ is defined coinductively as follows:

- Its family of objects is the constant family $X_0\ i :\equiv \mathbf{1}$.
- Its opetopic type of relations is $X_{>0} :\equiv \mathbf{1}^o_{M/X_0}$, the terminal opetopic for the monad M/X_0.

If we take M to be the monad Id, we precisely obtain the opetopes as defined in Chapter 1 although monad laws now hold definitionally. For now, an opetopic type is just an infinite collection of cells whose geometry of 1-cells is governed by the base monad M and whose geometry of higher cells is given by the slice construction but which is devoid of algebraic structure. In the next section, we will introduce fibrant opetopic types which are opetopic types whose cells can be coherently composed.

A presentation of opetopic types would not be complete without introducing their morphisms. They are simply defined coinductively as levelwise functions.

Firstly, we need to define the reindexing of an opetopic type along a monad morphism. We make use of morphisms of monads that we introduced in the previous section to that effect.

Definition 3.1.3 (Reindexing of an opetopic type)**.** Given a monad morphism $f : M \to_m N$ and an opetopic type $X : \mathcal{O}_N$, we can reindex X along f and

obtain a M-opetopic type, denoted $f^*\ X$, defined coinductively by the following equations:

$$(f^*\ X)_0\ i :\equiv X_0\ (\mathsf{idx}_f^{\rightarrow}\ i)$$
$$(f^*\ X)_{>0} :\equiv (f/\mathsf{id})^*\ X_{>0}$$

This allows the definition of morphisms of opetopic types.

Definition 3.1.4 (Morphism of opetopic types)**.** Given a monad M and two M-opetopic types X and Y, a morphism from X to Y, denoted $X \to_o Y$, is defined coinductively and consists of the following:

- A function $f : \{i : \mathsf{Idx}_M\} \to X_0\ i \to Y_0\ i$.
- A morphism of opetopic types $X_{>0} \to_o (\mathsf{id}_M/f)^*\ Y_{>0}$.

An equivalence of opetopic types is then a morphism of opetopic types whose levelwise functions are all equivalences.

3.2 Algebras

In this section, we discuss algebras of polynomial monads which are a first step towards the notion of fibrant opetopic type. Informally, we will express algebras as functional relations. For any monad M, an algebra will be defined as the data comprising a type family $X : \mathsf{Fam}_M$ — the carrier of the algebra — along with a relation — its action — witnessing that, for any constructor of M and any decoration of its inputs with elements of X, there exists an element decorating its output. Moreover, the data of this last element along with the element witnessing the relation live in a contractible type. Such a relation is typically defined as a type family in type theory. Its indexing type will be the type of constructors of M whose both inputs and output are decorated with elements of X. These relations are therefore elements of type $\mathsf{Fam}_{M/X}$.

3.2.1 Algebraic structure

We call *pasting diagram* a pair (c, d) of type $[\![M]\!]\ X\ i$ for some monad M, family $X : \mathsf{Fam}_M$, and index $i : \mathsf{Idx}_M$. We regard c as specifying a configuration of inputs and d as providing compatible elements of X for each position of c. We will mostly be concerned with pasting diagrams of some slice monad $M/$. In this case, pasting diagrams are trees of constructors of M decorated with compatible cells. We will extend our notion of algebra from polynomial monads to opetopic types by requiring that any two consecutive cell families (X_n, X_{n+1}) form an algebra for the monad indexing X_n. Given such an opetopic type, we can compose any pasting diagram of n-cells and its composite is unique up to a higher cell which happen to coincide with a propositional equality.

We start with the definition of 0-algebras. A 0-algebra determines an operation of composition of pasting diagrams witnessed by a higher cell, but this composition does not obey any law.

Definition 3.2.1 (0-algebra)**.** Given a monad M, a 0-algebra for M is a type family $X_0 : \mathsf{Fam}_M$ along with a type family $X_1 : \mathsf{Fam}_{M/X_0}$ such that for any index $i : \mathsf{Idx}_M$ and any pasting diagram $(c, x) : [\![M]\!]\ X_0\ i$, the following type is contractible:

$$\textstyle\sum_{(y : X_0\ i)} X_1\ ((i, y) \triangleleft (c, x))$$

In other words, X_1 is a *functional* and *entire* relation. This predicate, being expressed as the contractibility of some type, is a proposition and therefore a property of X_1. This property brings to mind the Segal condition as presented in the bisimplicial setting of Riehl and Shulman (RIEHL and SHULMAN 2017).

This property provides us with the ability to compose pasting diagrams and this composition is witnessed by a *filler*:

$$\begin{aligned}
\alpha_{X_1} &: \{i : \mathsf{Idx}_M\} \to [\![M]\!]\ X_0\ i \to X_0\ i \\
\alpha^{\mathsf{fill}}_{X_1} &: \{i : \mathsf{Idx}_M\} \to (x : [\![M]\!]\ X_0\ i) \to X_1\ ((i, \alpha_{X_1}\ x) \triangleleft x)
\end{aligned}$$

These two functions are simply defined to be the two components of the centre of contraction of the proof that (X_0, X_1) is a 0-algebra.

> **Notation.** Note that we will often implicitly curry functions such as α_X and write $\alpha_X\ c\ x$ instead of $\alpha_X\ (c, x)$.

We will say that a family is a 1-algebra if it satisfies the usual laws of a set-level algebra (MAC LANE 2013, VI.2); that is, the following informal diagrams should commute up to a propositional identity.

$$\begin{array}{ccc}
[\![M]\!]\ [\![M]\!]\ X_0 & \xrightarrow{[\![M]\!]\ \alpha_{X_1}} & [\![M]\!]\ X_0 \\
{\scriptstyle \mu_M\ X_0}\downarrow & & \downarrow{\scriptstyle \alpha_{X_1}} \\
[\![M]\!]\ X_0 & \xrightarrow{\alpha_{X_1}} & X_0
\end{array}
\qquad
\begin{array}{ccc}
X_0 & \xrightarrow{\eta_M\ X_0} & [\![M]\!]\ X_0 \\
& {\scriptstyle \mathsf{id}}\searrow & \downarrow{\scriptstyle \alpha_{X_1}} \\
& & X_0
\end{array}$$

Definition 3.2.2 (1-algebra)**.** Given a monad M and a 0-algebra (X_0, X_1) with $X_0 : \mathsf{Fam}_M$ and $X_1 : \mathsf{Fam}_{M/X_0}$, (X_0, X_1) is a 1-algebra if α_{X_1} is compatible with the multiplication μ_M and the unit η_M of the monad M; that is, it satisfies the following identities:

$$\alpha_{X_1}\ (\mu_M\ c\ d)\ x = \alpha_{X_1}\ c\ (\lambda p \to \alpha_{X_1}\ (d\ p)\ (\lambda q \to x\ (\mathsf{pair}^{\mu}\ p\ q))) \qquad (\alpha\_\mu)$$

$$\alpha_{X_1}\ (\eta_M\ i)\ x = x\ (\mathsf{pos}^{\eta}\ i) \qquad (\alpha\_\eta)$$

We now state a key property of 0-algebras as a characterisation of its action family which will come handy in the rest of this chapter.

Lemma 3.2.3 (Cell characterisation)**.** *Let M be a monad and let (X_0, X_1) be a 0-algebra for M. For any constructor $c : \mathsf{Cns}_M\ i$, any family $x : \overrightarrow{X_0}\ c$, and any element $y : X_0\ i$, a cell $f : X_1\ ((i, z) \triangleleft (c, x))$ implies the existence of a family of equivalences*

$$e_y : X_1\ ((i, y) \triangleleft (c, x)) \simeq (z = y)$$

such that $e_z\ f = \mathsf{refl}$. *In particular, the algebra* (X_0, X_1) *guarantees the existence of the cell* $\alpha^{\mathsf{fill}}_{X_1}\ c\ x : X_1\ ((i, \alpha_{X_1}\ c\ x) \triangleleft (c, x))$.

Proof. The equivalence follows from the HoTT book's Theorem 5.8.2 which states that if a pointed predicate (R, r^*) over a pointed type (A, a^*) is such that the type $\sum_{(a:A)} (R\ a)$ is contractible then the equivalence $e_a : R\ a \simeq (a^* = a)$ holds for any $a : A$; moreover, $e_{a^*}\ r^* = \mathsf{refl}$. In our situation, the pointed predicate in question is $R\ y :\equiv X_1\ ((i, y) \triangleleft (c, x))$ with point $f : R\ z$. As (X_0, X_1) is a 0-algebra, the type $\sum_{(y:X_0\ i)} R\ y$ is contractible which allows us to conclude that there is a family of equivalences $e_y : R\ y \simeq (z = y)$ such that $e_z\ f = \mathsf{refl}$. □

Algebras for slice monads We now turn to algebras for slice monads and consider families $X_0 : \mathsf{Fam}_{M/}$ and $X_1 : \mathsf{Fam}_{M//X_0}$. We will see in Section 3.5 that their algebras correspond to what we call wild M-multicategories, multicategories whose arity of morphisms is parametrised by constructors of the monad M but whose laws are not necessarily coherent.

Notation. Given a monad M, a family $X : \mathsf{Fam}_{M/}$, a cell $t : X\ (x \triangleleft y)$, and a family of cells $u : \overrightarrow{X}\ (y \blacktriangleleft z)$, we will make great use of the depth-2 pasting diagram whose root node is decorated with the cell t and whose family of nodes grafted on the root node is decorated with the cells specified by u. It is defined as the pair $(c, d) : [\![M/]\!]\ X\ x$ where c is the depth-2 tree $\mathsf{nd}\ (x \triangleleft y)\ (\lambda p \to \eta_{M/}\ (\mathsf{Typ}_M\ y\ p \triangleleft z\ p))$ and where d is a family decorating the nodes of c defined by the following equations:

$$\begin{aligned} d\ (\mathsf{inl}\ \star) &:\equiv t \\ d\ (\mathsf{inr}\ (p, \mathsf{inl}\ \star)) &:\equiv u\ p \end{aligned}$$

We denote this pasting diagram $\theta_{t,u}$.

When (X_0, X_1) is a 0-algebra, we define a biased composition of 0-cells $\mu^{\alpha}_{X_1}$ along with its filler $\mu^{\alpha^{\mathsf{fill}}}_{X_1}$:

$$\begin{aligned} \mu^{\alpha}_{X_1} &: \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\} \\ &\to (t : X_0\ (x \triangleleft y))\ (u : \overrightarrow{X_0}\ (y \blacktriangleleft z)) \\ &\to X_0\ (x \triangleleft \mu_M\ y\ z) \\ \mu^{\alpha^{\mathsf{fill}}}_{X_1} &: \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\} \\ &\to (t : X_0\ (x \triangleleft y))\ (u : \overrightarrow{X_0}\ (y \blacktriangleleft z)) \\ &\to X_1\ (((x \triangleleft \mu_M\ y\ z), \mu^{\alpha}_{X_1}\ t\ u) \triangleleft \theta_{t,u}) \end{aligned}$$

These operations are defined by the following equations:

$$\mu^{\alpha}_{X_1}\ t\ u :\equiv \alpha_{X_1}\ \theta_{t,u}$$
$$\mu^{\alpha^{\mathsf{fill}}}_{X_1}\ t\ u :\equiv \alpha^{\mathsf{fill}}_{X_1}\ \theta_{t,u}$$

We will see that under a certain condition, this composition is unital and associative up to a propositional identity.

We define a second operation η^{α} which associates to any index $x : \mathsf{Idx}_M$, a unary cell of type $X_0\ (x \triangleleft \eta_M\ x)$ along with its filler:

$$\eta^{\alpha}_{X_1} : (x : \mathsf{Idx}_M) \to X_0\ (x \triangleleft \eta_M\ x)$$
$$\eta^{\alpha^{\mathsf{fill}}}_{X_1} : (x : \mathsf{Idx}_M) \to X_1\ (((x \triangleleft \eta_M\ x), \eta^{\alpha}_{X_1}\ x) \triangleleft (\mathsf{lf}\ x, \bot\text{-}\mathsf{elim}))$$

These operations are defined by the following equations:

$$\eta^{\alpha}_{X_1}\ x :\equiv \alpha_{X_1}\ (\mathsf{lf}\ x)\ \bot\text{-}\mathsf{elim}$$
$$\eta^{\alpha^{\mathsf{fill}}}_{X_1}\ x :\equiv \alpha^{\mathsf{fill}}_{X_1}\ (\mathsf{lf}\ x)\ \bot\text{-}\mathsf{elim}$$

The unary cells $\eta^{\alpha}_{X_1}\ x$ will act as units for $\mu^{\alpha}_{X_1}$ when (X_0, X_1) is a 1-algebra.

Algebraic laws We now study how, given a 0-algebra (X_0, X_1), a second 0-algebra (X_1, X_2) forces (X_0, X_1) to be a 1-algebra.

> **Methodology.** We will quite often want to prove that two cells $x, y : X_0\ i$ are equal. Most of the time, we will use the fact that we have a family $X_1 : \mathsf{Fam}_{M/X_0}$ such that (X_0, X_1) is a 0-algebra and that one of x or y — say x — is accompanied by a filler of type $X_1\ ((i, x) \triangleleft (c, z))$. That (X_0, X_1) is a 0-algebra means that we can deduce an identity $\alpha_{X_1}\ c\ z = x$ from Lemma 3.2.3. Therefore, proving $x = y$ amounts to finding another filler of type $X_1\ ((i, y) \triangleleft (c, z))$ which allows, in turn, to deduce an identity $\alpha_{X_1}\ c\ z = y$ from which we finally deduce the identity $x = y$.
> The difficulty will therefore lie in finding this second filler. When one has, in addition, a family $X_2 : \mathsf{Fam}_{M/X_0/X_1}$ such that (X_1, X_2) is a 0-algebra, this filler will often be obtained as the composition of a pasting diagram of 1-cells.

Theorem 3.2.4. *Given a monad M and families $X_0 : \mathsf{Fam}_M$, $X_1 : \mathsf{Fam}_{M/X_0}$, and $X_2 : \mathsf{Fam}_{M/X_0/X_1}$, if (X_0, X_1) and (X_1, X_2) are 0-algebras then (X_0, X_1) is a 1-algebra.*

Proof. We start to prove α-μ.

Let $i : \mathsf{Idx}_M$, $c : \mathsf{Cns}_M\ i$, $d : \overrightarrow{\mathsf{Cns}_M}\ c$, and a family $x : \overrightarrow{X_0}\ (\mu_M\ c\ d)$. We want to establish the following identity in $X_0\ i$:

$$\alpha_{X_1}\ (\mu_M\ c\ d)\ x = \alpha_{X_1}\ c\ (\lambda p \to \alpha_{X_1}\ (d\ p)\ (\lambda q \to x\ (\mathsf{pair}^{\mu}\ p\ q)))$$

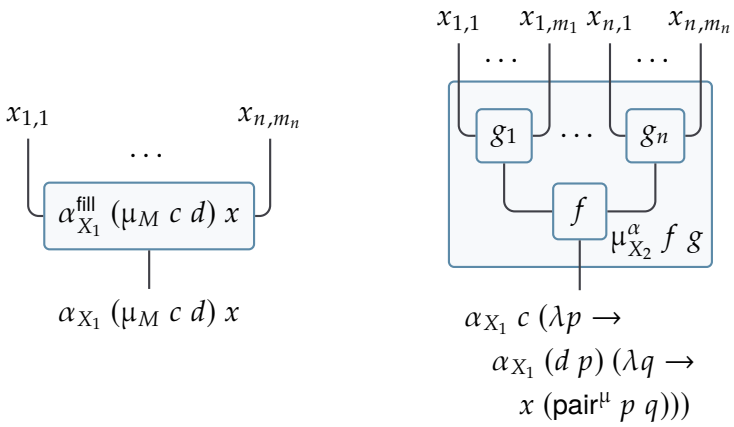


Figure 3.1: The two ways to compose $(\mu_M\ c\ d, x)$

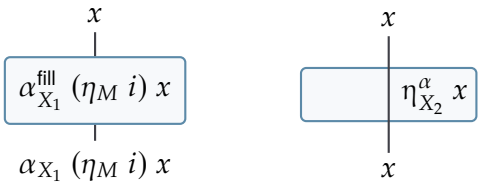


Figure 3.2: Cells witnessing the unit law

Let us denote y the right-hand side. According to Lemma 3.2.3, establishing this identity amounts to defining a cell of type

$$X_1\ ((i, y) \triangleleft (\mu_M\ c\ d, x))$$

We define it as the composition of the pasting diagram $\theta_{f,g}$ represented on the right part of Figure 3.1 and that we now describe.

We define the family g as the fillers witnessing the compositions of the pasting diagrams $(d\ p, \lambda q \to x\ (\mathsf{pair}^{\mu}\ p\ q))$ for any position $p : \mathsf{Pos}_M\ c$, hence:

$$g\ p :\equiv \alpha^{\mathsf{fill}}_{X_1}\ (d\ p)\ (\lambda q \to x\ (\mathsf{pair}^{\mu}\ p\ q))$$

We then define f as the filler for the composition of the pasting diagram made of the targets of the family g:

$$f :\equiv \alpha^{\mathsf{fill}}_{X_1}\ c\ (\lambda p \to \alpha_{X_1}\ (d\ p)\ (\lambda q \to x\ (\mathsf{pair}^{\mu}\ p\ q)))$$

One can finally check that $\mu^{\alpha}_{X_2}\ f\ g$ has the required type

$$X_1\ ((i, y) \triangleleft (\mu_M\ c\ d, x))$$

which concludes the proof of α-μ.

We now prove α-η. Let $i : \mathsf{Idx}_M$ and $x : \overrightarrow{X_0}\ (\eta_M\ i)$, we want to establish the following identity in $X_0\ i$:

$$\alpha_{X_1}\ (\eta_M\ i)\ x = x\ (\mathsf{pos}^{\eta}\ i)$$

According to Lemma 3.2.3, establishing this identity amounts to defining a cell of type

$$X_1\ ((i, x\ (\mathsf{pos}^\eta\ i)) \triangleleft (\eta_M\ i, x))$$

The situation is represented on Figure 3.2. Using the algebraicity of X_2, we first obtain the cell of type

$$X_1\ ((i, x\ (\mathsf{pos}^\eta\ i)) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ (x\ (\mathsf{pos}^\eta\ i))))$$

as $\eta^\alpha_{X_2}\ (i, x\ (\mathsf{pos}^\eta\ i))$. We obtain the desired cell by transporting the previous cell along a path $\eta\text{-}\mathsf{dec}\ (x\ (\mathsf{pos}^\eta\ i)) = x$ using the elimination principle for $\mathsf{Pos}_M\ (\eta_M\ i)$. This concludes the proof of α_η. □

We now prove that, if the base monad is a slice monad of the form $M/$, the biased composition $\mu^\alpha_{X_1}$ is associative and unital.

Theorem 3.2.5. *Given a monad M and families $X_0 : \mathsf{Fam}_{M/}$, $X_1 : \mathsf{Fam}_{M//X_0}$, and $X_2 : \mathsf{Fam}_{M//X_0/X_1}$, if (X_0, X_1) and (X_1, X_2) are 0-algebras, then $\mu^\alpha_{X_1}$ is associative and unital with unit $\eta^\alpha_{X_1}$:*

$$\mu^\alpha_{X_1}\ f\ (\lambda p \to \eta^\alpha_{X_1}\ (\mathsf{Typ}_M\ y\ p)) = f \qquad (\mu^\alpha_{X_1}\text{-unit-r})$$

$$\mu^\alpha_{X_1}\ (\eta^\alpha_{X_1}\ x)\ f = f\ (\mathsf{pos}^\eta\ x) \qquad (\mu^\alpha_{X_1}\text{-unit-l})$$

$$\mu^\alpha_{X_1}\ f\ (\lambda p \to \mu^\alpha_{X_1}\ (g\ p)\ (\lambda q \to h\ (\mathsf{pair}^\mu\ p\ q))) = \mu^\alpha_{X_1}\ (\mu^\alpha_{X_1}\ f\ g)\ h \qquad (\mu^\alpha_{X_1}\text{-assoc})$$

Proof. We start with the proof of $\mu^\alpha_{X_1}$-unit-r. Given a cell $f : X_0\ (x \triangleleft y)$, we want to prove the following equation:

$$\mu^\alpha_{X_1}\ f\ (\lambda p \to \eta^\alpha_{X_1}\ (\mathsf{Typ}_M\ y\ p)) = f$$

We use the fact that (X_0, X_1) is a 0-algebra which implies the contractibility of the following type:

$$\textstyle\sum_{(g : X_0\ (x \triangleleft y))} X_1\ (((x \triangleleft y), g) \triangleleft \eta_{X_0^*(M/)}\ ((x \triangleleft y), f))$$

The pair $(f, \eta^\alpha_{X_2}((x \triangleleft y), f))$ lives in this type and it remains to find a second pair whose first component is therefore $\mu^\alpha_{X_1}\ f\ (\lambda p \to \eta^\alpha_{X_1}\ (\mathsf{Typ}_M\ y\ p))$ — the left-hand side of the equation that we denote l — in order to establish the wanted identity. We obtain the second component — denoted β on Figure 3.3 — as the composition of the pasting diagram that we now describe and which will consist in filling the units grafted on f with leaves. This pasting diagram must have target l and we choose its root node to be the cell $\mu^{\alpha^{\mathsf{fill}}}_{X_1}\ f\ (\lambda p \to \eta^\alpha_{X_1}\ (\mathsf{Typ}_M\ y\ p))$. The source pasting diagram of this cell is $\theta_{f, \lambda p \to \eta^\alpha_{X_1}\ (\mathsf{Typ}_M\ y\ p)}$. We define a family of trees to graft on its positions by induction. In the case of the root node, f, we just plug a leaf $\mathsf{lf}\ ((x \triangleleft y), f)$ as we want to keep f intact in the first diagram. In the case of the other positions p of y, the source of f, we graft unit fillers corollas

$$\eta^{\alpha^{\mathsf{fill}}}_{X_1}\ (\mathsf{Typ}_M\ y\ p) : X_1\ ((((\mathsf{Typ}_M\ y\ p) \triangleleft \eta_M\ (\mathsf{Typ}_M\ y\ p)), \eta^\alpha_{X_1}\ (\mathsf{Typ}_M\ y\ p)) \\ \triangleleft (\mathsf{lf}\ (\mathsf{Typ}\ M\ y\ p), \bot\text{-elim}))$$

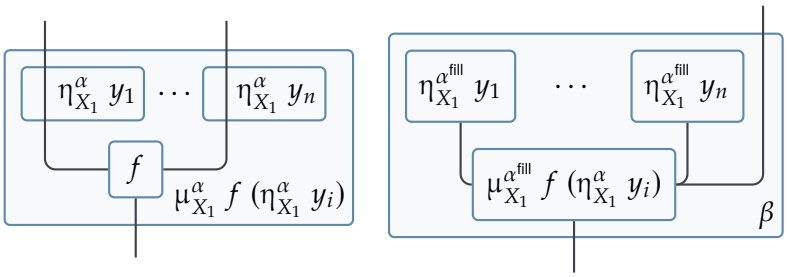


Figure 3.3: $\mu^{\alpha}_{X_1}$ is right unital

This corresponds to filling the units in the first pasting diagram with leaves. We compose the resulting pasting diagram using the fact that (X_1, X_2) is a 0-algebra and obtain a cell of type

$$X_1\ ((x \triangleleft y, l) \triangleleft \mu_{X_0^*(M/)}\ (\theta_{f,\lambda p \to \eta^{\alpha}_{X_1}\ (\mathsf{Typ}_M\ y\ p)})\ m)$$

where m is the family of pasting diagrams indexed by the positions of

$$\theta_{f,\lambda p \to \eta^{\alpha}_{X_1}\ (\mathsf{Typ}_M\ y\ p)}$$

determined by the source pasting diagrams of the family of cells grafted on $\mu^{\alpha^{\mathsf{fill}}}_{X_1}\ f\ (\lambda p \to \eta^{\alpha}_{X_1}\ (\mathsf{Typ}_M\ y\ p))$. It then remains to transport this cell along the easy to establish path

$$\mu_{X_0^*(M/)}\ (\theta_{f,\lambda p \to \eta^{\alpha}_{X_1}\ (\mathsf{Typ}_M\ y\ p)})\ m = \eta_{X_0^*(M/)}\ ((x \triangleleft y), f)$$

We now prove $\mu^{\alpha}_{X_1}$-unit-l using a similar argument. Given an index $x : \mathsf{Idx}_M$, a family of constructors indexed by a unit $y : \overrightarrow{\mathsf{Cns}_M}\ (\eta_M\ x)$, and a family $f : \overrightarrow{X_0}\ (\eta_M\ x \blacktriangleleft y)$, we want to establish the following identity

$$\mu^{\alpha}_{X_1}\ (\eta^{\alpha}_{X_1}\ x)\ f = f\ (\mathsf{pos}^{\eta}\ x)$$

We denote l its left-hand side. We once again use the fact that (X_0, X_1) is a 0-algebra and this time we are looking for a cell of type

$$X_1\ (((x \triangleleft y\ (\mathsf{pos}^{\eta}\ x)), \mu^{\alpha}_{X_1}\ (\eta^{\alpha}_{X_1}\ x)\ f) \triangleleft (\eta_{X_0^*(M/)}\ (x \triangleleft y\ (\mathsf{pos}^{\eta}\ x)), f\ (\mathsf{pos}^{\eta}\ x)))$$

We obtain this cell — denoted β on Figure 3.4 — as the composition of the pasting diagram that we now describe and which will consist in filling the unit on which is grafted f with a leaf. This pasting diagram must have target l and we choose its root node to be the cell $\mu^{\alpha^{\mathsf{fill}}}_{X_1}\ (\eta^{\alpha}_{X_1}\ x)\ f$. Its source pasting diagram is $\theta_{(\eta^{\alpha}_{X_1}\ x),f}$ and we define a family of trees to graft on its positions by induction. In the case of the root node, $\eta^{\alpha}_{X_1}\ x$, we graft the unit filler corolla

$$\eta^{\alpha^{\mathsf{fill}}}_{X_1}\ x : X_1\ (((x \triangleleft \eta_M\ x), \eta^{\alpha}_{X_1}\ x) \triangleleft (\mathsf{lf}\ x, \bot\text{-}\mathsf{elim}))$$

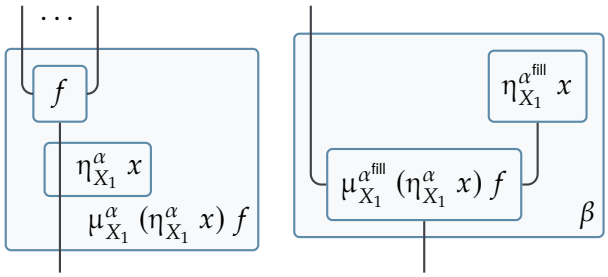


Figure 3.4: $\mu^{\alpha}_{X_1}$ is left unital

and, in the case of the position of f, we just plug a leaf $\mathsf{lf}\ ((x \triangleleft y\ (\mathsf{pos}^{\eta}\ x)), f\ (\mathsf{pos}^{\eta}\ x))$ as we want to keep f intact in the first diagram.

The family of pasting diagrams that we just specified implicitly determines a family of source pasting diagrams that we denote m. We compose this pasting diagram using the fact that (X_1, X_2) is 0-algebra and obtain a cell of type

$$X_1\ ((x \triangleleft y, l) \triangleleft \mu_{X_0^*(M/)}\ (\theta_{(\eta^{\alpha}_{X_1}\ x),f})\ m)$$

It then remains to transport this cell along the easily established path

$$\mu_{X_0^*(M/)}\ (\theta_{(\eta^{\alpha}_{X_1}\ x),f})\ m = \eta_{X_0^*(M/)}\ ((x \triangleleft y\ (\mathsf{pos}^{\eta}\ x)), f\ (\mathsf{pos}^{\eta}\ x))$$

We finally prove the associativity $\mu^{\alpha}_{X_1}$-assoc. Given $f : X_0\ (x \triangleleft y)$, $g : \overrightarrow{X_0}\ (y \blacktriangleleft z)$, and $h : \overrightarrow{X_0}\ (\mu_M\ y\ z \blacktriangleleft t)$, we want to establish the following identity:

$$\mu^{\alpha}_{X_1}\ f\ (\lambda p \to \mu^{\alpha}_{X_1}\ (g\ p)\ (\lambda q \to h\ (\mathsf{pair}^{\mu}\ p\ q))) = \mu^{\alpha}_{X_1}\ (\mu^{\alpha}_{X_1}\ f\ g)\ h$$

Establishing this identity amounts to finding two cells sharing the same source and whose target are the two sides of the identity that we denote $f(gh)$ and $(fg)h$ then concluding using that fact that (X_0, X_1) is a 0-algebra. The two cells in question correspond to the two different ways to multiply the cells f, g, and h and correspond to the cells β and γ on Figure 3.5. We start with the case of the cell β. We obtain it as the composition of the pasting diagram whose root node is $\mu^{\alpha^{\mathsf{fill}}}_{X_1}\ f\ gh$ where gh is the family

$$gh\ p :\equiv \mu^{\alpha}_{X_1}(g\ p)\ (\lambda q \to h\ (\mathsf{pair}^{\mu}\ p\ q))$$

The source of this node is the pasting diagram $\theta_{f,}\ gh$ and we specify the trees to graft on its positions by induction. In the case of the root node, f, we graft a leaf, while on positions p of y, we graft corollas

$$\mu^{\alpha^{\mathsf{fill}}}_{X_1}(g\ p)\ (\lambda q \to h\ (\mathsf{pair}^{\mu}\ p\ q))$$

This concludes the description of the pasting diagram whose composition is the cell β.

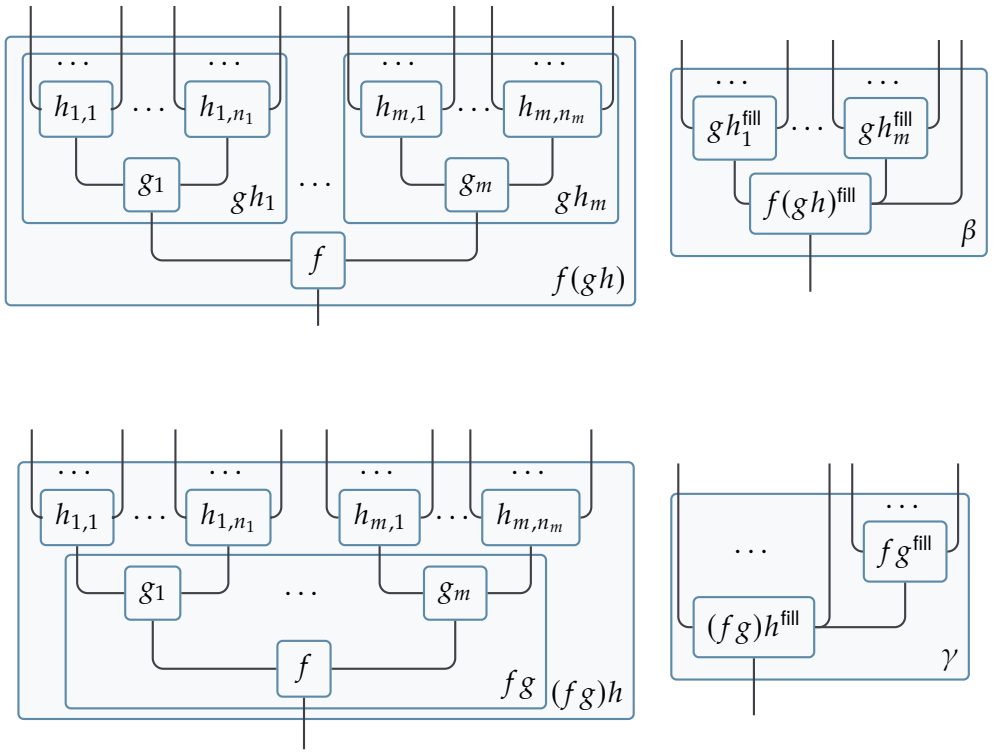


Figure 3.5: The two different ways to associate fgh

We now turn to the case of the cell γ. It is defined as the composition of the pasting diagram that we now describe. Its root node is $\mu_{X_1}^{\alpha^{\mathsf{fill}}}\ (f\,g)\ h$ where $f\,g :\equiv \mu_{X_1}^{\alpha}\ f\ g$. The source pasting diagram is therefore $\theta_{(f\,g),h}$ and we specify the pasting diagrams to graft on its positions by induction. In the case of the root node, $f\,g$, we graft the corolla $\mu_{X_1}^{\alpha^{\mathsf{fill}}}\ f\ g$. As for the other positions p of $\mu_M\ y\ z$, we simply graft leaves $\mathsf{lf}\ ((\mathsf{Typ}_M\ (\mu_M\ y\ z)\ p \triangleleft t\ p), h\ p)$. This concludes the description of the pasting diagram whose composition is the cell γ.

The two cells β and γ both have a source propositionally equal to the pasting diagram denoting the unbiased composition of f, g, and h which allows us to conclude the proof.

□

We call the algebraic structure thus described a wild M-multicategory which is the subject of Section 3.5.

We finish this section with a last technical result concerning the interaction of the algebra with the grafting operation γ.

Lemma 3.2.6. *Let M be a monad and let families $X_0 : \mathsf{Fam}_{M/}$ and $X_1 : \mathsf{Fam}_{M//X_0}$ such that (X_0, X_1) is a 1-algebra. For any tree $t : \mathsf{Cns}_{M/}\ (x \triangleleft y)$, family $u : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z)$,*

and family $v : \overrightarrow{X_0}\ (\gamma_M\ t\ u)$,

$$\alpha_{X_1}\ (\gamma_M\ t\ u)\ v = \mu^{\alpha}_{X_1}\ (\alpha_{X_1}\ t\ (\lambda p \to v\ (\mathsf{inl}^{\gamma}\ p)))\ (\lambda p \to \alpha_{X_1}\ (u\ p)\ (\lambda q \to v\ (\mathsf{inr}^{\gamma}\ p\ q)))$$

Proof. We notice that $\gamma_M\ t\ u$ is equal to $\mu_{M/}\ \theta_{t,u}$ where $\theta_{t,u}$ is seen as a depth-2 pasting diagram whose root node is decorated with t and whose children nodes are decorated with elements of the family u. This allows to conclude using the identity α_μ as (X_0, X_1) is a 1-algebra. □

The following simple corollary follows.

Corollary 3.2.7. *Let M be a monad and let families $X_0 : \mathsf{Fam}_{M/}$ and $X_1 : \mathsf{Fam}_{M//X_0}$ such that (X_0, X_1) is a 1-algebra. For any constructor $y : \mathsf{Cns}_M\ x$, any family of trees $t : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z)$, and any family $v : \overrightarrow{X_0}\ (\mathsf{nd}\ (x \triangleleft y)\ t)$,*

$$\alpha_{X_1}\ (\mathsf{nd}\ (x \triangleleft y)\ t)\ v = \mu^{\alpha}_{X_1}\ (v\ (\mathsf{inl}\ \star))\ (\lambda p \to \alpha_{X_1}\ (t\ p)\ (\lambda q \to v\ (\mathsf{inr}\ (p, q))))$$

Proof. We notice that the constructor $\mathsf{nd}\ (x \triangleleft y)\ t$ is actually equal to $\gamma_{X_1}\ (\eta_{M/}\ (x \triangleleft y))\ t$ which allows us to apply Lemma 3.2.6. □

3.2.2 Fibrant opetopic types

In light of the precedent section, we can expect a fully coherent M-algebra to be an infinite sequence of families $(X_0, X_1, \dots)$ such that $X_{n+1} : \mathsf{Fam}_{M/\dots/X_n}$ — in other words, an opetopic type — and such that any pair (X_n, X_{n+1}) is a 0-algebra. We call these opetopic types *fibrant*.

Definition 3.2.8 (Fibrant opetopic type)**.** Given a monad M and a M-opetopic type X, we say that this opetopic type is fibrant if it satisfies the following coinductive *property*:

- The pair (X_0, X_1) is a 0-algebra.
- The opetopic type $X_{>0}$ is fibrant.

A fibrant opetopic type is to be seen as a collection of n-dimensional cells for all $n : \mathbb{N}$ which can be composed. Moreover, the operations of composition are fully coherent.

We are now equipped to define our next important class of examples of opetopic types.

Definition 3.2.9 (∞-groupoid)**.** An ∞-groupoid is an Id-opetopic type which is fibrant.

Weakening this definition to only require $X_{>0}$ to be fibrant allows having 1-cells which are not necessarily invertible which is precisely the definition of an $(\infty, 1)$-category. We will not deal with (∞, n)-categories for $n > 1$ so we just call them ∞-categories for short.

Definition 3.2.10 (∞-category)**.** An ∞-category is an Id-opetopic type X such that $X_{>0}$ is fibrant.

We will prove elementary results on ∞-groupoids and ∞-categories in order to showcase opetopic methods later on in this chapter.

3.3 Algebraic characterisation

The property of fibrancy very elegantly embodies both an algebraic structure on an opetopic type stated in terms of the existence of enough cells and a property of univalence connecting this structure with the higher structures endowing its families of cells granted by their identity types. This situation is reminiscent of complete Segal spaces. In this section we make explicit the former structure, that we call coherent M-algebra structure, and the property of univalence then show that their conjunction is equivalent to the fibrancy property. Coherent M-algebras correspond to Baez and Dolan's coherent O-algebras (Baez and Dolan 1998) with the exception that we ask for all cells to be invertible in our case.

Notation. Let M be a monad whose types of positions have decidable equality and let $X : \mathsf{Fam}_M$ be a type family. Suppose that we have a constructor $c : \mathsf{Cns}_M\ i$ and a family $x : \overrightarrow{X}\ c$. Given a position $p : \mathsf{Pos}_M\ c$ and an element $y : X\ (\mathsf{Typ}_M\ c\ p)$, we denote $x[y/p]$ the function of type $\overrightarrow{X}\ c$ which takes the same values as x excepted at the position p where it is equal to y:

$$x[y/p]\ q :\equiv \begin{cases} y & \text{if } p = q \\ x\ q & \text{otherwise} \end{cases}$$

In the rest of this section, we assume that whenever we introduce a monad M, its types of positions have decidable equality. This property is preserved by the pullback monad constructor as positions of constructors of $X^* M$ are defined to be those of the underlying constructors of M. Likewise, the slice monad constructor preserves this property.

Notation. We will sometimes use ellipses in place of easily inferable indices. For example, we may write $\eta^{\alpha}_{X_1}\ (\ldots, f)$ instead of $\eta^{\alpha}_{X_1}\ (i, f)$ if i is actually too large and can be inferred from f in order not to clutter a proof.

We define the notions of target and source universality which are mutually dependent and which are central to the definition of coherent M-algebra in Definition 3.3.1 and Definition 3.3.2.

Definition 3.3.1 (Target universality)**.** Let X be a M opetopic type. A cell $f : X_1\ ((i, x) \triangleleft (c, y))$ is target universal in X if for every cell sharing the same source $g : X_1\ ((i, z) \triangleleft (c, y))$, there exists a unary cell

$$h : X_1\ ((i, z) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ x))$$

and a filler

$$h^{\mathsf{fill}} : X_2\ ((\ldots, g) \triangleleft \theta_{h,f})$$

Moreover, h^{fill} is target universal in $X_{>0}$ and source universal in $X_{>0}$ at position $\mathsf{inl}\ \star$, the position of h.

A target universal cell witnesses that its target is, in some sense, a good composite of its source as any cell sharing the same source factorises through it uniquely. Note that this is an additional structure and not a proposition.

Similarly, we define the notion of source universality.

Definition 3.3.2 (Source universality)**.** Let X be a M opetopic type. A cell $f : X_1\ ((i, x) \triangleleft (c, y))$ is source universal in X at position $p : \mathsf{Pos}_M\ c$ if for any cell $z : X_0\ (\mathsf{Typ}_M\ c\ p)$ and cell $g : X_1\ ((i, x) \triangleleft (c, y[z/p]))$, there exists a unary cell

$$h : X_1\ ((\mathsf{Typ}_M\ c\ p, y\ p) \triangleleft (\eta_M\ (\mathsf{Typ}_M\ c\ p), \eta\text{-}\mathsf{dec}\ z))$$

and a filler

$$h^{\mathsf{fill}} : X_2\ ((\ldots, g) \triangleleft \theta_{f,h})$$

where $\theta_{f,h}$ denotes the pasting diagram with root node f on which are grafted leaves expected at position p where we graft h. Moreover h^{fill} is target universal in $X_{>0}$ and source universal in $X_{>0}$ at position $\mathsf{inr}\ (p, \mathsf{inl}\ \star)$, the position of h.

Note that, once again, being source universal is an additional structure.

We now define the structure of coherent M-algebra on a M-opetopic type. This definition is heavily inspired from Baez and Dolan's coherent O-algebras with the exception that we ask for all cells to be target universal in our case corresponding to the fact that we only deal with invertible cells.

Definition 3.3.3 (Coherent M-algebra structure)**.** A M-opetopic type X has a structure of coherent M-algebra if it satisfies the following conditions:

- Every pasting diagram $(c, x) : [\![M]\!]\ X_0\ i$ has a composite $\alpha_X\ c\ x : X_0\ i$ and a filler $\alpha_X^{\mathsf{fill}}\ c\ x : X_1\ ((i, \alpha_X\ c\ x) \triangleleft (c, x))$.
- Every cell $X_1\ ((i, x) \triangleleft (c, y))$ is target universal.
- The opetopic type $X_{>0}$ has a structure of coherent M/X_0-algebra.

Note that we reused the symbols α and α^{fill}, the context should disambiguate whether they refer to the fibrant structure or to the structure of coherent M-algebra. This also applies to the operations η^α, μ^α, and their fillers that we now introduce.

This definition allows us to recover the biased operators that we defined earlier, among them:

$$\begin{aligned}
&\eta_X^\alpha : (x : \mathsf{Idx}_{X_0^* M}) \to X_1\ (x \triangleleft \eta_{X_0^* M}\ x)\\
&\mu_X^\alpha : \{x : \mathsf{Idx}_{X_0^* M}\}\ \{y : \mathsf{Cns}_{X_0^* M}\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_{X_0^* M}}\ y\}\\
&\quad \to X_1\ (x \triangleleft y) \to \overrightarrow{X_1}\ (y \blacktriangleleft z) \to X_1\ (x \triangleleft \mu_{X_0^* M}\ y\ z)
\end{aligned}$$

Note that α_X satisfies the algebraic laws and that μ_X^α is associative and unital with units given by η_X^α. These laws are not witnessed by an identity this time

but by a higher unary cell and are obtained by following the proofs exposed earlier in this chapter using target universality instead of the 0-algebra property.

We now define *univalent* coherent M-algebras. In a coherent M-algebra, all cells are target universal and all unary cells are therefore equivalences. Supposing that we have a M-opetopic type X equipped with a structure of coherent M-algebra, we define

$$\mathsf{id\text{-}to\text{-}equiv}_X : \{i : \mathsf{Idx}_M\}\ \{x\ y : X_0\ i\} \to x = y \to X_1\ ((i, y) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ x))$$

with defining equation

$$\mathsf{id\text{-}to\text{-}equiv}_X\ \mathsf{refl}_x :\equiv \eta^{\alpha}_X\ (i, x)$$

This leads us to the definition of univalent coherent M-algebras.

Definition 3.3.4 (Univalent coherent M-algebra)**.** Let X be a M-opetopic type with a structure of coherent M-algebra. We say that X is univalent if the function $\mathsf{id\text{-}to\text{-}equiv}_X$ is an equivalence and if the opetopic type $X_{>0}$ is univalent.

We will see that being a univalent coherent M-algebra is a property.

We have now introduced all the required notions in order to state the main theorem of this section.

Theorem 3.3.5. *Let X be a M-opetopic type. The property witnessing that X is fibrant is equivalent to the structure of univalent coherent M-algebra on X.*

The plan of the proof is to first establish the corresponding logical equivalence and conclude with the observation that being a univalent coherent M-algebra is a property. We start with a number of lemmas about algebras which we will later use to make the connection between fibrancy and target and source universality.

Lemma 3.3.6. *Let a monad M and let families $X_0 : \mathsf{Fam}_M$, $X_1 : \mathsf{Fam}_{M/X_0}$, and $X_2 : \mathsf{Fam}_{M/X_0/X_1}$ such that (X_0, X_1) and (X_1, X_2) are 0-algebras. Let a constructor $c : \mathsf{Cns}_M\ i$, a position $p : \mathsf{Pos}_M\ c$, two cells $y_f, y_g : X_0\ i$, a cell $f : X_1\ ((i, y_f) \triangleleft (c, x))$, and a cell $g : X_1\ ((i, y_g) \triangleleft (c, x))$. Then we have the following equivalence:*

$$(\textstyle\sum_{(h : X_1\ ((i, y_g) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ y_f)))} X_2\ ((\ldots, g) \triangleleft \theta_{h,f})) \simeq ((y_f, f) = (y_g, g))$$

Proof. We equivalently define the equivalence

$$\begin{aligned}&\textstyle\sum_{(h : X_1\ ((i, y_g) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ y_f)))} X_2\ ((\ldots, g) \triangleleft \theta_{h,f})\\ &\simeq \textstyle\sum_{(q : y_f = y_g)} \mathsf{transport}^{\lambda y \to X_1\ ((i,y) \triangleleft (c,x))}\ q\ f = g\end{aligned}$$

We first apply Lemma 3.2.3 with the cell

$$\eta^{\alpha}_{X_2}\ (i, y_f) : X_1\ ((i, y_f) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ y_f))$$

in order to establish the family of equivalences

$$e_{y_g} : X_1\ ((i, y_g) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ y_f)) \simeq (y_f = y_g)$$

such that $e_{y_f}\ (\eta^{\alpha}_{X_2}\ (i, y_f)) = \mathsf{refl}$. We finally define, for any identity $q : y_f = y_g$, the equivalence

$$X_2\ ((\dots, g) \triangleleft \theta_{e^{-1}\ q, f}) \simeq (\mathsf{transport}^{\lambda y \to X_1\ ((i,y) \triangleleft (c,x))}\ q\ f = g)$$

We assume that $q \equiv \mathsf{refl}$ therefore $e^{-1}\ q = \eta^{\alpha}_{X_2}\ (i, y_f)$ and we are left to establish the equivalence

$$X_2\ ((\dots, g) \triangleleft \theta_{\eta^{\alpha}_{X_2}\ (i,y_f), f}) \simeq (f = g)$$

Applying Lemma 3.2.3, we obtain the equivalence

$$X_2\ ((\dots, g) \triangleleft \theta_{\eta^{\alpha}_{X_2}\ (i,y_f), f}) \simeq (\mu^{\alpha}_{X_2}\ (\eta^{\alpha}_{X_2}\ (i, y_f))\ f = g)$$

We conclude by left unitality of $\mu^{\alpha}_{X_2}$. □

We deduce the following corollary.

Corollary 3.3.7. *Let a monad M and let families $X_0 : \mathsf{Fam}_M$, $X_1 : \mathsf{Fam}_{M/X_0}$, and $X_2 : \mathsf{Fam}_{M/X_0/X_1}$ such that (X_0, X_1) and (X_1, X_2) are 0-algebras. Let two cells $x_f, x_g : X_0\ i$, a constructor $c : \mathsf{Cns}_M\ i$, and a family of cells $y : \overrightarrow{X_0}\ c$. Suppose given two cells $f : X_1\ ((i, x_f) \triangleleft (c, y))$ and $g : X_1\ ((i, x_g) \triangleleft (c, y))$, the following type is contractible:*

$$\textstyle\sum_{(h : X_1\ ((i,y_g) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ y_f)))} X_2\ ((\dots, g) \triangleleft \theta_{h,f})$$

Proof. This last type is equivalent to the identity type $(x_f, f) = (x_g, g)$ by Lemma 3.3.6 which lives in the type $\sum_{(x : X_0\ i)} X_1\ ((i, x) \triangleleft (c, y))$ which we know to be contractible since (X_0, X_1) is a 0-algebra. □

We can derive similar results for identities between two cells which differ at a source position.

Lemma 3.3.8. *Let a monad M and let families $X_0 : \mathsf{Fam}_M$, $X_1 : \mathsf{Fam}_{M/X_0}$, and $X_2 : \mathsf{Fam}_{M/X_0/X_1}$ such that (X_0, X_1) and (X_1, X_2) are 0-algebras. Let a cell $x : X_0\ i$, a constructor $c : \mathsf{Cns}_M\ i$, a position $p : \mathsf{Pos}_M\ c$, a family $y : \overrightarrow{X_0}\ c$ defined everywhere but at position p, two cells $y_f, y_g : X_0\ (\mathsf{Typ}_M\ c\ p)$, a cell $f : X_1\ ((i, x) \triangleleft (c, y[y_f/p]))$, and a cell $g : X_1\ ((i, x) \triangleleft (c, y[y_g/p]))$. Then we have the following equivalence:*

$$(\textstyle\sum_{(h : X_1\ ((\dots,y_f) \triangleleft (\eta_M\ \dots, \eta\text{-}\mathsf{dec}\ y_g)))} X_2\ ((\dots, g) \triangleleft \theta_{f,h})) \simeq ((y_g, g) = (y_f, f))$$

Proof. We equivalently define the equivalence

$$\begin{aligned}&\textstyle\sum_{(h : X_1\ ((\dots,y_f) \triangleleft (\eta_M\ \dots, \eta\text{-}\mathsf{dec}\ y_g)))} X_2\ ((\dots, g) \triangleleft \theta_{f,h})\\ &\simeq \textstyle\sum_{(q : y_g = y_f)} \mathsf{transport}^{\lambda y_p \to X_1\ ((i,x) \triangleleft (c, y[y_p/p]))}\ q\ g = f\end{aligned}$$

We first use Lemma 3.2.3 with the cell

$$\eta^{\alpha}_{X_2}\ (i, y_g) : X_1\ ((\dots, y_g) \triangleleft (\eta_M\ \dots, \eta\text{-}\mathsf{dec}\ y_g))$$

in order to establish the family of equivalences

$$e_{y_f} : X_1\ ((\dots, y_f) \triangleleft (\eta_M\ \dots, \eta\text{-dec}\ y_g)) \simeq (y_g = y_f)$$

such that $e_{y_g}\ (\eta^{\alpha}_{X_2}\ (\dots, y_g)) = \mathsf{refl}$. We finally define, for any identity $q : y_g = y_f$, the equivalence

$$X_2\ ((\dots, g) \triangleleft \theta_{f, e^{-1}\ q}) \simeq (\mathsf{transport}^{\lambda y_p \to X_1\ ((i,x) \triangleleft (c, y[y_p/p]))}\ q\ g = f)$$

We assume that $q \equiv \mathsf{refl}$ therefore $e^{-1}\ q = \eta^{\alpha}_{X_2}\ (\dots, y_g)$ and we are left to establish the equivalence

$$X_2\ ((\dots, g) \triangleleft \theta_{f, \eta^{\alpha}_{X_2}\ (\dots, y_g)}) \simeq (g = f)$$

Applying Lemma 3.2.3, we obtain the equivalence

$$X_2\ ((\dots, g) \triangleleft \theta_{f, \eta^{\alpha}_{X_2}\ (\dots, y_g)}) \simeq (\mu^{\alpha}_{X_2}\ f\ (\eta\text{-dec}\ (\eta^{\alpha}_{X_2}\ (i, y_f))) = g)$$

We conclude by right unitality of $\mu^{\alpha}_{X_2}$. □

This lemma has a corresponding corollary which differs from Corollary 3.3.7 in that we need a further assumption.

Corollary 3.3.9. *Let a monad M and let families $X_0 : \mathsf{Fam}_M$, $X_1 : \mathsf{Fam}_{M/X_0}$, and $X_2 : \mathsf{Fam}_{M/X_0/X_1}$ such that (X_0, X_1) and (X_1, X_2) are 0-algebras. Let a cell $x : X_0\ i$, a constructor $c : \mathsf{Cns}_M\ i$, and a family of cells $y : \overrightarrow{X_0}\ c$ defined everywhere but at a position $p : \mathsf{Pos}_M\ c$. We suppose the following type to be a proposition:*

$$\textstyle\sum_{(y_p : X_0\ (\mathsf{Typ}_M\ c\ p))} X_1\ ((i, x) \triangleleft (c, y[y_p/p]))$$

Suppose given two cells $y_f, y_g : X_0\ (\mathsf{Typ}_M\ c\ p)$ and another two cells $f : X_1\ ((i, x) \triangleleft (c, y[y_f/p]))$ and $g : X_1\ ((i, x) \triangleleft (c, y[y_g/p]))$, the following type is contractible:

$$\textstyle\sum_{(h : X_1\ ((\dots, y_f) \triangleleft (\eta_M\ \dots, \eta\text{-dec}\ y_g)))} X_2\ ((\dots, g) \triangleleft \theta_{f,h})$$

Proof. This last type is equivalent to the identity type $(y_f, f) = (y_g, g)$ by Lemma 3.3.8 which lives in the type $\sum_{(y_p : X_0\ (\mathsf{Typ}_M\ c\ p))} X_1\ ((i, x) \triangleleft (c, y[y_p/p]))$ which we know to be a proposition by assumption. □

We now establish the counterpart of Lemma 3.2.3 in the case of a univalent coherent M-algebra.

Lemma 3.3.10. *Let X be a M-opetopic type with a structure of univalent coherent M-algebra. For any constructor $c : \mathsf{Cns}_M\ i$, any family $y : \overrightarrow{X_0}\ c$, and any element $x : X_0\ i$, a cell $f : X_1\ ((i, z) \triangleleft (c, y))$ implies the existence of a family of equivalences*

$$e_x : X_1\ ((i, x) \triangleleft (c, y)) \simeq (z = x)$$

such that $e_z\ f = \mathsf{refl}$. In particular, the structure of coherent M-algebra guarantees the existence of the cell $\alpha^{\mathsf{fill}}_X\ c\ y : X_1\ ((i, \alpha_X\ c\ y) \triangleleft (c, y))$.

Proof. Note that the proof is stated by taking f to be $\alpha_X^{\mathsf{fill}}\ c\ y$ but the truth of this lemma does not depend on this choice.

Proving this lemma amounts to establishing the family of equivalences

$$e_x : X_1\ ((i,x) \triangleleft (c,y)) \simeq X_1\ ((i,x) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ (\alpha_X\ c\ y)))$$

such that $e_{\alpha_X\ c\ y}\ (\alpha_X^{\mathsf{fill}}\ c\ y) = \eta_X^{\alpha}\ (i, \alpha_X\ c\ y)$ and concluding by univalence.

In the forward direction, given a cell $f : X_1\ ((i,x) \triangleleft (c,y))$, there exists another cell sharing the same source $\alpha_X^{\mathsf{fill}}\ c\ y : X_1\ ((i, \alpha_X\ c\ y) \triangleleft (c,y))$. This last cell being target universal, we obtain the wanted unary cell $g : X_1\ ((i,x) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ (\alpha_X\ c\ y)))$.

We check that $e_{\alpha_X\ c\ y}\ (\alpha_X^{\mathsf{fill}}\ c\ y) = \eta_X^{\alpha}\ (i, \alpha_X\ c\ y)$. By target universality of $\alpha_X^{\mathsf{fill}}\ c\ y$, we have a unary cell $g : X_1\ ((i, \alpha_X\ c\ y) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ (\alpha_X\ c\ y)))$ along with its filler $g^{\mathsf{fill}} : X_2\ ((\ldots, \alpha_X^{\mathsf{fill}}\ c\ y) \triangleleft \theta_{g, \alpha_X^{\mathsf{fill}}\ c\ y})$. By left-unitality of μ_X^{α}, we have a concurrent unary cell $\eta_X^{\alpha}\ (i, \alpha_X\ c\ y) : X_1\ ((i, \alpha_X\ c\ y) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ (\alpha_X\ c\ y)))$ along with its filler $\mu_{X_{>0}}^{\alpha^{\mathsf{fill}}}\ (\eta_X^{\alpha}\ (i, \alpha_X\ c\ y))\ (\alpha_X^{\mathsf{fill}}\ c\ y) : X_2\ ((\ldots, \alpha_X^{\mathsf{fill}}\ c\ y) \triangleleft \theta_{\eta_X^{\alpha}\ (i, \alpha_X\ c\ y), \alpha_X^{\mathsf{fill}}\ c\ y})$. By source universality of g^{fill}, we obtain the required unary cell $X_2\ ((\ldots, g) \triangleleft (\eta_{M/X_0}\ \ldots, \eta\text{-}\mathsf{dec}\ (\eta_X^{\alpha}\ (i, \alpha_X\ c\ y))))$.

In the other direction, from a unary cell $f : X_1\ ((i,x) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ (\alpha_X\ c\ y)))$, we obtain the wanted cell as $\mu_{X_{>0}}^{\alpha}\ f\ (\alpha_X^{\mathsf{fill}}\ c\ y) : X_1\ ((i,x) \triangleleft (c,y))$.

It remains to show that the two sides of the equivalence are inverse to each other. Starting with a cell $f : X_1\ ((i,x) \triangleleft (c,y))$, we want to establish the identity $\mu_{X_{>0}}^{\alpha}\ g\ (\alpha_X^{\mathsf{fill}}\ c\ y) = f$ where g is the lift of f that we obtain by target universality of $\alpha_X^{\mathsf{fill}}\ c\ y$. This amounts to finding a unary cell $X_1\ ((\ldots, f) \triangleleft (\eta_{M/X_0} \ldots, \eta\text{-}\mathsf{dec}\ (\mu_{X_{>0}}^{\alpha}\ g\ (\alpha_X^{\mathsf{fill}}\ c\ y))))$. We obtain it by target universality of $\mu_{X_{>0}}^{\alpha^{\mathsf{fill}}}\ g\ (\alpha_X^{\mathsf{fill}}\ c\ y) : X_2\ ((\ldots, \mu_{X_{>0}}^{\alpha}\ g\ (\alpha_X^{\mathsf{fill}}\ c\ y)) \triangleleft \theta_{g, \alpha_X^{\mathsf{fill}}\ c\ y})$ along with the second cell sharing the same source: $g^{\mathsf{fill}} : X_2\ ((\ldots, f) \triangleleft \theta_{g, \alpha_X^{\mathsf{fill}}\ c\ y})$.

Finally, starting with a cell $f : X_1\ ((i,x) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ (\alpha_X\ c\ y)))$, we want to establish the identity $g = f$ where g is the lift of $\mu_{X_{>0}}^{\alpha}\ f\ (\alpha_X^{\mathsf{fill}}\ c\ y) : X_1\ ((i,x) \triangleleft (c,y))$ obtained by target universality of $\alpha_X^{\mathsf{fill}}\ c\ y$. The corresponding filler is $g^{\mathsf{fill}} : X_2\ ((\ldots, \mu_{X_{>0}}^{\alpha}\ f\ (\alpha_X^{\mathsf{fill}}\ c\ y)) \triangleleft \theta_{g, \alpha_X^{\mathsf{fill}}\ c\ y})$. Consider the other cell $\mu_X^{\alpha^{\mathsf{fill}}}\ f\ (\alpha_X^{\mathsf{fill}}\ c\ y) : X_2\ ((\ldots, \mu_{X_{>0}}^{\alpha}\ f\ (\alpha_X^{\mathsf{fill}}\ c\ y)) \triangleleft \theta_{f, \alpha_X^{\mathsf{fill}}\ c\ y})$. By source universality of g^{fill}, we use $\mu_X^{\alpha^{\mathsf{fill}}}\ g\ (\alpha_X^{\mathsf{fill}}\ c\ y)$ to obtain the required unary cell and conclude by univalence. □

This allows us to deduce that such an opetopic type is fibrant.

Lemma 3.3.11. *A M-opetopic type X with a structure of univalent coherent M-algebra is fibrant.*

Proof. We show that (X_0, X_1) is a 0-algebra. That is, given a constructor $c : \mathsf{Cns}_M\ i$, and a family $x : \overrightarrow{X_0}\ c$, the following type is contractible:

$$\textstyle\sum_{(y : X_0\ i)} X_1\ ((i,y) \triangleleft (c,x))$$

The centre of contraction is readily provided by the structure of coherent M-algebra. It remains to show that for any pair of cells $f : X_1\ ((i, y_f) \triangleleft (c,x))$

and $g : X_1\ ((i, y_g) \triangleleft (c, x))$, there is an identity $(y_f, f) = (y_g, g)$. Consider Lemma 3.3.6 and notice that its proof only uses Lemma 3.2.3 that we reestablished in Lemma 3.3.10 in the case of univalent coherent M-algebras. This signifies that we have the equivalence

$$(\textstyle\sum_{(h:X_1\ ((i,y_g)\triangleleft(\eta_M\ i, \eta\text{-dec}\ y_f)))} X_2\ ((\dots, g) \triangleleft \theta_{h,f})) \simeq ((y_f, f) = (y_g, g))$$

We are then reduced to finding a pair of type

$$\textstyle\sum_{(h:X_1\ ((i,y_g)\triangleleft(\eta_M\ i, \eta\text{-dec}\ y_f)))} X_2\ ((\dots, g) \triangleleft \theta_{h,f})$$

which is obtained by target universality of f.

Finally, the coinductive hypothesis applied to $X_{>0}$, which has a structure of univalent coherent M-algebra, allows us to conclude that X is fibrant. □

It remains to establish the other side of the equivalence. We start by proving the following two lemmas.

Lemma 3.3.12. *Let X be a fibrant M-opetopic type. Any cell of type $X_1\ ((i, y) \triangleleft (c, x))$ is* uniquely *target universal.*

Lemma 3.3.13. *Let X be a fibrant M-opetopic type. Let a cell $x : X_0\ i$, a constructor $c : \mathsf{Cns}_M\ i$, a position $p : \mathsf{Pos}_M\ c$, and a family of cells $y : \overrightarrow{X_0}\ c$. We suppose the following type to be a proposition:*

$$\textstyle\sum_{(y_p:X_0\ (\mathsf{Typ}_M\ c\ p))} X_1\ ((i, x) \triangleleft (c, y[y_p/p]))$$

Then any cell $f : X_1\ ((i, x) \triangleleft (c, y))$ is uniquely source universal at position p.

It is important to note that, when an opetopic type X is fibrant, being target universal is no longer an additional structure but a property as we shall see. The same remark applies in the case of source universality under the appropriate condition.

Proof of Lemma 3.3.12. Let X be a fibrant M-opetopic type. The coinductive hypothesis allows us to establish that property for $X_{>0}$ which is fibrant.

It remains to show that all 1-cells are uniquely target universal. Consider a cell $f : X_1\ ((i, y_f) \triangleleft (c, x))$ that we want to show to be target universal and another cell sharing the same source $g : X_1\ ((i, y_g) \triangleleft (c, x))$. The 0-algebra provides us with an identity $(y_f, f) = (y_g, g)$ that gives us a pair $(h, h^{\mathsf{fill}}) : \textstyle\sum_{(h:X_1\ ((i,y_g)\triangleleft(\eta_M\ i, \eta\text{-dec}\ y_f)))} X_2\ ((\dots, g) \triangleleft \theta_{h,f})$ by Lemma 3.3.6. Moreover, this pair lives in a contractible type by Corollary 3.3.7. To conclude that f is target universal, we have to prove that h^{fill} is target universal and source universal at position $\mathsf{inl}\ \star$. The coinductive hypothesis allows to readily conclude that h^{fill} is uniquely target universal.

In order to conclude that it is also uniquely source universal we first note that the type

$$\textstyle\sum_{(h:X_1\ ((i,y_g)\triangleleft(\eta_M\ i, \eta\text{-dec}\ y_f)))} X_2\ ((\dots, g) \triangleleft \theta_{h,f})$$

is contractible and is, in particular, a proposition, as it is equivalent to a path $(y_f, f) = (y_g, g)$ by Lemma 3.3.6 which lives in a contractible type. This allows us to conclude that h^{fill} is uniquely source universal at position $\mathsf{inl}\ \star$ by applying Lemma 3.3.13. □

Proof of Lemma 3.3.13. For any cell $y_g : X_0\ (\mathsf{Typ}_M\ c\ p)$ and cell $g : X_1\ ((i, x) \triangleleft (c, y[y_g/p]))$, we apply Corollary 3.3.9 to obtain the pair

$$(h, h^{\mathsf{fill}}) : \textstyle\sum_{(h:X_1\ ((\ldots,y)\triangleleft(\eta_M\ \ldots,\eta\text{-}\mathsf{dec}\ y_g)))} X_2\ ((\ldots, g) \triangleleft \theta_{f,h})$$

which therefore lives in a contractible type. We apply Lemma 3.3.12 to deduce that h^{fill} is uniquely target universal, and we use the coinductive hypothesis along with the fact that the type of the pair (h, h^{fill}) is contractible, and is therefore a proposition, to conclude that h^{fill} is uniquely source universal at position $\mathsf{inr}\ (p, \mathsf{inl}\ \star)$, the position of h. □

Corollary 3.3.14. *For a M-opetopic type, being a univalent coherent M-algebra is a property.*

We want to emphasise the surprising nature of this result. Being a coherent M-algebra is an additional structure which turns out to be a property when the structure is univalent.

Proof. Let X be a M-opetopic type and suppose it is a univalent coherent M-algebra. By Lemma 3.3.11, X is therefore fibrant and, for any pasting diagram, its composition and filler live in a contractible type. Finally, Lemma 3.3.12 informs us that the data witnessing that any cell is target universal lives in a contractible type too. We conclude that being a univalent coherent M-algebra is therefore a property. □

At last, we define the remaining side of the equivalence.

Lemma 3.3.15. *A fibrant M-opetopic type has a unique structure of univalent coherent M-algebra.*

Proof. Let X be a fibrant M-opetopic type. The coinductive hypothesis allows us to establish that $X_{>0}$, which is fibrant, has a unique structure of univalent coherent M-algebra.

Now, the composition and filler for pasting diagrams of 0-cells are readily given by the 0-algebra (X_0, X_1) and live in a contractible type. The fact that any 1-cell is uniquely target universal follows from Lemma 3.3.12. □

We summarise our results in the proof of Theorem 3.3.5.

Proof of Theorem 3.3.5. Corollary 3.3.14 states that being a univalent coherent M-algebra is a property. Lemma 3.3.11 and Lemma 3.3.15 establish a logical equivalence between being fibrant and being a univalent coherent M-algebra. We conclude that this logical equivalence is an equivalence of types. □

3.4 An explicit characterisation of the composition

Given a monad M and a M-0-algebra (X_0, X_1), Lemma 3.2.3 informs us that X_1 is equivalent to a family of identity types. A family $X_2 : \mathsf{Fam}_{M/X_0/X_1}$ such that (X_1, X_2) is a 0-algebra then induces an operation of composition on these 1-cells. In this section we provide a concrete definition of this composition acting on these identity types in terms of the usual operations on identity types and prove that it is equivalent to the one induced by X_2. This perspective allows to leverage the body of results that has already been established about identity types to deduce properties about the multiplication induced by X_2. This result was initially used by the author in order to gain intuition about the properties of the multiplication induced by X_2.

We start with the definition of the families $X_1^= : \mathsf{Fam}_{M/X_0}$ and $X_2^= : \mathsf{Fam}_{M/X_0/X_1^=}$ where $X_1^=$ equivalent to X_1 under Lemma 3.2.3.

Definition 3.4.1. Let M be a monad and let the families $X_0 : \mathsf{Fam}_M$, $X_1 : \mathsf{Fam}_{M/X_0}$, and $X_2 : \mathsf{Fam}_{M/X_0/X_1}$ such that (X_0, X_1) and (X_1, X_2) are 0-algebras. For any constructor $c : \mathsf{Cns}_M\ i$, any family of elements $x : \overrightarrow{X_0}\ c$, and any element $y : X_0\ i$, we define the family

$$X_1^=\ ((i, y) \triangleleft (c, x)) :\equiv \alpha_{X_1}\ c\ x = y$$

We define an operation of composition of pasting diagrams of cells valued in $X_1^=$; that is, given a constructor $c : \mathsf{Cns}_{M/X_0}\ i$ and a family $x : \overrightarrow{X_1^=}\ c$, we define an element of $X_1^=\ i$. We name this operation $\alpha_{X_2^=}$ which defines a family $X_2^= : \mathsf{Fam}_{M/X_0/X_1^=}$ defined as

$$X_2^=\ ((i, y) \triangleleft (c, x)) :\equiv \alpha_{X_2^=}\ c\ x = y$$

We define $\alpha_{X_2^=}$ by induction on c.

- If c is of the form $\mathsf{lf}\ (i, x)$, we need to define an element of type $X_1^=\ ((i, x) \triangleleft (\eta_M\ i, \eta\text{-}\mathsf{dec}\ x))$, that is an identity $\alpha_{X_1}\ (\eta_M\ i)\ (\eta\text{-}\mathsf{dec}\ x) = x$ which we define to be α_η using the fact that X_1 is a 1-algebra.
- If c is of the form $\mathsf{nd}\ ((i, y) \triangleleft (c, z))\ \{w\}\ t$, we need to define an identity of type

  $$\alpha_{X_1}\ (\mu_M\ c\ (\lambda p \to \mathsf{pr_1}\ (w\ p)))\ (\lambda p \to \mathsf{pr_2}\ (w\ (\mathsf{pr}_1^\mu\ p))\ (\mathsf{pr}_2^\mu\ p)) = y$$

  We use α_μ to establish the first identity:

  $$\begin{aligned}&\alpha_{X_1}\ (\mu_M\ c\ (\lambda p \to \mathsf{pr_1}\ (w\ p)))\ (\lambda p \to \mathsf{pr_2}\ (w\ (\mathsf{pr}_1^\mu\ p))\ (\mathsf{pr}_2^\mu\ p))\\ &= \alpha_{X_1}\ c\ (\lambda p \to \alpha_{X_1}\ (\mathsf{pr_1}\ (w\ p))\ (\lambda q \to \mathsf{pr_2}\ (w\ p)\ q))\end{aligned}$$

  Then, we get a second identity by congruence and from the induction hypothesis applied to the pasting diagram $(t\ p, \lambda q \to x\ (\mathsf{inr}\ (p, q)))$ for any position $p : \mathsf{Pos}_M\ c$:

  $$\alpha_{X_1}\ c\ (\lambda p \to \alpha_{X_1}\ (\mathsf{pr_1}\ (w\ p))\ (\lambda q \to \mathsf{pr_2}\ (w\ p)\ q)) = \alpha_{X_1}\ c\ z$$

Finally, the last identity $\alpha_{X_1}\ c\ z = y$ is given by $x\ (\mathsf{inl}\ \star)$.

In the end, the formal definition of the identity thus defined is:

$$\alpha\text{-}\mu \cdot \mathsf{ap}\ (\alpha_{X_1}\ c)\ (\mathsf{funext}\ (\lambda p \to \alpha_{X_2^=}\ (t\ p)\ (\lambda q \to x\ (\mathsf{inr}\ (p, q)))))\cdot x\ (\mathsf{inl}\ \star)$$

By contractibility of singletons, $(X_1^=, X_2^=)$ is a 0-algebra and we now want to check that $X_2^=$ is equivalent to X_2 in a suitable sense in order to transport the properties of $\alpha_{X_2^=}$ to α_{X_2}. To this effect, we first have to show that the equivalence of Lemma 3.2.3 is a morphism of 0-algebras — in that the laws are not mapped — from (X_1, X_2) to $(X_1^=, X_2^=)$.

Lemma 3.4.2. *Let M be a monad and let the families $X_0 : \mathsf{Fam}_M$, $X_1 : \mathsf{Fam}_{M/X_0}$, and $X_2 : \mathsf{Fam}_{M/X_0/X_1}$ such that (X_0, X_1) and (X_1, X_2) are 1-algebras. Consider the 0-algebra $(X_1^=, X_2^=)$ defined in Definition 3.4.1. The equivalence e of Lemma 3.2.3 — where we take the cell f to be the filler provided by the algebra (X_0, X_1) — is a morphism of 0-algebras from (X_1, X_2) to $(X_1^=, X_2^=)$, that is*

$$e\ (\alpha_{X_2}\ c\ x) = \alpha_{X_2^=}\ c\ (\lambda p \to e\ (x\ p))$$

for any constructor $c : \mathsf{Cns}_{M/X_0}\ i$ and family $x : \overrightarrow{X_1}\ c$.

Proof. We proceed by induction on c.

- If c is of the form $\mathsf{lf}\ (i, y)$, we have to establish the following identity:

  $$e\ (\alpha_{X_2}\ (\mathsf{lf}\ (i, y))\ x) = e\ (\mathsf{transport}^{\lambda y \to X_1\ ((i,y)\ (\mathsf{pos}^{\eta})\ i))\triangleleft(\eta_M\ i, y))}\ p\ (\eta^{\alpha}_{X_2}\ (i, y)))$$

  with $p :\equiv \mathsf{funext}\ (\mathsf{Pos}\text{-}\eta\text{-}\mathsf{elim}_M\ i\ (\lambda p \to y = y)\ \mathsf{refl})$ and where the right-hand side is the formal definition of $\alpha\text{-}\eta$ established in Theorem 3.2.4. Remember that $\eta^{\alpha}_{X_2}\ (i, y) \equiv \alpha_{X_2}\ (\mathsf{lf}\ (i, y))\ \bot\text{-}\mathsf{elim}$. It is clear that $p = \mathsf{refl}$ and we conclude the proof by noticing that $x = \bot\text{-}\mathsf{elim}$ is obviously true as they are both functions with domain $\bot$.

- If c is of the form $\mathsf{nd}\ ((i, y) \triangleleft (c, z))\ \{w\}\ t$, we have to establish the following identity:

  $$e\ (\alpha_{X_2}\ (\mathsf{nd}\ ((i, y) \triangleleft (c, z))\ \{w\}\ t)\ x) = \alpha\text{-}\mu \cdot \mathsf{ap}\ (\alpha_{X_1}\ c)\ p \cdot e\ (x\ (\mathsf{inl}\ \star))$$

  with $p :\equiv \mathsf{funext}\ (\lambda p \to \alpha_{X_2^=}\ (t\ p)\ (\lambda q \to e\ (x\ (\mathsf{inr}\ (p, q)))))$.

  We first establish the identity

  $$\begin{aligned} &e\ (\alpha_{X_2}\ (\mathsf{nd}\ ((i, y) \triangleleft (c, z))\ t)\ x) \\ &= e\ (\mu^{\alpha}_{X_2}\ (x\ (\mathsf{inl}\ \star))\ (\lambda p \to \alpha_{X_2}\ (t\ p)\ (\lambda q \to x\ (\mathsf{inr}\ (p, q))))) \end{aligned}$$

  using Corollary 3.2.7 in context.

  Then we establish the identity

  $$\begin{aligned} &e\ (\mu^{\alpha}_{X_2}\ (x\ (\mathsf{inl}\ \star))\ (\lambda p \to \alpha_{X_2}\ (t\ p)\ (\lambda q \to x\ (\mathsf{inr}\ (p, q))))) \\ &= \alpha\text{-}\mu \cdot \mathsf{ap}\ (\alpha_{X_1}\ c)\ q \cdot e\ (x\ (\mathsf{inl}\ \star)) \end{aligned}$$

where $q :\equiv \mathsf{funext}\ (\lambda p \to e\ (\alpha_{X_2}\ (t\ p)\ (\lambda q \to x\ (\mathsf{inr}\ (p, q)))))$.

Instead, we prove the more general statement:

$$e\ (\mu^{\alpha}_{X_2}\ f\ g) = \alpha\text{-}\mu \cdot \mathsf{ap}\ (\alpha_{X_1}\ c)\ (\mathsf{funext}\ (\lambda p \to e\ (g\ p))) \cdot e\ f$$

for any constructors $c : \mathsf{Cns}_M\ i$ and $d : \overrightarrow{\mathsf{Cns}_M}\ c$, any 0-cells $x : X_0\ i$, $y : \overrightarrow{X_0}\ c$, and $z : \overrightarrow{X_0}\ (\mu_M\ c\ d)$, and any 1-cells $f : X_1\ ((i, x) \triangleleft (c, y))$ and $g : \overrightarrow{X_1}\ ((c, y) \blacktriangleleft (\lambda p \to (d\ p, \lambda q \to z\ (\mathsf{pair}^{\mu}\ p\ q))))$.

Knowing that (X_0, X_1) is a 0-algebra, we can eliminate f and g and assume the following equalities

$$g\ p \equiv \alpha^{\mathsf{fill}}_{X_1}\ (d\ p)\ (\lambda q \to z\ (\mathsf{pair}^{\mu}\ p\ q))$$
$$f\ \equiv \alpha^{\mathsf{fill}}_{X_1}\ c\ (\lambda p \to \alpha_{X_1}\ (d\ p)\ (\lambda q \to z\ (\mathsf{pair}^{\mu}\ p\ q)))$$

Then we readily obtain $e\ (\mu^{\alpha}_{X_2}\ f\ g) \equiv \alpha\text{-}\mu$ according to the definition of $\alpha\text{-}\mu$ that we gave in Theorem 3.2.4 and it remains to show that both $e\ f$ and $\mathsf{ap}\ (\alpha_{X_1}\ c)\ (\mathsf{funext}\ (\lambda p \to e\ (g\ p)))$ are equal to refl. But both $e\ f$ and $e\ (g\ p)$ are equal to refl by the very definition of e which concludes the proof.

Returning to the original proof, it is easy to establish the remaining identity by applying the induction hypothesis in context:

$$\mathsf{funext}\ (\lambda p \to e\ (\alpha_{X_2}\ (t\ p)\ (\lambda q \to x\ (\mathsf{inr}\ (p, q)))))$$
$$= \mathsf{funext}\ (\lambda p \to \alpha_{X_2^{=}}\ (t\ p)\ (\lambda q \to e\ (x\ (\mathsf{inr}\ (p, q)))))$$

□

This last lemma allows concluding that $X_2^{=}$ is equivalent to X_2.

Lemma 3.4.3. *Let M be a monad and let the families $X_0 : \mathsf{Fam}_M$, $X_1 : \mathsf{Fam}_{M/X_0}$, and $X_2 : \mathsf{Fam}_{M/X_0/X_1}$ such that (X_0, X_1) and (X_1, X_2) are 1-algebras. For any constructor $c : \mathsf{Cns}_{M/X_0}\ i$ and 1-cells $x : \overrightarrow{X_1}\ c$ and $y : X_1\ i$,*

$$X_2\ ((i, y) \triangleleft (c, x)) \simeq X_2^{=}\ ((i, e\ y) \triangleleft (c, \lambda p \to e\ (x\ p)))$$

with e the equivalence established in Lemma 3.4.2 where we take the cell f to be the filler provided by the algebra (X_0, X_1).

Proof. Let $c : \mathsf{Cns}_{M/X_0}\ i$ be a constructor and let $x : \overrightarrow{X_1}\ c$ and $y : X_1\ i$ be 1-cells. We start by applying Lemma 3.2.3 and obtain a first equivalence:

$$X_2\ ((i, y) \triangleleft (c, x)) \simeq (\alpha_{X_2}\ c\ x = y)$$

Then using the fact that e is an equivalence, we obtain the second equivalence:

$$(\alpha_{X_2}\ c\ x = y) \simeq (e\ (\alpha_{X_2}\ c\ x) = e\ y)$$

Finally, using the equivalence established in Lemma 3.4.2 we obtain the final equivalence:

$$(e\ (\alpha_{X_2}\ c\ x) = e\ y) \simeq (\alpha_{X_2^{=}}\ c\ (\lambda p \to e\ (x\ p)) = e\ y)$$

□

Having established that X_2 is equivalent to $X_2^=$, let us revisit Corollary 3.3.7 and Corollary 3.3.9 in light of this new connection. In the former case, establishing that the following type is contractible

$$\sum_{(h:X_1\ ((i,y_g)\triangleleft(\eta_M\ i,\eta\text{-dec }y_f)))} X_2\ ((\ldots,g)\triangleleft\theta_{h,f})$$

for any 1-cells $f : X_1\ ((i,y_f)\triangleleft(c,x))$ and $g : X_1\ ((i,y_g)\triangleleft(c,x))$ is equivalent to showing that the following type is contractible

$$\sum_{(h:X_1^=\ ((i,y_g)\triangleleft(\eta_M\ i,\eta\text{-dec }y_f)))} X_2^=\ ((\ldots,g)\triangleleft\theta_{h,f})$$

for any 1-cells $f : X_1^=\ ((i,y_f)\triangleleft(c,x))$ and $g : X_1^=\ ((i,y_g)\triangleleft(c,x))$. Using the definition of $X_2^=$, we realise that this actually amounts to showing that the following function is an equivalence:

$$\lambda h \to \alpha_{X_2^=}\ \theta_{h,f}$$

for any 1-cell $f : X_1^=\ ((i,y_f)\triangleleft(c,x))$. Or, equivalently, if we replace $\alpha_{X_2^=}$ by its definition:

$$\begin{aligned}\lambda h \to &\alpha_{\text{-}}\mu\cdot\mathsf{ap}\ (\alpha_{X_1}\ (\eta_M\ i))\ (\mathsf{funext}\ (\eta\text{-dec}\\ &(\alpha_{X_2^=}\ (\eta_{M/X_0}\ ((i,y_f)\triangleleft(c,x)))\ (\eta\text{-dec}\ f))))\cdot h\end{aligned}$$

which can, in the present case, be simplified to

$$\lambda h \to \alpha_{\text{-}}\mu\cdot\mathsf{ap}\ (\alpha_{X_1}\ (\eta_M\ i))\ f\cdot h$$

owing to the fact that $(X_1^=, X_2^=)$ is a 1-algebra as the equivalent algebra (X_1, X_2) is a 1-algebra too. But, this function is obviously an equivalence as the operation of precomposing with an identity is an equivalence.

Now, in the case of Corollary 3.3.9 things become a bit more subtle. Establishing that the following type is contractible

$$\sum_{(h:X_1\ ((\ldots,x_f)\triangleleft(\eta_M\ \ldots,\eta\text{-dec }x_g)))} X_2\ ((\ldots,g)\triangleleft\theta_{f,h})$$

for any 1-cells $f : X_1\ ((i,y)\triangleleft(c,x[x_f/p]))$ and $g : X_1\ ((i,y)\triangleleft(c,x[x_g/p]))$ amounts to showing that the following function is an equivalence under the hypotheses mentioned in the corollary:

$$\lambda h \to \alpha_{X_2^=}\ \theta_{f,h}$$

for any 1-cell $f : X_1^=\ ((i,y)\triangleleft(c,x[x_f/p]))$ and where h has type $X_1^=\ ((\ldots,x_f)\triangleleft(\eta_M\ \ldots,\eta\text{-dec }x_g))$. Or, equivalently, using the definition of $\alpha_{X_2^=}$

$$\begin{aligned}\lambda h \to &\alpha_{\text{-}}\mu\cdot\mathsf{ap}\ (\alpha_{X_1}\ c)\ (\mathsf{funext}\ (\lambda p \to\\ &\alpha_{X_2^=}\ (d\ p)\ (h'\ p)))\cdot f\end{aligned}$$

where d and h' are defined as follows:

$$d\ q :\equiv \begin{cases}\eta_{M/X_0}\ ((\mathsf{Typ}_M\ c\ p, x_f)\triangleleft(\eta_M\ (\mathsf{Typ}_M\ c\ p), \eta\text{-dec }x_g)) & \text{if } q \equiv p\\ \mathsf{lf}\ (\mathsf{Typ}_M\ c\ q, x\ q) & \text{otherwise}\end{cases}$$

$$h'\ q :\equiv \begin{cases} \eta\text{-dec}\ h & \text{if } q \equiv p \\ \bot\text{-elim} & \text{otherwise} \end{cases}$$

Using the fact that $(X_1^=, X_2^=)$ is a 1-algebra, we consider the following equivalent function:

$$\begin{aligned} \lambda h \to & \alpha\_\mu \\ & \cdot\ \mathsf{ap}\ (\alpha_{X_1}\ c)\ (\mathsf{funext}\ (\lambda p \to \alpha\_\eta)) \\ & \cdot\ \mathsf{ap}\ (\lambda x_p \to \alpha_{X_1}\ c\ x[x_p/p])\ h \\ & \cdot\ f \end{aligned}$$

This function is an equivalence if and only if the following function is itself an equivalence:

$$\lambda h \to \mathsf{ap}\ (\lambda x_p \to \alpha_{X_1}\ c\ x[x_p/p])\ h$$

which is equivalent to requiring that the function $\lambda x_p \to \alpha_{X_1}\ c\ x[x_p/p]$ is an embedding. But this corresponds exactly to the hypothesis of Corollary 3.3.9 stating that the following type is a proposition:

$$\textstyle\sum_{(x_p : X_0\ (\mathsf{Typ}_M\ c\ p))} X_1\ ((i, y) \triangleleft (c, x[x_p/p]))$$

3.5 M-multicategories

In this section, we define M-multicategories which are a generalisation of ordinary categories whose arity of morphisms is specified by the constructors of a monad M. We establish an equivalence between M-multicategories and fibrant $M/$-opetopic types whose family of 1-cells is a set. This equivalence specialises to categories if we take the monad M to be the identity monad Id.

Definition 3.5.1 (Wild M-multicategory)**.** Let M be a monad. A wild M-multicategory C is defined by the following data:

- A family $X : \mathsf{Fam}_{M/}$ whose elements are called morphisms. The objects are then the indices of the monad M.
- A function providing a unary morphism for every index:

$$\eta_C^\alpha : (x : \mathsf{Idx}_M) \to X\ (x \triangleleft \eta_M\ x)$$

- For every objects $x : \mathsf{Idx}_M$, $y : \mathsf{Cns}_M\ x$, and $z : \overrightarrow{\mathsf{Cns}_M}\ y$, a composition function

$$\mu_C^\alpha : X\ (x \triangleleft y) \to \overrightarrow{X}\ (y \blacktriangleleft z) \to X\ (x \triangleleft \mu_M\ y\ z)$$

- The composition μ_C is right and left unital. That is, for every morphism $f : X\ (x \triangleleft y)$, we have the following identity:

$$\mu_C^\alpha\ f\ (\lambda p \to \eta_C^\alpha\ (\mathsf{Typ}_M\ y\ p)) = f$$

Likewise, for every families $y : \overrightarrow{\mathsf{Cns}_M}\ (\eta_M\ x)$ and $f : \overrightarrow{X}\ (\eta_M\ x \blacktriangleleft y)$, we have the following identity:

$$\mu^{\alpha}_{C}\ (\eta_M\ x)\ f = f\ (\mathsf{pos}^{\eta}\ x)$$

- The composition μ^{α}_{C} is associative. For every morphisms $f : X\ (x \triangleleft y)$, $g : \overrightarrow{X}\ (y \blacktriangleleft z)$, and $h : \overrightarrow{X}\ (\mu_M\ y\ z \blacktriangleleft t)$, we have

$$\mu^{\alpha}_{C}\ f\ (\lambda p \to \mu^{\alpha}_{C}\ (g\ p)\ (\lambda q \to h\ (\mathsf{pair}^{\mu}_{M}\ p\ q))) = \mu^{\alpha}_{C}\ (\mu^{\alpha}_{C}\ f\ g)\ h$$

We borrow the terminology of Capriotti and Kraus (CAPRIOTTI and KRAUS 2017) and call a M-multicategory whose family of morphisms is not a set *wild*. These multicategories do not necessarily satisfy their laws coherently.

Definition 3.5.2 (M-multicategory)**.** Let M be a monad. A M-multicategory is a wild M-multicategory such that its family of morphisms is a set.

The type of objects of a M-multicategory is the type of indices of the monad M, that is Idx_M. It is clear from this definition that, given a family of sets $X : \mathbf{1} \to \mathcal{U}$, $(X^*\ \mathsf{Id})$-multicategories are nothing more than precategories — as defined in the HoTT book — whose type of objects is $X\ \star$.

We now state the theorem of this section.

Theorem 3.5.3. *Let M be a monad. M-multicategories are equivalent to fibrant $M/$-opetopic types X whose family X_1 is a set.*

We break its proof into several lemmas.

Lemma 3.5.4. *Let M be a monad and let $X_0 : \mathsf{Fam}_{M/}$, $X_1 : \mathsf{Fam}_{M//X_0}$, and $X_2 : \mathsf{Fam}_{M//X_0/X_1}$ be families such that X_0 is a set. If (X_0, X_1) and (X_1, X_2) are 0-algebras then we can define a M-multicategory whose family of morphisms is given by X_0.*

Proof. The data defining the wild multicategory has been established in Section 3.2. In addition, we use the fact that X_0 is a set. □

The previous lemma admits a reciprocal and we break this result down into two lemmas which will each define the required 0-algebra.

Lemma 3.5.5. *Let X be a M-multicategory and let $X_0 : \mathsf{Fam}_{M/}$ be its family of morphisms. The structure of multicategory gives rise to a family $X_1 : \mathsf{Fam}_{M//X_0}$ such that (X_0, X_1) is a 1-algebra.*

Proof. We want X_1 to witness the composition of morphisms given by μ^{α}_{X}, and we start by defining an unbiased composition function:

$$\alpha_{X_1} : \{i : \mathsf{Idx}_{M/}\}\ (c : \mathsf{Cns}_{M/}\ i)\ (x : \overrightarrow{X_0}\ c) \to X_0\ i$$

We define it by induction on c:

- If c is of the form $\mathsf{lf}\ x$ then we need a cell $X_0\ (x \triangleleft \eta_M\ x)$ which we define to be $\eta^{\alpha}_{X}\ x$.

- If c is of the form $\mathsf{nd}\ (w \triangleleft y)\ \{z\}\ t$, we need a cell of type $X_0\ (w \triangleleft \mu_M\ y\ z)$ which is given by $\mu^\alpha_X\ (x\ (\mathsf{inl}\ \star))\ (\lambda p \to \alpha_{X_1}\ (t\ p)\ (\lambda q \to x\ (\mathsf{inr}\ (p, q))))$.

We now define $X_1 : \mathsf{Fam}_{M//X_0}$,

$$X_1\ ((i, y) \triangleleft (c, x)) :\equiv \alpha_{X_1}\ c\ x = y$$

An element of X_1 then witnesses that a certain pasting diagram of 0-cells (c, x) multiplies to the target 0-cell y. It is immediate that (X_0, X_1) is a 0-algebra as the type $\sum_{(y:X_0\ i)} (\alpha_{X_1}\ c\ x = y)$ is contractible. Note how α_{X_1} that we just defined coincides exactly with the composition function induced by the structure of 0-algebra on (X_0, X_1) therefore there is no clash of notation.

We now show that the 0-algebra (X_0, X_1) is a 1-algebra. We start with the unit law which asserts that, for any index $i : \mathsf{Idx}_{M/}$ and family $x : \overrightarrow{X_0}\ (\eta_{M/}\ i)$, the identity $\alpha_{X_1}\ (\eta_{M/}\ i)\ x = x\ (\mathsf{pos}^\eta\ i)$ holds. By definition of α_{X_1} and $\mathsf{pos}^\eta\ i$, this amounts to proving the identity

$$\mu^\alpha_X\ (x\ (\mathsf{inl}\ \star))\ (\lambda p \to \eta^\alpha_X\ (\mathsf{Typ}_{M/}\ (\eta_{M/}\ i)\ p)) = x\ (\mathsf{inl}\ \star)$$

which holds by right-unitality of μ^α_X.

In order to show that α_{X_1} is compatible with $\mu_{M/}$, we first need to show that it is compatible with the grafting operation γ_M. That is, for any tree $t : \mathsf{Cns}_{M/}\ (x \triangleleft y)$, any family $u : \overrightarrow{\mathsf{Cns}_{M/}}\ (y \blacktriangleleft z)$, and any family $v : \overrightarrow{X_0}\ (\gamma_M\ t\ u)$:

$$\begin{aligned}
&\alpha_{X_1}\ (\gamma_M\ t\ u)\ v \\
&= \mu^\alpha_X\ (\alpha_{X_1}\ t\ (\lambda p \to v\ (\mathsf{inl}^\gamma\ p))) \\
&\qquad (\lambda p \to \alpha_{X_1}\ (u\ p)\ (\lambda q \to v\ (\mathsf{inr}^\gamma\ p\ q)))
\end{aligned}$$

We proceed by induction on t:

- If t is of the form $\mathsf{lf}\ x$ then, by definition of α_{X_1}, we are left to prove the following identity which holds by virtue of the fact that μ^α_X is left-unital.

$$\alpha_{X_1}\ (u\ (\mathsf{pos}^\eta\ x))\ v = \mu^\alpha_X\ (\eta^\alpha_X\ x)\ (\lambda p \to \alpha_{X_1}\ (u\ p)\ (\lambda q \to v\ (\mathsf{inr}^\gamma\ p\ q)))$$

- If t is of the form $\mathsf{nd}\ (x \triangleleft y)\ t$, the left-hand side is equal to

$$\mu^\alpha_X\ (v\ (\mathsf{inl}\ \star))\ (\lambda p \to \alpha_{X_1}\ (\gamma_M\ (t\ p)\ (\lambda q \to u\ (\mathsf{pair}^\mu\ p\ q)))\ (\lambda q \to v\ (\mathsf{inr}\ (p, q))))$$

  As for the right-hand side, it is of the form $\mu^\alpha_X\ (\mu^\alpha_X\ v_1\ v_2)\ v_3$ where v_1, v_2, and v_3 are defined as follows:

$$\begin{aligned}
v_1 &:\equiv v\ (\mathsf{inl}\ \star) \\
v_2\ p &:\equiv \alpha_{X_1}\ (t\ p)\ (\lambda q \to v\ (\mathsf{inr}\ (p, \mathsf{inl}^\gamma\ q))) \\
v_3\ p &:\equiv \alpha_{X_1}\ (u\ p)\ (\lambda q \to v\ (\mathsf{inr}\ (\mathsf{pr}^\mu_1\ p, \mathsf{inr}^\gamma\ (\mathsf{pr}^\mu_2\ p)\ q)))
\end{aligned}$$

  Using the inductive hypothesis we get a first identity:

$$\begin{aligned}
&\mu^\alpha_X\ v_1\ (\lambda p \to \alpha_{X_1}\ (\gamma_M\ (t\ p)\ (\lambda q \to u\ (\mathsf{pair}^\mu\ p\ q)))\ (\lambda q \to v\ (\mathsf{inr}\ (p, q)))) \\
&= \mu^\alpha_X\ v_1\ (\lambda p \to \mu^\alpha_X\ (v_2\ p)\ (\lambda q \to v_3\ (\mathsf{pair}^\mu\ p\ q)))
\end{aligned}$$

Therefore, we are left to show that

$$\begin{aligned}&\mu^{\alpha}_{X}\ v_1\ (\lambda p \to \mu^{\alpha}_{X}\ (v_2\ p)\ (\lambda q \to v_3\ (\mathsf{pair}^{\mu}\ p\ q)))\\ &= \mu^{\alpha}_{X}\ (\mu^{\alpha}_{X}\ v_1\ v_2)\ v_3\end{aligned}$$

But this is readily proved as we know that μ^{α}_{X} is associative.

We can now prove that α_{X_1} is compatible with the operation $\mu_{M/}$. That is, given an index $x : \mathsf{Idx}_{M/}$, a constructor $y : \mathsf{Cns}_{M/}\ x$, a family of constructors $z : \overrightarrow{\mathsf{Cns}_{M/}}\ y$, and a family of elements $v : \overrightarrow{X_0}\ (\mu_{M/}\ y\ z)$, we have the following identity:

$$\alpha_{X_1}\ (\mu_{M/}\ y\ z)\ v = \alpha_{X_1}\ y\ (\lambda p \to \alpha_{X_1}\ (z\ p)\ (\lambda q \to v\ (\mathsf{pair}^{\mu}\ p\ q)))$$

We proceed by induction on y:

- If y is of the form $\mathsf{lf}\ x$, both sides of the identity reduce to $\eta^{\alpha}_{X}\ x$ by definition of α_{X_1} and we conclude the proof by reflexivity.
- If y is of the form $\mathsf{nd}\ (x \triangleleft y)\ t$, the left-hand side of the identity is equal to

  $$\alpha_{X_1}(\gamma_M\ (z\ (\mathsf{inl}\ \star))\ (\lambda p \to \mu_{M/}\ (t\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q))))))\ v$$

  while the right-hand side is equal to

  $$\mu^{\alpha}_{X}\ v_1\ (\lambda p \to \alpha_{X_1}\ (t\ p)\ (\lambda q \to \alpha_{X_1}\ (z\ (\mathsf{inr}\ (p, q)))\ (\lambda r \to v\ (\mathsf{inr}^{\gamma}\ p\ (\mathsf{pair}^{\mu}\ q\ r)))))$$

  where $v_1 :\equiv \alpha_{X_1}\ (z\ (\mathsf{inl}\ \star))\ (\lambda p \to v\ (\mathsf{inl}^{\gamma}\ p))$. We obtain a first identity using the fact that α_{X_1} is compatible with γ_M.

  $$\begin{aligned}&\alpha_{X_1}\ (\gamma_M\ (z\ (\mathsf{inl}\ \star))\ (\lambda p \to \mu_{M/}\ (t\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q))))))\ v\\ &= \mu^{\alpha}_{X}\ v_1\ (\lambda p \to \alpha_{X_1}\ (\mu_{M/}\ (t\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q))))\ (\lambda q \to v\ (\mathsf{inr}^{\gamma}\ p\ q)))\end{aligned}$$

  We conclude the proof using the inductive hypothesis to establish the following identity for any position $p : \mathsf{Pos}_{M/}\ y$:

  $$\begin{aligned}&\alpha_{X_1}\ (\mu_{M/}\ (t\ p)\ (\lambda q \to z\ (\mathsf{inr}\ (p, q))))\ (\lambda q \to v\ (\mathsf{inr}^{\gamma}\ p\ q)))\\ &= \alpha_{X_1}\ (t\ p)\ (\lambda q \to \alpha_{X_1}\ (z\ (\mathsf{inr}\ (p, q)))\ (\lambda r \to v\ (\mathsf{inr}^{\gamma}\ p\ (\mathsf{pair}^{\mu}\ q\ r))))\end{aligned}$$

□

We are now in a position to define the second 0-algebra.

Lemma 3.5.6. *Let X be a M-multicategory and let $X_0 : \mathsf{Fam}_{M/}$ be its family of morphisms. Let $X_1 : \mathsf{Fam}_{M//X_0}$ be the family that we defined in Lemma 3.5.5, the structure of X gives rise to a family $X_2 : \mathsf{Fam}_{M//X_0/X_1}$ such that (X_1, X_2) is a 0-algebra.*

Proof. We first need to establish a composition function on 1-cells which has the following type:

$$\alpha_{X_2} : \{i : \mathsf{Idx}_{M//X_0}\}\ (c : \mathsf{Cns}_{M//X_0}\ i) \to (x : \overrightarrow{X_1}\ c) \to X_1\ i$$

Let us pause a moment to understand what such a function amounts to. The definition will essentially be the same as the one of $\alpha_{X_2^=}$ that we introduced in Definition 3.4.1.

The type $X_1\ ((i, y) \triangleleft (c, x))$ is defined as $\alpha_{X_1}\ c\ x = y$. An inhabitant of this type then witnesses that the pasting diagram (c, x) multiplies to y. Now, suppose we have a pasting diagram of 1-cells $(d, z) : [\![M//X_0]\!]\ X_1\ ((i, y) \triangleleft (c, x))$. The pasting diagram (d, z) then denotes a way of composing its source pasting diagram (c, x) to y. Having a function α_{X_2} then amounts to saying that for any such configuration, we can prove that the unbiased composition of (c, x) composes to y.

We define $\alpha_{X_2}\ c\ v$ inductively on c:

- If c is of the form $\mathsf{lf}\ (i, x)$ then we have to inhabit the type

  $$X_1\ ((i, x) \triangleleft \eta_{X_0^* M/}\ (i, x))$$

  which is defined as

  $$\alpha_{X_1}\ (\eta_{M/}\ i)\ (\eta\text{-}\mathsf{dec}_{M/}\ X_0\ x) = x$$

  and which holds since (X_0, X_1) is an algebra according to Lemma 3.5.5.

- If c is $\mathsf{nd}\ ((i, x) \triangleleft (c, y))\ \{z\}\ t$ then we have to inhabit the type

  $$X_1\ ((i, x) \triangleleft \mu_{X_0^* M/}\ (c, y)\ z)$$

  which is defined as

  $$\alpha_{X_1}\ (\mu_{M/}\ c\ (\lambda p \to \mathsf{pr_1}\ (z\ p)))\ (\lambda p \to \mathsf{pr_2}\ (z\ (\mathsf{pr}_1^\mu\ p))\ (\mathsf{pr}_2^\mu\ p)) = x$$

  The operation α_{X_1} being compatible with $\mu_{M/}$, we have the identity α-μ:

  $$\begin{aligned}&\alpha_{X_1}\ (\mu_{M/}\ c\ (\lambda p \to \mathsf{pr_1}\ (z\ p)))\ (\lambda p \to \mathsf{pr_2}\ (z\ (\mathsf{pr}_1^\mu\ p))\ (\mathsf{pr}_2^\mu\ p))\\ &\quad = \alpha_{X_1}\ c\ (\lambda p \to \alpha_{X_1}\ (\mathsf{pr_1}\ (z\ p))\ (\mathsf{pr_2}\ (z\ p)))\end{aligned}$$

  For any position $p : \mathsf{Pos}_{M/}\ c$, the inductive hypothesis $\alpha_{X_2}\ (t\ p)\ (\lambda q \to v\ (\mathsf{inr}\ (p, q)))$ of type $\alpha_{X_1}\ (\mathsf{pr_1}\ (z\ p))\ (\mathsf{pr_2}\ (z\ p)) = y\ p$ allows us to establish the second identity:

  $$\alpha_{X_1}\ c\ (\lambda p \to \alpha_{X_1}\ (\mathsf{pr_1}\ (z\ p))\ (\mathsf{pr_2}\ (z\ p))) = \alpha_{X_1}\ c\ y$$

  Finally, $v\ (\mathsf{inl}\ \star)$ concludes the proof by providing the following identity:

  $$\alpha_{X_1}\ c\ y = x$$

We finally define the type family $X_2 : \mathsf{Fam}\ (M//X_0/X_1)$ with defining equation

$$X_2\ ((i, y) \triangleleft (c, x)) :\equiv \alpha_{X_2}\ c\ x = y$$

Once again, it is immediate that (X_1, X_2) is a 0-algebra as the type

$$\textstyle\sum_{(y:X_1\ i)} (\alpha_{X_2}\ c\ x = y)$$

is contractible for any constructor $c : \mathsf{Cns}_M\ i$ and family $x : \overrightarrow{X_1}\ c$. □

At this stage, the data of wild multicategory structure allowed us to construct the two families X_1 and X_2. Now, if we require in addition that X_0 is a family of sets, we can define a fibrant $M/$-opetopic type whose first three cell families are X_0, X_1, and X_2.

Lemma 3.5.7. *Let C be a M-multicategory. Consider the $M/$-opetopic type X defined as follows:*

- *X_0 is the family of morphisms of C.*
- *X_1 and X_2 are the families defined in Lemma 3.5.5 and Lemma 3.5.6.*
- *$X_{>2}$ is the terminal opetopic type for the monad $M//X_0/X_1/X_2$.*

Then X is a fibrant opetopic type.

Proof. We already know that (X_0, X_1) and (X_1, X_2) are 0-algebras. We therefore have to show that the opetopic type $X_{>1}$ is fibrant. We show that (X_2, X_3) is a 0-algebra, that is, for any index $i : \mathsf{Idx}_{M//X_0/X_1}$, and pasting diagram $(c, y) : [\![M//X_0/X_1]\!]\ X_2\ i$, the type $\sum_{(x:X_2\ i)}\ X_3\ ((i, x) \triangleleft (c, y))$ is contractible. It suffices to show that both types of the sigma type are contractible. We already know that X_3 is a trivial family so all its fibres are contractible.

As for X_2, it is defined as a family of identity types living in fibres of X_1 which is itself defined as a family of identity types living in fibres of X_0 which are sets. We deduce that X_1 is a family of propositions and that X_2 is a family of contractible types. We are left with $X_{>2}$ but this is the terminal opetopic type $\mathbf{1}^o_{M//X_0/X_1/X_2}$ which is trivially fibrant. □

It remains to show that our constructions are inverse to each other.

Lemma 3.5.8. *Let C be a M-multicategory and let C' be the M-multicategory associated to the fibrant $M/$-opetopic type obtained from C using Lemma 3.5.4, Lemma 3.5.5, and Lemma 3.5.6. The M-multicategories C and C' are equivalent.*

Proof. Firstly, the family of morphisms $X : \mathsf{Fam}_{M/}$ remains unchanged and all we have to show is that the structure of M-multicategory of C' is equivalent to the one of C. As X is a family of sets, we readily obtain that the laws are propositions and must be equal. Therefore, we only have to show that the units η_C and $\eta_{C'}$ as well as the composition operations μ_C and $\mu_{C'}$ are equal.

For any index $x : \mathsf{Idx}_M$, we have the following definitional equalities:

$$\begin{aligned}\eta_{C'}\ x &\overset{3.5.4}{\equiv} \alpha_{X_1}\ (\mathsf{lf}\ x)\ \bot\text{-}\mathsf{elim} \\ &\overset{3.5.5}{\equiv} \eta_C\ x\end{aligned}$$

where X_1 is the family of 1-cells of the opetopic type obtained from C.

As for the composition operation, given a morphism $f : X\ (x \triangleleft y)$, and a family $g : \overrightarrow{X}\ (y \blacktriangleleft z)$, we have:

$$\begin{aligned}\mu_{C'}\ f\ g &\overset{3.5.4}{\equiv} \alpha_{X_1}\ \theta_{f,g} \\ &\overset{3.5.5}{\equiv} \mu_C\ f\ (\lambda p \to \mu_C\ (g\ p)\ (\lambda q \to \eta_C\ (z\ (\mathsf{pair}^{\mu}\ p\ q)))) \\ &= \mu_C\ f\ g\end{aligned}$$

where the last identity is obtained by right unitality of μ_C. □

Finally, we prove that defining a multicategory out of a fibrant set-truncated opetopic type then reconstructing a fibrant opetopic type from this information gives us back an equivalent opetopic type.

Lemma 3.5.9. *Let X be a fibrant M/-opetopic type whose family X_0 is a family of sets. Let X' be the M/-opetopic type obtained from the M-multicategory associated with X using Lemma 3.5.4, Lemma 3.5.5, and Lemma 3.5.6. Then X and X' are equivalent opetopic types.*

Proof. We have to show an equivalence of opetopic types between X and X' (see Definition 3.1.4); that is, we need to define a coinductive sequence of equivalences between their cell families. We start by noticing that $X_0 \equiv X'_0$ therefore we simply use the identity equivalence. Next, we have to show that for all index $i : \mathsf{Idx}\ (M//X_0)$, the equivalence $X_1\ i \simeq X'_1\ i$ holds.

But Lemma 3.5.5 defines $X'_1\ ((i, y) \triangleleft (c, x))$ to be $\alpha_{X'_1}\ c\ x = y$
and Lemma 3.2.3 applied to X_1 gives us the equivalence $X_1\ ((i, y) \triangleleft (c, x)) \simeq (\alpha_{X_1}\ c\ x = y)$. We therefore conclude by establishing the identity $\alpha_{X_1}\ c\ x = \alpha_{X'_1}\ c\ x$ for any constructor $c : \mathsf{Cns}_M\ i$ and family $x : \overrightarrow{X_0}\ c$ by induction on c.

- If c is of the form $\mathsf{lf}\ (i, y)$, we must define the identity $\alpha_{X_1}\ (\mathsf{lf}\ (i, y))\ x = \eta^{\alpha}_{X_1}\ (i, y)$. But $\eta^{\alpha}_{X_1}\ (i, y) \equiv \alpha_{X_1}\ (\mathsf{lf}\ (i, y))\ \bot\text{-}\mathsf{elim}$ thus it suffices to show that $x = \bot\text{-}\mathsf{elim}$ which is the case as both functions have domain the empty type $\bot$.

- If c is of the form $\mathsf{nd}\ (y \triangleleft z)\ t$, the identity to define is now

  $$\begin{aligned}&\alpha_{X_1}\ (\mathsf{nd}\ (y \triangleleft z)\ t)\ x \\ &= \mu^{\alpha}_{X_1}\ (x\ (\mathsf{inl}\ \star))\ (\lambda p \to \alpha_{X'_1}\ (t\ p)\ (\lambda q \to x\ (\mathsf{inr}\ (p, q))))\end{aligned}$$

  which is readily established by applying Corollary 3.2.7 and by using the induction hypothesis.

Finally, in order to conclude that the opetopic types $X_{>1}$ and $X'_{>1}$ are equivalent — up to a base change— we note that they are both fibrant and that their families of objects have contractible fibres. They are therefore necessarily equivalent to the terminal opetopic type for the appropriate monad. □

We gather these results into the proof of our main theorem.

Proof of Theorem 3.5.3. Lemmas 3.5.4, 3.5.5, 3.5.6, and 3.5.7 define the two directions of the equivalence. Lemmas 3.5.8 and 3.5.9 show that they are both inverse to each other. □

This concludes this section which shown that the type of truncated fibrant $M/$-opetopic types and M-multicategories are equivalent.

3.6 Fibrations of opetopic types

We take advantage of monad families and extend opetopic types to families of opetopic types indexed over an opetopic type. We regard these families as specifying a collection of opetopic cells each indexed over a base cell. We conclude by introducing their fibrations which generalise the notion of fibrant opetopic type to families of opetopic types. We will apply these notions in the context of ∞-categories in a subsequent section.

3.6.1 Families of opetopic types

Definition 3.6.1 (Family of opetopic types)**.** Given a monad family $M\!\downarrow : \mathcal{M}\!\downarrow_M$ and an opetopic type $X : \mathcal{O}_M$, a family of opetopic types indexed by the monad family $M\!\downarrow$ over the opetopic type X is defined by the following data:

- A type family $X\!\downarrow : \mathsf{Fam}\!\downarrow^{X_0}_{M\downarrow}$.
- A family of opetopic types indexed by the monad family $M\!\downarrow/X\!\downarrow$ over the opetopic type $X_{>0}$.

We denote $\mathcal{O}\!\downarrow^X_{M\downarrow}$ the type of opetopic types over an opetopic type $X : \mathcal{O}_M$ and indexed by a monad family $M\!\downarrow : \mathcal{M}\!\downarrow_M$.

3.6.2 Dependent algebras

Likewise, we extend the notions of algebras and fibrant opetopic types to the dependent setting.

Notation. We extend the notation for pasting diagrams to monad families. Let $M{\downarrow} : \mathcal{M}{\downarrow}_M$ be a monad family indexed over a monad M and let $X{\downarrow} : \mathsf{Fam}{\downarrow}^{X}_{M{\downarrow}}$ be a family of types dependent on the indices of $M{\downarrow}$ and on a family $X : \mathsf{Fam}_M$. Let $i{\downarrow} : \mathsf{Idx}{\downarrow}_{M{\downarrow}}\ i$ be an index and let $(c, y) : [\![M]\!]\ X\ i$ be a pasting diagram. We define a notation for pasting diagrams of cells depending on the cells of a base pasting diagram:

$$[\![M{\downarrow}]\!]\ X{\downarrow}\ i{\downarrow}\ (c, y) :\equiv \textstyle\sum_{(c{\downarrow} : \mathsf{Cns}{\downarrow}_{M{\downarrow}}\ i{\downarrow}\ c)} \overrightarrow{X{\downarrow}}\ c{\downarrow}\ y$$

Notation. We extend our notation for depth-2 pasting diagrams of dependent cells. Given a monad family $M{\downarrow} : \mathcal{M}{\downarrow}_M$, a family $X{\downarrow} : \mathsf{Fam}{\downarrow}^{X}_{M{\downarrow}/}$, a cell $t{\downarrow} : X{\downarrow}\ (x{\downarrow} \triangleleft y{\downarrow})\ t$, and a family of cells $u{\downarrow} : \overrightarrow{X{\downarrow}}\ (y{\downarrow} \blacktriangleleft z{\downarrow})\ u$, we define the type of depth-2 pasting diagrams whose root node is decorated with the cell $t{\downarrow}$ and whose family of nodes grafted on the root node is decorated with the cells specified by $u{\downarrow}$. It is defined as the pair $(c{\downarrow}, d{\downarrow}) : [\![M/{\downarrow}]\!]\ X{\downarrow}\ x{\downarrow}\ \theta_{t,u}$ where $c{\downarrow}$ is the depth-2 tree $\mathsf{nd}{\downarrow}\ (x{\downarrow} \triangleleft y{\downarrow})\ (\lambda p \to \eta{\downarrow}_{M{\downarrow}/}\ (\mathsf{Typ}{\downarrow}_M\ y{\downarrow}\ p \triangleleft z{\downarrow}\ p))$ and where $d{\downarrow}$ is a family decorating the nodes of $c{\downarrow}$ defined by the following equations:

$$\begin{aligned} d{\downarrow}\ (\mathsf{inl}\ \star) &:\equiv t{\downarrow} \\ d{\downarrow}\ (\mathsf{inr}\ (p, \mathsf{inl}\ \star)) &:\equiv u{\downarrow}\ p \end{aligned}$$

We denote this pasting diagram $\theta_{t{\downarrow},u{\downarrow}}$.

We start with the definition of dependent 0-algebras.

Definition 3.6.2 (Dependent 0-algebra)**.** Let $M{\downarrow} : \mathcal{M}{\downarrow}_M$ be a monad family, let $X_0 : \mathsf{Fam}_M$, let $X_1 : \mathsf{Fam}_{M/X_0}$, let $X{\downarrow}_0 : \mathsf{Fam}{\downarrow}^{X_0}_{M{\downarrow}}$, and let $X{\downarrow}_1 : \mathsf{Fam}{\downarrow}^{X_1}_{M{\downarrow}/X{\downarrow}_0}$. We say that $(X{\downarrow}_0, X{\downarrow}_1)$ is a dependent 0-algebra if for any base cell $u : X_1\ ((i, y) \triangleleft (c, x))$ and any pasting diagram $(c{\downarrow}, x{\downarrow}) : [\![M{\downarrow}]\!]\ X{\downarrow}_0\ i{\downarrow}\ (c, x)$, the following type is contractible:

$$\textstyle\sum_{(y{\downarrow} : X{\downarrow}_0\ i{\downarrow}\ y)} X{\downarrow}_1\ ((i{\downarrow}, y{\downarrow}) \triangleleft (c{\downarrow}, x{\downarrow}))\ u$$

Let $M{\downarrow} : \mathcal{M}{\downarrow}_M$ be a monad family and let $X{\downarrow}_0 : \mathsf{Fam}{\downarrow}^{X_0}_{M{\downarrow}}$ and $X{\downarrow}_1 : \mathsf{Fam}{\downarrow}^{X_1}_{M{\downarrow}/X{\downarrow}_0}$ be families. If (X_0, X_1) and $(X{\downarrow}_0, X{\downarrow}_1)$ are 0-algebras, we recover

the usual operations of composition of cells.

$$\begin{aligned}
&\alpha{\downarrow_{X{\downarrow_1}}} : \{i : \mathsf{Idx}_M\}\ \{x : [\![M]\!]\ X\ i\}\ \{i{\downarrow} : \mathsf{Idx}{\downarrow_{M{\downarrow}}}\ i\}\ (x{\downarrow} : [\![M{\downarrow}]\!]\ X{\downarrow}\ i{\downarrow}\ x)\\
&\quad \to X{\downarrow_0}\ i{\downarrow}\ (\alpha_{X_1}\ x)\\
&\alpha{\downarrow^{\mathsf{fill}}_{X{\downarrow_1}}} : \{i : \mathsf{Idx}_M\}\ \{x : [\![M]\!]\ X\ i\}\ \{i{\downarrow} : \mathsf{Idx}{\downarrow_{M{\downarrow}}}\ i\}\ (x{\downarrow} : [\![M{\downarrow}]\!]\ X{\downarrow}\ i{\downarrow}\ x)\\
&\quad \to X{\downarrow_1}\ ((\ldots, \alpha{\downarrow_{X_1}}\ x{\downarrow}) \triangleleft x{\downarrow})\ (\alpha^{\mathsf{fill}}_{X_1}\ x)
\end{aligned}$$

They are defined as the two projections of the centre of contraction of the witness of the corresponding dependent 0-algebra using $\alpha^{\mathsf{fill}}_{X_1}\ c\ x$ as base cell. Note the use of the 0-algebra (X_0, X_1) to index the resulting dependent cells. We will often place ourselves in a case where the base opetopic type is fibrant.

Similarly, in the case of dependent 0-algebras of a slice monad $M/$, we obtain the following operators:

$$\begin{aligned}
&\eta{\downarrow^{\alpha}_{X{\downarrow_1}}} : \{i : \mathsf{Idx}_M\}\ (i{\downarrow} : \mathsf{Idx}{\downarrow_{M{\downarrow}}}\ i) \to X{\downarrow_0}\ (i{\downarrow} \triangleleft \eta{\downarrow_{M{\downarrow}}}\ i{\downarrow})\ (\eta^{\alpha}_{X_1}\ i)\\
&\eta{\downarrow^{\alpha^{\mathsf{fill}}}_{X{\downarrow_1}}} : \{i : \mathsf{Idx}_M\}\ (i{\downarrow} : \mathsf{Idx}{\downarrow_{M{\downarrow}}}\ i) \to X{\downarrow_1}\ ((\ldots, \eta{\downarrow^{\alpha}_{X{\downarrow_1}}}\ i{\downarrow}) \triangleleft (\mathsf{If}{\downarrow}\ i{\downarrow}, \bot\text{-elim}))\ (\eta^{\alpha^{\mathsf{fill}}}_{X_1}\ i)
\end{aligned}$$

with defininings equations:

$$\begin{aligned}
\eta{\downarrow^{\alpha}_{X{\downarrow_1}}}\ i{\downarrow} &:\equiv \alpha{\downarrow_{X{\downarrow_1}}}\ (\mathsf{If}{\downarrow}\ i{\downarrow})\ \bot\text{-elim}\\
\eta{\downarrow^{\alpha^{\mathsf{fill}}}_{X{\downarrow_1}}}\ i{\downarrow} &:\equiv \alpha{\downarrow^{\mathsf{fill}}_{X{\downarrow_1}}}\ (\mathsf{If}{\downarrow}\ i{\downarrow})\ \bot\text{-elim}
\end{aligned}$$

In addition, we have a biased composition along with its filler:

$$\begin{aligned}
&\mu{\downarrow^{\alpha}_{X{\downarrow_1}}} : \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\}\\
&\quad \to \{f : X_0\ (x \triangleleft y)\}\ \{g : \overrightarrow{X_0}\ (y \blacktriangleleft z)\}\\
&\quad \to \{x{\downarrow} : \mathsf{Idx}{\downarrow_{M{\downarrow}}}\ x\}\ \{y{\downarrow} : \mathsf{Cns}{\downarrow_{M{\downarrow}}}\ x{\downarrow}\ y\}\ \{z{\downarrow} : \overrightarrow{\mathsf{Cns}{\downarrow_{M{\downarrow}}}}\ y{\downarrow}\ z\}\\
&\quad \to (f{\downarrow} : X{\downarrow_0}\ (x{\downarrow} \triangleleft y{\downarrow})\ f)\ (g{\downarrow} : \overrightarrow{X{\downarrow_0}}\ (y{\downarrow} \blacktriangleleft z{\downarrow})\ g)\\
&\quad \to X{\downarrow_0}\ (x{\downarrow} \triangleleft \mu{\downarrow_{M{\downarrow}}}\ y{\downarrow}\ z{\downarrow})\ (\mu^{\alpha}_{X_1}\ f\ g)\\
&\mu{\downarrow^{\alpha^{\mathsf{fill}}}_{X{\downarrow_1}}} : \{x : \mathsf{Idx}_M\}\ \{y : \mathsf{Cns}_M\ x\}\ \{z : \overrightarrow{\mathsf{Cns}_M}\ y\}\\
&\quad \to \{f : X_0\ (x \triangleleft y)\}\ \{g : \overrightarrow{X_0}\ (y \blacktriangleleft z)\}\\
&\quad \to \{x{\downarrow} : \mathsf{Idx}{\downarrow_{M{\downarrow}}}\ x\}\ \{y{\downarrow} : \mathsf{Cns}{\downarrow_{M{\downarrow}}}\ x{\downarrow}\ y\}\ \{z{\downarrow} : \overrightarrow{\mathsf{Cns}{\downarrow_{M{\downarrow}}}}\ y{\downarrow}\ z\}\\
&\quad \to (f{\downarrow} : X{\downarrow_0}\ (x{\downarrow} \triangleleft y{\downarrow})\ f)\ (g{\downarrow} : \overrightarrow{X{\downarrow_0}}\ (y{\downarrow} \blacktriangleleft z{\downarrow})\ g)\\
&\quad \to X{\downarrow_1}\ ((\ldots, \mu{\downarrow^{\alpha}_{X{\downarrow_1}}}\ f{\downarrow}\ g{\downarrow}) \triangleleft \theta_{f{\downarrow}, g{\downarrow}})\ (\mu^{\alpha^{\mathsf{fill}}}_{X_1}\ f\ g)
\end{aligned}$$

with defininings equations:

$$\begin{aligned}
\mu{\downarrow^{\alpha}_{X{\downarrow_1}}}\ f{\downarrow}\ g{\downarrow} &:\equiv \alpha{\downarrow_{X{\downarrow_1}}}\ \theta_{f{\downarrow}, g{\downarrow}}\\
\mu{\downarrow^{\alpha^{\mathsf{fill}}}_{X{\downarrow_1}}}\ f{\downarrow}\ g{\downarrow} &:\equiv \alpha{\downarrow^{\mathsf{fill}}_{X{\downarrow_1}}}\ \theta_{f{\downarrow}, g{\downarrow}}
\end{aligned}$$

Algebraic laws In the presence of a further dependent algebra, we recover a dependent version of the algebraic laws satisfied by a 1-algebra.

Let $M\downarrow : \mathcal{M}\downarrow_M$ be a monad family indexed over a monad M and let $X\downarrow_0 : \mathsf{Fam}\downarrow^{X_0}_{M\downarrow}$, $X\downarrow_1 : \mathsf{Fam}\downarrow^{X_1}_{M\downarrow/X\downarrow_0}$, and $X\downarrow_2 : \mathsf{Fam}\downarrow^{X_2}_{M\downarrow/X\downarrow_0/X\downarrow_1}$ be dependent families over the families X_0, X_1, and X_2. We suppose that $(X\downarrow_0, X\downarrow_1)$ and $(X\downarrow_1, X\downarrow_2)$ are dependent 0-algebras and that (X_0, X_1) and (X_1, X_2) are 0-algebras.

In that case, $(X\downarrow_0, X\downarrow_1)$ satisfies the usual algebraic laws albeit with an additional transport:

$$\begin{aligned}
&\mathsf{transport}^{X\downarrow_0\ i\downarrow}\ \alpha\text{-}\eta\ (\alpha\downarrow_{X\downarrow_1}\ (\eta\downarrow_{X\downarrow_1}\ i\downarrow)\ x\downarrow) = x\downarrow\ (\mathsf{pos}^{\eta}\ i)\\
&\mathsf{transport}^{X\downarrow_0\ i\downarrow}\ \alpha\text{-}\mu\ (\alpha\downarrow_{X\downarrow_1}\ (\mu\downarrow_{X\downarrow_1}\ c\downarrow\ d\downarrow)\ x\downarrow)\\
&\quad = \alpha\downarrow_{X\downarrow_1}\ c\downarrow\ (\lambda p \to \alpha\downarrow_{X\downarrow_1}\ (d\downarrow\ p)\ (\lambda q \to x\downarrow\ (\mathsf{pair}^{\mu}\ p\ q)))
\end{aligned}$$

We do not prove these laws as they are a straightforward generalisation of the non-dependent ones. We say that $(X\downarrow_0, X\downarrow_1)$ is a dependent 1-algebra.

Similarly, if the monad M is of the form $M'/$ for some monad M', the operation $\mu\downarrow^{\alpha}_{X\downarrow_1}$ is associative and unital with unit $\eta\downarrow^{\alpha}_{X\downarrow_1}$:

$$\begin{aligned}
&\mathsf{transport}^{X\downarrow_0\ (x\downarrow \triangleleft y\downarrow)}\ \mu^{\alpha}_{X_1}\text{-unit-r}\ (\mu\downarrow^{\alpha}_{X\downarrow_1} f\downarrow\ (\lambda p \to \eta\downarrow^{\alpha}_{X\downarrow_1}(\mathsf{Typ}\downarrow_{M\downarrow}\ y\downarrow\ p))) = f\downarrow\\
&\mathsf{transport}^{X\downarrow_0\ (x\downarrow \triangleleft y\downarrow\ (\mathsf{pos}^{\eta}\ i))}\ \mu^{\alpha}_{X_1}\text{-unit-l}\ (\mu\downarrow^{\alpha}_{X\downarrow_1}(\eta\downarrow_{X\downarrow_1}\ i\downarrow)\ f\downarrow) = f\downarrow\ (\mathsf{pos}^{\eta}\ i)\\
&\mathsf{transport}^{X\downarrow_0\ (x\downarrow \triangleleft \mu\downarrow_{M\downarrow}\ y\downarrow\ (\lambda p \to \mu\downarrow_{M\downarrow}\ (z\downarrow\ p)\ (\lambda q \to t\downarrow\ (\mathsf{pair}^{\mu}\ p\ q))))}\ \mu^{\alpha}_{X_1}\text{-assoc}\\
&\quad (\mu\downarrow^{\alpha}_{X\downarrow_1}\ f\downarrow\ (\lambda p \to \mu\downarrow^{\alpha}_{X\downarrow_1}\ (g\downarrow\ p)\ (\lambda q \to h\downarrow\ (\mathsf{pair}^{\mu}\ p\ q))))\\
&\qquad = \mu\downarrow^{\alpha}_{X\downarrow_1}\ (\mu\downarrow^{\alpha}_{X\downarrow_1}\ f\downarrow\ g\downarrow)\ h\downarrow
\end{aligned}$$

3.6.3 Fibrations of opetopic types

We are now able to define fibrations of opetopic types.

Definition 3.6.3 (Fibration of opetopic types)**.** Let $M\downarrow : \mathcal{M}\downarrow_M$ be a monad family indexed over a monad M and let $X\downarrow$ be a $M\downarrow$-opetopic type over a M-opetopic type X. We say that $X\downarrow$ is a fibration of opetopic types if it satisfies the following conditions:

- $(X\downarrow_0, X\downarrow_1)$ is a dependent 0-algebra.
- $X\downarrow_{>0}$ is a fibration of opetopic types.

We also define fibrations of ∞-categories which we will use in Section 3.9.

Definition 3.6.4 (Fibration of ∞-categories)**.** Let X be an ∞-category. A family of opetopic types $X\downarrow : O\downarrow^{X}_{\mathsf{Id}\downarrow}$ over X is a fibration of ∞-categories if $X\downarrow_{>0}$ is a fibration of opetopic types. We denote $\infty\text{-}\mathsf{category}\downarrow_X$ the type of fibrations of ∞-categories over X.

We expect the fibres of a fibration of ∞-categories to be ∞-categories. However, there is no easy way to define such a construction in our current formalism.

3.6.4 Dependent sums of opetopic types

Opetopic types are closed under dependent sums. Let $X{\downarrow} : O{\downarrow}^{X}_{M{\downarrow}}$ be a family of opetopic types indexed over an opetopic type $X : O_M$. We form the opetopic type

$$\Sigma^o(X, X{\downarrow}) : O_{\Sigma^m(M,M{\downarrow})}$$

It is defined by coinduction by the following equations

$$\begin{aligned}\Sigma^o(X, X{\downarrow})_0 &:\equiv \Sigma{\downarrow}(X_0, X{\downarrow}_0)\\ \Sigma^o(X, X{\downarrow})_{>0} &:\equiv f^*(\Sigma^o(X_{>0}, X{\downarrow}_{>0}))\end{aligned}$$

where the second equation specifies a reindexing (Definition 3.1.3) along a monad map f whose type is

$$\Sigma^m(M, M{\downarrow})/\Sigma{\downarrow}(X_0, X{\downarrow}_0) \to_m \Sigma^m(M/X_0, M{\downarrow}/X{\downarrow}_0)$$

and which is defined as

$$\Sigma^m/(X_0^* M, X{\downarrow}_0^* M{\downarrow}) \circ_m \Sigma^{m*}(M, M{\downarrow}, X_0, X{\downarrow}_0)/$$

We end this short section by proving that fibrant opetopic types are closed under dependent sums.

Lemma 3.6.5. *Let $X{\downarrow} : O{\downarrow}^{X}_{M{\downarrow}}$ be a fibration of opetopic types indexed over a fibrant opetopic type $X : O_M$. The opetopic type $\Sigma^o(X, X{\downarrow})$ is fibrant.*

Proof. We first have to show that for any index $(i, i{\downarrow}) : \mathsf{Idx}_{\Sigma^m(M,M{\downarrow})}$, any constructor $(c, c{\downarrow}) : \mathsf{Cns}_{\Sigma^m(M,M{\downarrow})}\,(i, i{\downarrow})$, and any family of elements $x : \overrightarrow{\Sigma^m(M, M{\downarrow})}\, c$, the following type is contractible:

$$\textstyle\sum_{((y,y{\downarrow}):\ \sum_{(y:X_0\ i)} X{\downarrow}_0\ i{\downarrow}\ y)} \sum_{(z:X_1\ ((i,y)\triangleleft(c,\mathsf{pr}_1\circ x)))} X{\downarrow}_1\ ((i{\downarrow}, y{\downarrow}) \triangleleft (c{\downarrow}, \mathsf{pr}_2 \circ x))\ z$$

Reordering the sigma types, it is equivalent to prove that the following type is contractible:

$$\textstyle\sum_{((y,z):\ \sum_{(y:X_0\ i)} X_1\ ((i,y)\triangleleft(c,\mathsf{pr}_1\circ x)))} \sum_{(y{\downarrow}:X{\downarrow}_0\ i{\downarrow}\ y)} X{\downarrow}_1\ ((i{\downarrow}, y{\downarrow}) \triangleleft (c{\downarrow}, \mathsf{pr}_2 \circ x))\ z$$

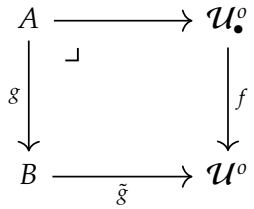
But this follows from the fact that (X_0, X_1) and $(X{\downarrow}_0, X{\downarrow}_1)$ are 0-algebras. Sigma types of contractible types are again contractible.

It remains to prove that the opetopic type $f^*(\Sigma^o(X_{>0}, X{\downarrow}_{>0}))$ is fibrant where f is the appropriate monad morphism. But fibrant opetopic types are closed under reindexing and the coinduction hypothesis states that $\Sigma^o(X_{>0}, X{\downarrow}_{>0})$ is fibrant as $X_{>0}$ and $X{\downarrow}_{>0}$ are themselves fibrant which concludes this proof. □

3.7 The opetopic universe of types

A compelling argument in favour of an opetopic theory of types is the ease with which we can define the universal opetopic fibration of types. Recall that in type theory, the universal fibration of types is the first projection of pointed types $\mathsf{pr}_1 : \sum_{(A:\mathcal{U})} A \to \mathcal{U}$ (see Section 4.8 of the HoTT book).

The universal fibration of Id-opetopic types is the unique fibration $f : \mathcal{U}^o_\bullet \to \mathcal{U}^o$ such that any fibration of Id-opetopic types $g : A \to B$ arises uniquely as the pullback of some fibration $\tilde{g} : B \to \mathcal{U}^o$ along f.



In this section, we will only define the map $f : \mathcal{U}^o_\bullet \to \mathcal{U}^o$ regarded as a family of opetopic types $\mathcal{U}^o_\bullet : O\downarrow^{\mathcal{U}^o}_{\mathsf{Id}\downarrow}$ over $\mathcal{U}^o : O_{\mathsf{Id}}$ without proving that it is a fibration, let alone that it is the universal one which is left for future work.

Intuitively, $\mathcal{U}^o$ is the opetopic type whose objects are types and whose higher cells are fibrant relations: relations which intuitively correspond to equivalences. $\mathcal{U}^o_\bullet$ is then the family of opetopic types whose objects over a type X are the elements of X and whose higher cells over a given fibrant relation and a frame — the data of a source pasting diagram and a target cell — are the witnesses that this relation holds for the given frame.

First, note that for any monad $M : \mathcal{M}$ and monad family $M\downarrow : \mathcal{M}\downarrow_M$ over M, there are two canonical type families $\mathsf{Rel}_{M\downarrow} : \mathsf{Fam}_M$ and $\mathsf{Rel}\downarrow_{M\downarrow} : \mathsf{Fam}\downarrow^{\mathsf{Rel}_{M\downarrow}}_{M\downarrow}$ defined by the following equations:

$$\mathsf{Rel}_{M\downarrow}\ i :\equiv \mathsf{Idx}\downarrow_{M\downarrow}\ i \to \mathcal{U}$$
$$\mathsf{Rel}\downarrow_{M\downarrow}\ i\downarrow\ R :\equiv R\ i\downarrow$$

We will see that these two families arrange themselves into opetopic types. The type family $\mathsf{Rel}_{M\downarrow}$ can be regarded as a family of relations over $\mathsf{Idx}\downarrow_{M\downarrow}\ i$ while the family $\mathsf{Rel}\downarrow_{M\downarrow}$ associates, to any relation R and any element $i\downarrow : \mathsf{Idx}\downarrow_{M\downarrow} i$, the type $R\ i\downarrow$ of witnesses that $i\downarrow$ satisfies the relation R.

As we are interested in fibrant relations, we specialise our type families in the case the indexing monad is of the form $M\downarrow/X\downarrow$ for some monad family $M\downarrow : \mathcal{M}\downarrow_M$ and dependent family $X\downarrow : \mathsf{Fam}\downarrow^X_{M\downarrow}$:

$$\mathsf{Rel}^f_{M\downarrow,X\downarrow}\ i :\equiv \sum_{(R:\mathsf{Rel}_{M\downarrow/X\downarrow}\ i)} \mathsf{is\text{-}algebraic}\ i\ R$$
$$\mathsf{Rel}\downarrow^f_{M\downarrow,X\downarrow}\ i\downarrow\ (R, R\text{-}\mathsf{is\text{-}algebraic}) :\equiv R\ i\downarrow$$

where is-algebraic is a predicate defined as follows:

$$\begin{aligned}&\mathsf{is\text{-}algebraic}\ ((i,y)\triangleleft(c,x))\ R :\equiv \{i{\downarrow} : \mathsf{Idx}{\downarrow}_{M{\downarrow}}\ i\}\ (c{\downarrow} : \mathsf{Cns}{\downarrow}_{M{\downarrow}}\ i{\downarrow}\ c)\\ &\quad\to (x{\downarrow} : \mathsf{Fam}{\downarrow}^{X{\downarrow}}_{M{\downarrow}}\ c{\downarrow}\ x)\\ &\quad\to \mathsf{is\text{-}contr}\ (\textstyle\sum_{(y{\downarrow}:X{\downarrow}\ i{\downarrow}\ y)} R((i{\downarrow},y{\downarrow}),(c{\downarrow},x{\downarrow})))\end{aligned}$$

We now define the universe.

Definition 3.7.1 (The opetopic universe of types)**.** The opetopic universe of types and their fibrant relations $\mathcal{U}^o$ is the Id-opetopic type defined by the equations

$$\begin{aligned}\mathcal{U}^o_0 &:\equiv \mathsf{Rel}_{\mathsf{Id}{\downarrow}}\\ \mathcal{U}^o_{>0} &:\equiv \mathcal{U}^o_{\mathsf{Id}{\downarrow},\mathsf{Rel}_{\mathsf{Id}{\downarrow}}}\end{aligned}$$

where the M/X-opetopic type $\mathcal{U}^o_{M{\downarrow},X{\downarrow}}$ is defined coinductively by the equations

$$\begin{aligned}(\mathcal{U}^o_{M{\downarrow},X{\downarrow}})_0 &:\equiv \mathsf{Rel}^f_{M{\downarrow},X{\downarrow}}\\ (\mathcal{U}^o_{M{\downarrow},X{\downarrow}})_{>0} &:\equiv \mathcal{U}^o_{M{\downarrow}/X{\downarrow},\mathsf{Rel}^f_{M{\downarrow},X{\downarrow}}}\end{aligned}$$

for any monad family $M{\downarrow} : \mathcal{M}{\downarrow}_M$ and family $X{\downarrow} : \mathsf{Fam}{\downarrow}^X_{M{\downarrow}}$.

We continue with the definition of the universal fibration.

Definition 3.7.2 (The universal opetopic fibration of types)**.** The universal opetopic fibration of types $\mathcal{U}^o_\bullet$ is the family of opetopic types over $\mathcal{U}^o$ indexed by the monad family $\mathsf{Id}{\downarrow}$:

$$\begin{aligned}\mathcal{U}^o_{\bullet\,0} &:\equiv \mathsf{Rel}{\downarrow}_{\mathsf{Id}{\downarrow}}\\ \mathcal{U}^o_{\bullet\,>0} &:\equiv \mathcal{U}^o_{\bullet\ \mathsf{Id}{\downarrow},\mathsf{Rel}{\downarrow}_{\mathsf{Id}{\downarrow}}}\end{aligned}$$

where the family of opetopic types $\mathcal{U}^o_{\bullet\ \mathsf{Id}{\downarrow},\mathsf{Rel}{\downarrow}_{\mathsf{Id}{\downarrow}}}$ over $\mathcal{U}^o_{\mathsf{Id}{\downarrow},\mathsf{Rel}{\downarrow}_{\mathsf{Id}{\downarrow}}}$ indexed by the monad family $M{\downarrow}/X{\downarrow}$ is defined coinductively by the equations

$$\begin{aligned}(\mathcal{U}^o_{\bullet\ M{\downarrow},X{\downarrow}})_0 &:\equiv \mathsf{Rel}{\downarrow}^f_{M{\downarrow},X{\downarrow}}\\ (\mathcal{U}^o_{\bullet\ M{\downarrow},X{\downarrow}})_{>0} &:\equiv \mathcal{U}^o_{\bullet\ M{\downarrow}/X{\downarrow},\mathsf{Rel}{\downarrow}^f_{M{\downarrow},X{\downarrow}}}\end{aligned}$$

for any monad family $M{\downarrow} : \mathcal{M}{\downarrow}_M$ and family $X{\downarrow} : \mathsf{Fam}{\downarrow}^X_{M{\downarrow}}$.

This concludes the definition of the universal fibration of fibrant opetopic types.

3.8 The opetopic type associated with a type

It is well-known that types are ∞-groupoids (Lumsdaine 2010; Van Den Berg and Garner 2011). It is therefore expected that we can define the fibrant Id-opetopic type associated with a type X. To this effect, we will use the monad family $\mathsf{Id}{\downarrow}_X$.

First, note that for any monad M and monad family $M{\downarrow} : \mathcal{M}{\downarrow}_M$, there are two canonical type families $\mathsf{Rel}_{M{\downarrow}} : \mathsf{Fam}_M$ and $\mathsf{Rel}{\downarrow}_{M{\downarrow}} : \mathsf{Fam}{\downarrow}_{M{\downarrow}}^{\mathsf{Rel}_{M{\downarrow}}}$ defined by the following equations:

$$\begin{aligned}\mathsf{Rel}_{M{\downarrow}} &:\equiv \mathsf{Idx}{\downarrow}_{M{\downarrow}}\\ \mathsf{Rel}{\downarrow}_{M{\downarrow}}\ i{\downarrow}\ j{\downarrow} &:\equiv i{\downarrow} = j{\downarrow}\end{aligned}$$

This is enough to define the opetopic type associated with a type.

Definition 3.8.1 (Associated opetopic type)**.** For any family of monad $M{\downarrow} : \mathcal{M}{\downarrow}_M$, we define the M-opetopic type $M{\downarrow}^o$:

$$\begin{aligned}M{\downarrow}^o_0 &:\equiv \mathsf{Idx}{\downarrow}_{M{\downarrow}}\\ M{\downarrow}^o_{>0}\ i &:\equiv (M{\downarrow}/\mathsf{Rel}{\downarrow}_{M{\downarrow}})^o\end{aligned}$$

The opetopic type associated with a type X is the opetopic type $\mathsf{Id}{\downarrow}^o_X$.

It remains to establish that $\mathsf{Id}{\downarrow}^o_X$ is fibrant. To this effect, we introduce a few more results about monad families and in particular those of the form $M{\downarrow}/\mathsf{Rel}{\downarrow}_{M{\downarrow}}$.

First, we will need the notion of fibration of monads.

Definition 3.8.2 (Fibration of monads)**.** Let $M{\downarrow}$ be a monad family indexed over a monad M. We say that $M{\downarrow}$ is a fibration of monads if, for any constructor $c : \mathsf{Cns}_M\ i$ and family of indices $x : \overrightarrow{\mathsf{Idx}{\downarrow}_M}\ c$, there is an index $i{\downarrow} : \mathsf{Idx}{\downarrow}_{M{\downarrow}}\ i$, a constructor $c{\downarrow} : \mathsf{Cns}{\downarrow}_{M{\downarrow}}\ i{\downarrow}\ c$ satisfying $\mathsf{Typ}{\downarrow}_{M{\downarrow}}\ c{\downarrow} = x$. Moreover, this data is unique. That is, the following type is contractible:

$$\textstyle\sum_{(i{\downarrow}:\mathsf{Idx}{\downarrow}_{M{\downarrow}}\ i)} \sum_{(c{\downarrow}:\mathsf{Cns}{\downarrow}_{M{\downarrow}}\ i{\downarrow}\ c)} \mathsf{Typ}{\downarrow}_{M{\downarrow}}\ c{\downarrow} = x$$

In particular, one can establish that a fibration of monads $M{\downarrow}$ implies that $(\mathsf{Idx}{\downarrow}_{M{\downarrow}}, \mathsf{Idx}{\downarrow}_{M{\downarrow}/\mathsf{Rel}{\downarrow}_{M{\downarrow}}})$ is a 0-algebra using the equivalence

$$\begin{aligned}&(\textstyle\sum_{(i{\downarrow}:\mathsf{Idx}{\downarrow}_{M{\downarrow}}\ i)} \sum_{(c{\downarrow}:\mathsf{Cns}{\downarrow}_{M{\downarrow}}\ i{\downarrow}\ c)} \mathsf{Typ}{\downarrow}_{M{\downarrow}}\ c{\downarrow} = x)\\ &\quad\simeq (\textstyle\sum_{(i{\downarrow}:\mathsf{Idx}{\downarrow}_{M{\downarrow}}\ i)} \mathsf{Idx}{\downarrow}_{M{\downarrow}/\mathsf{Rel}{\downarrow}_{M{\downarrow}}}\ ((i, i{\downarrow}) \triangleleft (c, x)))\end{aligned} \tag{3.1}$$

We now show that a particular slice construction is a fibration of monads.

Lemma 3.8.3. *Let $M{\downarrow}$ be a monad family indexed over a monad M. The monad family $M{\downarrow}/\mathsf{Rel}{\downarrow}_{M{\downarrow}}$ is a fibration of monads over $M/\mathsf{Rel}_{M{\downarrow}}$.*

Proof. We only sketch the proof, the fully detailed proof can be found in the formalisation. Let $i : \mathsf{Idx}_{M/\mathsf{Rel}_{M\downarrow}}$, $c : \mathsf{Cns}_{M/\mathsf{Rel}_{M\downarrow}}\ i$, and $x : \overrightarrow{\mathsf{Idx}\downarrow_{M\downarrow/\mathsf{Rel}\downarrow_{M\downarrow}}}\ c$. We prove that the necessary data exists and is unique by induction on c. Essentially, we show that a pasting diagram c can be uniquely lifted to a pasting diagram dependent on c — therefore having the same form — and whose nodes are given by x.

- If c is of the form $\mathsf{lf}\ (i, i\downarrow)$, it uniquely lifts to $\mathsf{lf}\downarrow\ (i\downarrow, \mathsf{refl})$ and the identity $\mathsf{Typ}\downarrow_{M\downarrow/\mathsf{Rel}\downarrow_{M\downarrow}}\ (\mathsf{lf}\downarrow\ (i\downarrow, \mathsf{refl})) = x$ is uniquely witnessed as the two sides of the identity are functions whose domain is the empty type.
- If c is of the form $\mathsf{nd}\ (y \triangleleft z)\ t$, it uniquely lifts to $\mathsf{nd}\downarrow\ (x\ (\mathsf{inl}\star))\ t\downarrow$ such that

  $$\mathsf{Typ}\downarrow_{M\downarrow/\mathsf{Rel}\downarrow_{M\downarrow}}\ (\mathsf{nd}\downarrow\ (x\ (\mathsf{inl}\star))\ t\downarrow) = x$$

  where $t\downarrow$ is the induction hypothesis applied to $t\ p$ and $\lambda q \longrightarrow x\ (\mathsf{inr}\ (p, q))$ for any position $p : \mathsf{Pos}_{M/\mathsf{Rel}_{M\downarrow}}\ z$ up to a tedious transport that we do not detail in order to get the indexing right.

□

This suffices to define a fibrant opetopic type from a fibration of monads.

Lemma 3.8.4. *Let $M\downarrow$ be a fibration of monads over a monad M. Then the opetopic type $M\downarrow^o$ is fibrant.*

Proof. Proving that $(M\downarrow^o_0, M\downarrow^o_1)$ is a 0-algebra essentially uses the fact that $M\downarrow$ is a fibration of monads along with Equivalence 3.1.

We conclude by applying the coinductive hypothesis to the monad family $M\downarrow/\mathsf{Rel}\downarrow_{M\downarrow}$ which is a fibration of monads according to Lemma 3.8.3. □

As a corollary, we obtain that $\mathsf{Id}\downarrow^o_X$ is fibrant.

Corollary 3.8.5. *The opetopic type $\mathsf{Id}\downarrow^o_X$ is fibrant.*

Proof. It is easy to check that $\mathsf{Id}\downarrow_X$ is a fibration of monads which allows to conclude that $\mathsf{Id}\downarrow^o_X$ is fibrant by virtue of Lemma 3.8.4. □

The operation of defining a fibrant opetopic type from a type and the operation of extracting the type of objects of a fibrant opetopic type are actually inverse to each other. This statement corresponds to the internalisation of the result stating that types are ∞-groupoids (FINSTER, ALLIOUX and SOZEAU 2021) whose proof is not detailed here.

3.9 Adjunctions

In order to display the capabilities of our formalism, we define the notion of adjoint functors following Lurie's definition (LURIE 2009, Definition 5.2.2.1) as an ∞-category over the interval.

Notation. In this section we will only deal with ∞-categories which are Id-indexed opetopic types X whose morphisms are therefore unary. We will denote their type of objects X_0 instead of $X_0\ \star$. Also, for any pair of objects $x, y : X_0$, we will write $X_1\ (x \triangleright y)$ for their type of morphisms from x to y instead of the more convoluted $X_1\ ((\star, y) \triangleleft (\star, \lambda p \to x))$. Similarly for fibrations of ∞-categories $X{\downarrow}$ over an ∞-category X, we will denote $X{\downarrow}_0\ x$ their type of objects over an object x instead of $X{\downarrow}_0\ \star\ x$. Also, for any pair of objects $x{\downarrow} : X{\downarrow}_0\ x$ and $y{\downarrow} : X{\downarrow}_0\ y$ along with a base morphism $f : X_1\ (x \triangleright y)$, we will write $X{\downarrow}_1\ (x{\downarrow} \triangleright y{\downarrow})\ f$ for the type of morphisms from $x{\downarrow}$ to $y{\downarrow}$ over f in place of $X{\downarrow}_1\ ((\star, y{\downarrow}) \triangleleft (\star, \lambda p \to x{\downarrow}))\ f$.

Finally, we will denote the composition of morphisms $g \circ f$ instead of $\mu^{\alpha}_{X_2}\ g\ (\eta\text{-}\mathsf{dec}\ f)$ or $\mu{\downarrow}^{\alpha}_{X{\downarrow}_2}\ g\ (\eta\text{-}\mathsf{dec}\ f)$ in the dependent case. As for the unit, we will write 1_x instead of $\eta^{\alpha}_{X_2}\ x$ or $\eta{\downarrow}^{\alpha}_{X_2}\ x$ in the dependent case.

We start with the definition of the higher categorical counterpart of Grothendieck (op)fibrations which rely on the existence of enough (co)cartesian morphisms.

Definition 3.9.1 (Cartesian morphism)**.** Let $X{\downarrow} : \infty\text{-}\mathsf{category}{\downarrow}_X$ be a fibration of ∞-categories over an ∞-category X. Let $f{\downarrow} : X{\downarrow}_1\ (x{\downarrow} \triangleright y{\downarrow})\ f$ be a morphism over $f : X_1\ (x \triangleright y)$, $f{\downarrow}$ is a *cartesian* morphism if for any morphism $g{\downarrow} : X_1\ (z{\downarrow} \triangleright y{\downarrow})\ g$ over $g : X_1\ (z \triangleright y)$ and any morphism $h : X_1\ (z \triangleright x)$ such that there is an identity $p : f \circ h = g$, there exists a unique morphism $h{\downarrow} : X{\downarrow}_1\ (z{\downarrow} \triangleright x{\downarrow})\ h$ such that

$$\mathsf{transport}^{X{\downarrow}_1\ (z{\downarrow} \triangleright y{\downarrow})}\ p\ (f{\downarrow} \circ h{\downarrow}) = g{\downarrow}$$

That is, the following type is contractible

$$\textstyle\sum_{(h{\downarrow} : X{\downarrow}_1\ (z{\downarrow} \triangleright x{\downarrow})\ h)} \mathsf{transport}^{X{\downarrow}_1\ (z{\downarrow} \triangleright y{\downarrow})}\ p\ (f{\downarrow} \circ h{\downarrow}) = g{\downarrow}$$

This leads us to the definition of Grothendieck fibration.

Definition 3.9.2 (Grothendieck fibration)**.** Let $X{\downarrow} : \infty\text{-}\mathsf{category}{\downarrow}_X$ be a fibration of ∞-categories over X. $X{\downarrow}$ is a Grothendieck fibration if for any morphism $f : X_1\ (x \triangleright y)$ and any object $y{\downarrow} : X{\downarrow}_0\ y$ there is an object $x{\downarrow} : X{\downarrow}_0\ x$ and a cartesian morphism $f{\downarrow} : X{\downarrow}_1\ (x{\downarrow} \triangleright y{\downarrow})\ f$ over f.

Similarly, we define cocartesian morphisms and opfibrations.

Definition 3.9.3 (Cocartesian morphism)**.** Let $X{\downarrow} : \infty\text{-}\mathsf{category}{\downarrow}_X$ be a fibration of ∞-categories over an ∞-category X. Let $f{\downarrow} : X{\downarrow}_1\ (x{\downarrow} \triangleright y{\downarrow})\ f$ be a morphism over $f : X_1\ (x \triangleright y)$, $f{\downarrow}$ is a *cocartesian* morphism if for any morphism $g{\downarrow} : X_1\ (x{\downarrow} \triangleright z{\downarrow})\ g$ over $g : X_1\ (x \triangleright z)$ and any morphism $h : X_1\ (y \triangleright z)$ such that there exists an identity $p : h \circ f = g$, there exists a unique morphism $h{\downarrow} : X{\downarrow}_1\ (y{\downarrow} \triangleright z{\downarrow})\ h$ such that

$$\mathsf{transport}^{X{\downarrow}_1\ (x{\downarrow} \triangleright z{\downarrow})}\ p\ (h{\downarrow} \circ f{\downarrow}) = g{\downarrow}$$

That is, the following type is contractible

$$\sum_{(h\downarrow : X\downarrow_1\ (y\downarrow \triangleright z\downarrow))\ h)} \mathsf{transport}^{X\downarrow_1\ (x\downarrow \triangleright z\downarrow)}\ p\ (h\downarrow \circ f\downarrow) = g\downarrow$$

Definition 3.9.4 (Grothendieck opfibration)**.** Let $X\downarrow : \infty\text{-}\mathsf{category}\downarrow_X$ be a fibration of ∞-categories over X. $X\downarrow$ is a Grothendieck opfibration if for any morphism $f : X_1\ (x \triangleright y)$ and any object $x\downarrow : X\downarrow_0\ x$ there is an object $y\downarrow : X\downarrow_0\ y$ and a cocartesian morphism $f\downarrow : X\downarrow_1\ (x\downarrow \triangleright y\downarrow)\ f$ over f.

Next, we define the interval ∞-category I.

Definition 3.9.5. The interval I is the ∞-category defined as follows:

- The type of objects I_0 is $\mathbf{2}$ whose sole elements are 0 and 1.
- The family of morphisms is defined by case analysis on the objects:

$$I_1\ (x \triangleright y) :\equiv \begin{cases} \mathbf{0} & \text{if } x \equiv 1 \text{ and } y \equiv 0 \\ \mathbf{1} & \text{otherwise} \end{cases}$$

- $I_{>1}$ is $\mathbf{1}^o_{\mathsf{Id}/I_0/I_1}$, the terminal opetopic type for the monad $\mathsf{Id}/I_0/I_1$.

It is easy to see that the opetopic type $I_{>0}$ is fibrant. I is therefore an ∞-category.

At last, we define adjunctions following Lurie's definition (LURIE 2009, Theorem 5.2.2.1) although we cannot establish a formal equivalence between each fibre and their corresponding ∞-category (i.e., we cannot state that the fibres of a fibration of ∞-categories are ∞-categories in the present formalism).

Definition 3.9.6 (Adjunction)**.** An adjunction is a fibration of ∞-categories over the interval I which is both a Grothendieck fibration and a Grothendieck opfibration.

In that case, the adjunction is between the two ∞-categories defined by the fibres. We now show that this definition is compatible with a more explicit characterisation of adjunction in low dimensions. We will in particular show that it induces a map on objects, a map on morphisms which satisfies the usual laws of functors, and, moreover, an equivalence of homs stemming from the adjunction.

We first check the functor part.

Lemma 3.9.7. *Given a Grothendieck opfibration over the interval $C : \infty\text{-}\mathsf{category}\downarrow_I$, we can define two functions between the two fibres:*

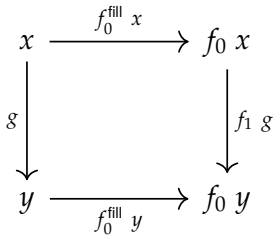
$$f_0 : C_0\ 0 \to C_0\ 1$$
$$f_1 : \{x\ y : C_0\ 0\} \to C_1\ (x \triangleleft y)\ 1_0 \to C_1\ (f_0\ x \triangleleft f_0\ y)\ 1_1$$

satisfying the following laws:

$$f_1\ (g \circ f) = f_1\ g \circ f_1\ f$$
$$f_1\ 1_x = 1_{f_0\ x}$$

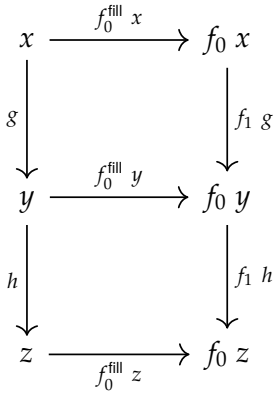
Proof. The proof is standard in category theory. We start with the definition of f_0. For any object $x : C_0\ 0$, the Grothendieck opfibration guarantees the existence of a morphism over $\star : I\ (0 \triangleright 1)$ that we denote $f_0^{\mathsf{fill}}\ x$ whose source is x and whose target over 1 is the image of x under f_0.

Next, for any morphism $g : C_1\ (x \triangleright y)\ 1_0$ in the fibre over 0, we can compose it with $f_0^{\mathsf{fill}}\ y : C_1\ (y \triangleright f_0\ y)\ \star$ and, given that $f_0^{\mathsf{fill}}\ x$ is a cocartesian morphism, there exists a morphism of type $C_1\ (f_0\ x \triangleright f_0\ y)\ 1_1$ which, composed with $f_0^{\mathsf{fill}}\ x$ is equal to $f_0^{\mathsf{fill}}\ y \circ g$. We associate this morphism with the image of g under f_1. This determines the function f_1.



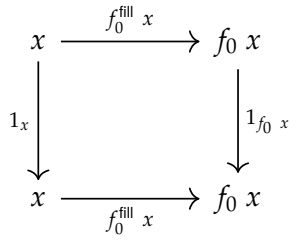
It remains to check the laws. Let $g : C_1\ (x \triangleright y)\ 1_0$ and $h : C_1\ (y \triangleright z)\ 1_0$ be morphisms in the fibre over 0. We want to establish the identity $f_1\ (h \circ g) = f_1\ h \circ f_1\ g$.

We start by noticing that $f_1\ (h \circ g)$ is obtained as the lift of the morphism $f_0^{\mathsf{fill}}\ z \circ h \circ g$ along the cocartesian morphism $f_0^{\mathsf{fill}}\ x$. We have to show that $f_1\ h \circ f_1\ g$ is another lifting candidate then conclude by uniqueness of the lift. By definition of $f_1\ g$, we have a first identity $f_1\ g \circ f_0^{\mathsf{fill}}\ x = f_0^{\mathsf{fill}}\ y \circ g$ then by definition of $f_1\ h$, we have a second identity $f_1\ h \circ f_0^{\mathsf{fill}}\ y = f_0^{\mathsf{fill}}\ z \circ h$. This allows to conclude that $f_1\ h \circ f_1\ g \circ f_0^{\mathsf{fill}}\ x$ is equal to $f_0^{\mathsf{fill}}\ z \circ h \circ g$ and therefore that $f_1\ (h \circ g) = f_1\ h \circ f_1\ g$ by uniqueness of the lift.



Regarding the mapping of identities, for any object $x : C_0\ 0$ in the fibre over 0, the morphism $1_{f_0\ x}$ is defined as the lift of $f_0^{\mathsf{fill}}\ x \circ 1_x$, that is $f_0^{\mathsf{fill}}\ x$, along $f_0^{\mathsf{fill}}\ x$. But $1_{f_0\ x}$ is an equally good lifting candidate therefore we conclude by uniqueness

of the lift.



□

A Grothendieck fibration induces the same data but in the other direction.

Lemma 3.9.8. *Given a Grothendieck fibration over the interval $C : \infty\text{-}\mathsf{category}\downarrow_I$, we can define two functions:*

$$f_0 : C_0\ 1 \to C_0\ 0$$
$$f_1 : \{x\ y : C_0\ 1\} \to C_1\ (x \triangleleft y)\ 1_1 \to C_1\ (f_0\ x \triangleleft f_0\ y)\ 1_0$$

satisfying the following laws:

$$f_1\ (g \circ f) = f_1\ g \circ f_1\ f$$
$$f_1\ 1_x = 1_{f_0\ x}$$

Proof. The proof is similar to the one of about Grothendieck opfibrations but proceeds in the other direction. □

We now check that the pair of functors thus defined forms an adjunction.

Lemma 3.9.9. *Let C be an adjunction. For any pair of objects $x, y : C_0\ 0$ in the fibre over 0, we have the following equivalence:*

$$C_1\ (f_0\ x \triangleright y)\ 1_1 \simeq C_1\ (x \triangleright g_0\ y)\ 1_0$$

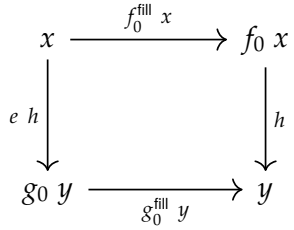
where f_0 (resp. g_0) is the action on objects due to the structure of Grothendieck opfibration (resp. fibration).

Proof. We start with the definition of the forward direction of the equivalence and define the function

$$e : C_1\ (f_0\ x \triangleright y)\ 1_1 \to C_1\ (x \triangleright g_0\ y)\ 1_0$$

Given a morphism $h : C_1\ (f_0\ x \triangleright y)\ 1_1$, we compose it with $f_0^{\mathsf{fill}}\ x$ then we use the cartesian nature of $g_0^{\mathsf{fill}}\ y$ to obtain a morphism of type $C_1\ (x \triangleright g_0\ y)\ 1_0$ that

we assign to the image of h under e. This is the unique morphism satisfying $g_0^{\mathsf{fill}}\ y \circ e\ h = h \circ f_0^{\mathsf{fill}}\ x$.



Conversely, we define the reciprocate function

$$e^{-1} : C_1\ (x \triangleright g_0\ y)\ 1_0 \to C_1\ (f_0\ x \triangleright y)\ 1_1$$

Given a morphism $h : C_1\ (x \triangleright g_0\ y)\ 1_0$, we compose it with $g_0^{\mathsf{fill}}\ y$ then we use the cocartesian nature of $f_0^{\mathsf{fill}}\ x$ to obtain a morphism of type $C_1\ (f_0\ x \triangleright y)\ 1_1$ that we assigns to the image of h under e^{-1}. This is the unique morphism satisfying $e^{-1}\ h \circ f_0^{\mathsf{fill}}\ x = g_0^{\mathsf{fill}}\ y \circ h$.

We now show that e is a section. Let $h : C_1\ (f_0\ x \triangleright y)\ 1_1$ be a morphism. $e^{-1}\ (e\ h)$ is the unique morphism such that $e^{-1}\ (e\ h) \circ f_0^{\mathsf{fill}}\ x = g_0^{\mathsf{fill}}\ y \circ e\ h$. But $e\ h$ is the unique morphism such that $g_0^{\mathsf{fill}}\ y \circ e\ h = h \circ f_0^{\mathsf{fill}}\ x$. From these two identities we deduce that $e^{-1}\ (e\ h) \circ f_0^{\mathsf{fill}}\ x = h \circ f_0^{\mathsf{fill}}\ x$ and therefore that $e^{-1}\ (e\ h) = h$ using the fact that $f_0^{\mathsf{fill}}\ x$ is cocartesian.

The fact that e is a retraction follows from the same argument therefore we do not detail the proof. □

Conclusion

We have explored an extension of type theory in which higher algebraic structures on types are definable. We managed to use this type theory to define an important range of algebraic structures including ∞-groupoids and $(\infty, 1)$-categories as well as to prove elementary facts about them. To this end, we chose to use opetopes: geometrical shapes which capture the combinatorics of the coherence data that we wish to describe.

We therefore opened this thesis by first giving a purely type-theoretical definition of opetopes in a type theory similar to book HoTT in Chapter 1. Our definition essentially defines opetopes as sequences of well-founded trees satisfying some properties which are nicely captured by their typing. The opetopic approach particularly shines in the context of type theory as well-founded trees fall within the realm of inductive types. More specifically, our construction is based on a sequence of polynomial monads, a notion which becomes central in later chapters. We concluded this chapter with an inductive definition of the faces of an opetope. This self-contained chapter is the opportunity for the reader to get familiar with opetopes before broaching on opetopic types whose complexity may hide the conceptual simplicity of opetopes.

Equipped with the understanding of opetopes, a simple example of polynomial monads, we extended type theory with a universe of cartesian polynomial monads $\mathcal{M}$ closed under some monad constructors in Chapter 2. We did so in the aim of defining opetopic types, collection of types whose combinatorics is described by opetopes, in Chapter 3. Contrary to our definition of opetopes which only involves sets, opetopic types are valued in arbitrary types. As a consequence, we can no longer state the equational laws of Chapter 1 in a coherent fashion, having no means to do so! We therefore defined our universe of polynomial monads in order that these equational laws hold definitionally. The constructors under which our universe is closed then allow us to define, in particular, the sequence of polynomial monads defining opetopes. In addition to this universe which constitutes the core of our addition to type theory, we added two sorts of universes as further extensions. First, we defined a family of universes $M \to_m N$ of cartesian monad morphisms from M to N. Then, we defined a family of universes $\mathcal{M}\downarrow_M$ of polynomial monad families over on a monad M. These two extensions were developed in order to establish some more advanced results in Chapter 3.

Finally, we took advantage of our universe of polynomial monads to define opetopic types in Chapter 3. This enabled us to define coherent higher algebraic

structures, among which ∞-groupoids and $(\infty, 1)$-categories. Crucially, their higher structure is encoded in terms of identity types. We then established expected results in order to motivate our definitions. We first compared our definition of fibrant opetopic type with Baez and Dolan definition of "coherent O-algebras" and shown that they were equivalent if we require all morphisms to be invertible. Then, we established that set-truncated fibrant Id-opetopic types are equivalent to precategories as defined in the HoTT book. We quickly reached for the dependent monads as alluded earlier in order to establish the remaining results. We started by defining fibrations of opetopic types which are a generalisation of fibrant opetopic types then we showed that fibrant opetopic types are closed under dependent sums. We also defined the universal fibration of types as an opetopic type. Finally, we defined Grothendieck (op)fibrations that we applied in order to define Lurie's definition of an adjunction (LURIE 2009, Definition 5.2.2.1) as a bifibration over the interval.

As future work, we aim to first improve our system by extending type theory with a primitive notion of opetopic types instead of relying on a universe of polynomial monads. This more synthetic framework would prevent some difficulties that we had to face with the current one. In particular, we could dispense with the universes of monad morphisms which have been introduced for purely technical reasons while none of our results crucially rely on it. Concerning the results that we wish to establish, we would like to first fully formalise the equivalence between types and ∞-groupoids that we presented at LICS2021 (FINSTER, ALLIOUX and SOZEAU 2021). Next, a work in progress consists in defining the fibrant opetopic types of the faces of an opetope. These opetopic types would correspond to the representable functors on the category of opetopes. A more ambitious project consists in the development of higher category theory in univalent opetopic foundations. A lot of basic category theory results remain to be established such as the slice construction of an opetopic type which we have failed to define so far. In that respect, there is no reason to think that approaches similar to ours but based on other geometries could not be fruitful. In particular, a presentation of types based on simplicial sets might be useful in order to get closer to the current higher category literature. However, we do not expect simplicial techniques to directly transfer to a homotopy type theory in which all types are homotopy types.

Apart from the wealth of results that remain to be established in our system, we identify two other themes for future work. The first one is the development of a dedicated type checker for our type theory or even a dedicated Agda mode. Our present implementation has been carried out in Agda and its system of rewrite rules has been invaluable to quickly implement and experiment with our framework. However, we had to get around some of its shortcomings more than once, and we feel that we have hit some of its limits. The second theme is the semantic justification of our system on which we stayed silent in this thesis. Once we reach an optimal presentation of our framework, we should investigate its precise semantics.

More broadly, regarding the general challenge of defining higher algebraic structures on types, the opetopic approach that we propose is the first which

departs from the line of work based on two-level type theory (Altenkirch, Capriotti and Kraus 2016; Annenkov et al. 2017; Capriotti and Kraus 2017; Kraus and Sattler 2017) which originates with Voevodsky's Homotopy Type System (Voevodsky 2013). Instead of reintroducing a strict equality, incompatible with the homotopy interpretation of types, we introduce a means of presentation for higher dimensional structures based on opetopes. This approach is still in its infancy, and only a better theoretical understanding of our system will allow us to grasp the full extent of its capabilities and how practical they are. Nonetheless, it paves the way for future approaches.

Bibliography

Abbott, Michael, Thorsten Altenkirch and Neil Ghani (2005). "Containers: Constructing strictly positive types". In: *Theoretical Computer Science* 342.1, pp. 3–27 (cit. on pp. 6, 53).

Abel, Andreas, Jesper Cockx et al. (2020). "Leibniz equality is isomorphic to Martin-Löf identity, parametrically". In: *Journal of Functional Programming* 30 (cit. on p. 134).

Abel, Andreas, Brigitte Pientka et al. (2013). "Copatterns: Programming infinite structures by observations". In: *ACM SIGPLAN Notices* 48.1, pp. 27–38 (cit. on p. 138).

Agda development team (2022). *Agda 2.6.2.2 documentation*. url: `https://agda.readthedocs.io/en/v2.6.2.2/` (cit. on p. 9).

Altenkirch, Thorsten, Paolo Capriotti and Nicolai Kraus (2016). "Extending homotopy type theory with strict equality". In: *arXiv preprint arXiv:1604.03799* (cit. on pp. 2, 4, 10, 121).

Altenkirch, Thorsten, Neil Ghani et al. (2015). "Indexed containers". In: *Journal of Functional Programming* 25 (cit. on pp. 6, 53).

Angiuli, Carlo et al. (2021). "Syntax and models of Cartesian cubical type theory". In: *Mathematical Structures in Computer Science* 31.4, pp. 424–468. doi: `10.1017/S0960129521000347` (cit. on p. 136).

Annenkov, Danil et al. (2017). "Two-level type theory and applications". In: *arXiv preprint arXiv:1705.03307* (cit. on pp. 2, 10, 121).

Awodey, Steve (2014). "Structuralism, invariance, and univalence". In: *Philosophia Mathematica* 22.1, pp. 1–11 (cit. on p. 2).

Awodey, Steve and Michael A Warren (2009). "Homotopy theoretic models of identity types". In: *Mathematical proceedings of the cambridge philosophical society*. Vol. 146. 1. Cambridge University Press, pp. 45–55 (cit. on p. 135).

Baez, John C and James Dolan (1998). "Higher-dimensional algebra III. n-categories and the algebra of opetopes". In: *Advances in Mathematics* 135.2, pp. 145–206 (cit. on pp. 2, 4, 6, 11, 53, 58, 73, 85, 147, 150).

Belnap, Nuel D. (1962). "Tonk, Plonk and Plink". In: *Analysis* 22.6, pp. 130–134. issn: 00032638, 14678284. url: `http://www.jstor.org/stable/3326862` (visited on 23/01/2023) (cit. on p. 132).

Bezem, Marc, Thierry Coquand and Simon Huber (2014). "A Model of Type Theory in Cubical Sets". In: *19th International Conference on Types for Proofs and Programs (TYPES 2013)*. Ed. by Ralph Matthes and Aleksy Schubert. Vol. 26. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl,

Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 107–128. ISBN: 978-3-939897-72-9. DOI: 10.4230/LIPIcs.TYPES.2013.107. URL: http://drops.dagstuhl.de/opus/volltexte/2014/4628 (cit. on p. 136).

BROUWER, L. E. J. (1949). "Consciousness, Philosophy, and Mathematics". In: *Proceedings of the Tenth International Congress of Philosophy* 2, pp. 1235–1249 (cit. on p. 127).

BROUWER, Luitzen Egbertus Jan (1907). *Over de grondslagen der wiskunde*. Maas & van Suchtelen (cit. on p. 127).

CAPRIOTTI, Paolo and Nicolai KRAUS (2017). "Univalent higher categories via complete semi-segal types". In: *Proceedings of the ACM on Programming Languages* 2.POPL, pp. 1–29 (cit. on pp. 2, 10, 98, 121).

CHENG, Eugenia (2004a). "Weak n-categories: comparing opetopic foundations". In: *Journal of Pure and Applied Algebra* 186.3, pp. 219–231 (cit. on p. 11).

— (2004b). "Weak n-categories: opetopic and multitopic foundations". In: *Journal of Pure and Applied Algebra* 186.2, pp. 109–137 (cit. on p. 11).

COCKX, Jesper (2020). "Type theory unchained: Extending Agda with user-defined rewrite rules". In: *25th International Conference on Types for Proofs and Programs (TYPES 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (cit. on p. 9).

COHEN, Cyril et al. (2016). "Cubical type theory: a constructive interpretation of the univalence axiom". In: *arXiv preprint arXiv:1611.02108* (cit. on p. 136).

DUMMETT, Michael (1991). *The logical basis of metaphysics*. Harvard university press (cit. on p. 132).

FINSTER, Eric, Antoine ALLIOUX and Matthieu SOZEAU (2021). "Types are internal ∞-groupoids". In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, pp. 1–13 (cit. on pp. 112, 120).

GAMBINO, Nicola and Richard GARNER (2008). "The identity type weak factorisation system". In: *Theoretical computer science* 409.1, pp. 94–109 (cit. on p. 135).

GAMBINO, Nicola and Joachim KOCK (2013). "Polynomial functors and polynomial monads". In: *Mathematical proceedings of the cambridge philosophical society*. Vol. 154. 1. Cambridge University Press, pp. 153–192 (cit. on p. 54).

GISIN, Nicolas (2020). "Mathematical languages shape our understanding of time in physics". In: *Nature Physics* 16.2, pp. 114–116 (cit. on p. 129).

HERMIDA, Claudio, Michael MAKKAI and John POWER (2000). "On weak higher dimensional categories I: Part 1". In: *Journal of pure and applied algebra* 154.1-3, pp. 221–246 (cit. on p. 11).

HEYTING, Arend (1930). "Die formalen Regeln der intuitionistischen Logik". In: *Sitzungsbericht PreuBische Akademie der Wissenschaften Berlin, physikalisch-mathematische Klasse II*, pp. 42–56 (cit. on p. 128).

HOFMANN, Martin and Thomas STREICHER (1998). "The groupoid interpretation of type theory". In: *Twenty-five years of constructive type theory (Venice, 1995)* 36, pp. 83–111 (cit. on p. 134).

HOWARD, William A (1980). "The formulae-as-types notion of construction". In: *To HB Curry: essays on combinatory logic, lambda calculus and formalism* 44, pp. 479–490 (cit. on p. 133).

Kapulkin, Krzysztof and Peter LeFanu Lumsdaine (2021). "The simplicial model of Univalent Foundations (after Voevodsky)". In: *Journal of the European Mathematical Society* 23.6, pp. 2071–2126 (cit. on p. 136).

Kock, Joachim et al. (2010). "Polynomial functors and opetopes". In: *Advances in Mathematics* 224.6, pp. 2690–2737 (cit. on pp. 11, 12).

Kraus, Nicolai and Christian Sattler (2017). "Space-valued diagrams, type-theoretically". In: *arXiv preprint arXiv:1704.04543* (cit. on pp. 2, 10, 121).

Lambek, Joachim and Philip J Scott (1988). *Introduction to higher-order categorical logic*. Vol. 7. Cambridge University Press (cit. on p. 133).

Leinster, Tom (2001). "Structures in higher-dimensional category theory". In: *arXiv preprint math/0109021* (cit. on p. 11).

— (2004). *Higher operads, higher categories*. 298. Cambridge University Press (cit. on pp. 6, 53, 147).

Lumsdaine, Peter LeFanu (2010). "Weak omega-categories from intensional type theory". In: *Logical Methods in Computer Science* 6 (cit. on pp. 1, 111, 135).

Lurie, Jacob (2009). *Higher topos theory*. Princeton University Press (cit. on pp. 9, 112, 114, 120).

Mac Lane, Saunders (2013). *Categories for the working mathematician*. Vol. 5. Springer Science & Business Media (cit. on p. 76).

Makkai, Michael (1995). "First order logic with dependent sorts, with applications to category theory". In: *Preprint: http://www. math. mcgill. ca/makkai* (cit. on p. 136).

Martin-Löf, Per (1975). "An intuitionistic theory of types: Predicative part". In: *Studies in Logic and the Foundations of Mathematics*. Vol. 80. Elsevier, pp. 73–118 (cit. on p. 129).

— (1994). "Analytic and synthetic judgements in type theory". In: *Kant and contemporary epistemology*. Springer, pp. 87–99 (cit. on p. 133).

— (1996). "On the meanings of the logical constants and the justifications of the logical laws". In: *Nordic journal of philosophical logic* 1.1, pp. 11–60 (cit. on p. 130).

Martin-Löf, Per and Giovanni Sambin (1984). *Intuitionistic type theory*. Vol. 9. Bibliopolis Naples (cit. on p. 136).

McBride, Conor (2002). "Elimination with a motive". In: *Types for Proofs and Programs: International Workshop, TYPES 2000 Durham, UK, December 8–12, 2000 Selected Papers*. Springer, pp. 197–216 (cit. on p. 131).

Paulin-Mohring, Christine (1993). "Inductive definitions in the system Coq rules and properties". In: *Typed Lambda Calculi and Applications: International Conference on Typed Lambda Calculi and Applications TLCA'93 March, 16–18, 1993, Utrech, The Netherlands Proceedings 1*. Springer, pp. 328–345 (cit. on p. 134).

Riehl, Emily and Michael Shulman (2017). "A type theory for synthetic ∞-categories". In: *arXiv preprint arXiv:1705.07442* (cit. on p. 76).

Shulman, Michael (2019). "All $(\infty, 1)$-toposes have strict univalent universes". In: *arXiv preprint arXiv:1904.07004* (cit. on p. 1).

Sørensen, Morten Heine and Pawel Urzyczyn (2006). *Lectures on the Curry-Howard isomorphism*. Elsevier (cit. on p. 133).

UNIVALENT FOUNDATIONS PROGRAM, The (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: `https://homotopytypetheory.org/book` (cit. on pp. 1, 127, 136, 141).

VAN DEN BERG, Benno and Richard GARNER (2011). "Types are weak ω-groupoids". In: *Proceedings of the london mathematical society* 102.2, pp. 370–394 (cit. on pp. 1, 111, 135).

VOEVODSKY, Vladimir (2006). "A very short note on the homotopy λ-calculus". In: *Unpublished note*, pp. 10–27 (cit. on p. 135).

— (2013). "A simple type system with two identity types". In: *Unpublished note* (cit. on pp. 4, 121).

WADLER, Philip (2015). "Propositions as types". In: *Communications of the ACM* 58.12, pp. 75–84 (cit. on p. 132).

WEYL, Herman (1921). "On the New Foundational Crisis of Mathematics". In: (cit. on p. 128).

Appendix A

Background material

We recall homotopy type theory and its history then we recall elementary results from the HoTT book (Univalent Foundations Program 2013) that will be used throughout this thesis in Section A.5. Homotopy type theory is an extension of Intuitionistic intensional type theory (ITT), a formal language which can serve as a foundation for constructive mathematics. It internalises certain principles which are satisfied by homotopical models of ITT.

A.1 Constructive mathematics

Broadly speaking, constructive mathematics qualifies the mathematics done in such a way that asserting the existence of a mathematical object consists in exhibiting a particular instance of this object. This implies that certain modes of reasoning typical of classical logic such as the double negation elimination are prohibited. There are a number of reasons why one would like to avoid those principles ranging from philosophical reasons to more practical ones.

There are many schools of constructivism with diverging motivations. Intuitionism, on which is based intuitionistic logic, the logic implemented by ITT, originates with the work of Brouwer (Luitzen Egbertus Jan Brouwer 1907). Brouwer advocates a mathematics conceived as a construction of the mind appealing to our intuitions: "There are no non-experienced truths"(L. E. J. Brouwer 1949). In this philosophy of mathematics, one cannot appeal to the properties of a Platonic realm of mathematical truths where any proposition is either true or false. This belief, not motivated by any intrinsic mathematical reason, and legitimised in some formal systems by logical axioms such as the law of excluded middle (LEM) made Brouwer suspicious of both Platonists and Formalists. He particularly denied the "creative role" of logic as he held the belief that logic had to be submitted to mathematics and not the other way around.

In Intuitionism, in order to assert the existence of a mathematical object, one has to provide a concrete construction of that object. Mathematics exist only to the extent that we can write them therefore we are constrained by the language we use and our means of calculation. This is to be compared with an "ideal mathematician" with unlimited memory and time. In this respect, the LEM is

forbidden as it allows concluding the truth of a proposition without explicitly proving it by virtue of the fact that, in the Platonic realm of mathematical truths, any proposition is either true or false. The axiom of choice (AC) is also commonly rejected as it entails the LEM by Diaconescu's theorem. Though, depending on its formulation, some forms of it can be accepted. Rejecting the LEM implies rejecting the double negation elimination as well. This means that we cannot prove a proposition P by assuming $\neg P$ and deriving a contradiction. However, we can still prove $\neg P$ by assuming P and deriving a contradiction.

Hermann Weyl, who temporarily adopted the intuitionistic views of Brouwer, held that if knowledge is a treasure then an existential statement in classical logic asserts the presence of a treasure without disclosing its location (WEYL 1921). A simple and illuminating example is the proof that there exist irrational numbers x and y such that x^y is rational. The classical proof goes as follows: either $\sqrt{2}^{\sqrt{2}}$ is a rational number therefore we take $x = y = \sqrt{2}$, or $\sqrt{2}^{\sqrt{2}}$ is not a rational number and we take $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$ as, in this case, $x^y = 2$ which is a rational number. Such a proof rests upon the use of the LEM as it uses the fact that $\sqrt{2}^{\sqrt{2}}$ is either rational or irrational. Constructivists object that this proof is not satisfying as it does not inform us about which one of the two pairs (x, y) is the one having the required property. A constructive proof of the same statement is the following: consider the two irrational numbers $x = \sqrt{2}$ and $y = 2 \log_2 3$, the number $x^y = 3$ is indeed a rational number. Such a proof gives concrete evidence of its truth by specifying particular irrational numbers x and y satisfying the proposition. Another gripe with the LEM is that it allows to assert the truth of propositions dealing with infinite collections that we might not be able to check anyway. Consider the statement "There are seven 7's in a row in the decimal representation of π". The LEM allows concluding that this is either true or false. Constructively, we would have to check each decimal until we find an instance of these seven 7's in order to conclude that this statement is provable. However, we could not refute it as we cannot check the infinity of digits of π in a finite amount of time.

There are other grounds for rejecting classical principles. In particular, statements which do not make use of classical reasoning principles are valid in more models. For example, a statement in intuitionistic logic is true in any topos, not just boolean ones which are home to classical logic. Another reason to be constructive is if one wants to extract a program from a proof. This is particularly useful when one wants to certify a piece of software. In this thesis, we reject unrestricted classical principles on the ground that they are, in general, incompatible with univalence which is central to homotopy type theory. However, HoTT identifies a class of types for which these principles apply. In that sense, we regard HoTT as subsuming set theory and classical logic.

Despite Brouwer's hostility towards formalism, one of his students, Arend Heyting, devised intuitionistic logic (HEYTING 1930) in order to give a formal foundation to Brouwer's mathematics. This logic is characterised by two key properties: the existence and the disjunction properties. The existence property

asserts that a proposition of the form $\exists x.B(x)$ is true if and only if there is a particular element a such that $B(a)$ is true. As for the disjunction property, it asserts that a proposition of the form $A \vee B$ is true if and only if we can prove that either A or B is true. The strong interpretation of the existential quantifier in intuitionistic logic reflects a shift in the interpretation of the truth of a proposition. In this logic, truth is identified with provability. It is then natural to see the notion of time emerging from this interpretation. Being able to prove $A \vee \neg A$ for example is being able to prove A or $\neg A$ right now which might not be possible. Consider an open conjecture such as the Riemann hypothesis for example. We cannot prove it nor refute it at the time of writing. This notion of time is implicit in Kripke models for intuitionistic logic where a model consists of a collection of consistent states of knowledge assembled into a poset modelling the accumulation of knowledge consistent with previously established facts. Today, some physicists study whether intuitionistic logic could be better suited to the study of the physical world than classical logic (Gisin 2020).

Having rejected the interpretation of logical judgements in terms of truth values, an explanation of their meaning is given by the so-called Brouwer-Heyting-Kolmogorov interpretation of intuitionistic logic. In this interpretation, logical connectors and quantifiers are explained in terms of their collections of proofs:

A proof of	consists of
$A \wedge B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \implies B$	a method which takes a proof of A and returns a proof of B
$\exists x.B(x)$	an element a along with a proof of $B(a)$
$\forall x.B(x)$	a method which, for any element x, produces a proof of $B(x)$
$\neg A$	a proof of $A \implies \bot$
$\bot$	nothing

This interpretation does not specify what a proof is nor what a method is, it is left to the ambient logical system to specify these notions. We can regard intuitionistic type theory as giving a formal treatment to the BHK interpretation.

A.2 Intuitionistic type theory

Intuitionistic intensional type theory is a formal language due to Per Martin-Löf (Martin-Löf 1975) which can serve as a foundation for constructive mathematics. It is a formal system of abstract constructions classified by *types*. Types are defined by specifying their canonical elements as well as when two terms of a type are equal *by definition*. This leads us to consider the following two forms of judgements:

Judgement	Meaning
$x : A$	x is a term of type A
$x \equiv y : A$	x and y are definitionally equal terms of type A

Judgements are the facts that can be established in our type theory. A judgement is an "object of knowledge" (MARTIN-LÖF 1996). They are assertions about the meta-language and cannot be denied internally. Compare a typing judgement with a set-theoretical membership proposition: one is external while the other is internal. In set-theory it is possible to ask whether the proposition $0 \in 1$ is true while there exists no corresponding typing judgement in type theory where such a question is just nonsensical.

New judgements may be derived from prior ones using the inference rules of type theory. For example, proving a mathematical proposition using the propositions-as-types paradigm corresponds to deriving a judgement of the form $t : A$ where A is the type corresponding to the proposition that we want to prove and t is a term witnessing the truth of this proposition.

Defining a new type consists in adding new inference rules to our type theory. Contrast this approach with the set-theoretical one where inference rules are fixed — they are most often those of first-order logic — and where new sets are built from existing ones using a fixed set of axioms — the Zermelo-Fraenkel axioms for example. Set theory lacks the distinction between judgements — which are external statements — and internal statements.

Our type theory has a universe à la Russell; that is, a family of types $(\mathcal{U}_n)_{n:\mathbb{N}}$ such that $\mathcal{U}_n : \mathcal{U}_{n+1}$. Any type inhabits a particular universe and a universe inhabits a greater universe in order to prevent paradoxes such as Russell's paradox. In order not to clutter the notation, we will omit the universe levels in this informal presentation of type theory.

ITT can be seen as a dependently typed lambda calculus as soon as we introduce function types which allow internalising hypothetical judgements. Let $A, B : \mathcal{U}$ be two types, we define their function type $A \to B : \mathcal{U}$. Considering a term $t : B$ under the assumption that a variable $x : A$ may appear in t, there is a function $(\lambda x \to t) : A \to B$ where x is now no longer an assumption — it is bound. Given a function $f : A \to B$ and a term $t : A$, f can be applied to t to yield the term $f(t) : B$. The application has to satisfy the β-reduction rule $(\lambda x \to u)(t) \equiv u[t/x]$ where $u[t/x]$ is the capture-avoiding substitution of the occurrences of x in u by the term t.

Along with the universe $\mathcal{U}$, function types permit the definition of type families — also called *dependent* types — which are functions of type $A \to \mathcal{U}$, where A is a type. This allows the generalisation of function types to dependent function types. Let $A : \mathcal{U}$ be a type and let $B : A \to \mathcal{U}$ be a type family. We form the type $(x : A) \to B(x) : \mathcal{U}$. Dependent functions are then introduced as sections of B; given a term $t : B(x)$ for any $x : A$ where the variable x may appear in t, we introduce the function $(\lambda x \to t) : (x : A) \to B(x)$. Given a function $f : (x : A) \to B(x)$ and a term $t : A$, f can be applied to t to yield the term $f(t) : B(t)$. The application also has to satisfy the β-reduction rule $(\lambda x \to u)(t) \equiv u[t/x]$. If B is the constant family $\lambda x \to B$ then $(x : A) \to B(x)$ coincides with the ordinary function type $A \to B$.

Defining a new type, in the predicative version of type theory, consists in introducing new inference rules following a particular pattern that we now illustrate with the definition of the type of natural numbers $\mathbb{N}$.

1. **Formation rule.** The formation rule introduces a new type by asserting that $\mathbb{N}$ is an element of the universe of types.

$$\mathbb{N} : \mathcal{U}$$

2. **Introduction rules.** The introduction rules introduce the canonical elements of $\mathbb{N}$. In the case of the unary definition of natural numbers, we distinguish two cases:

   - There is an element $0 : \mathbb{N}$ standing for the natural number 0.
   - For any natural number $n : \mathbb{N}$, there is an element $\mathsf{suc}(n) : \mathbb{N}$ standing for the successor of n.

3. **Elimination rules.** The elimination rule states the consequences that can be derived from an arbitrary element of $\mathbb{N}$. It prescribes how to define any function $f : \mathbb{N} \to X$ for an arbitrary type X that we sometimes call the *motive* of the elimination (McBRIDE 2002). In our case, this corresponds to functions defined by primitive recursion.

   For any type $X : \mathcal{U}$, a function defined by primitive recursion requires the data of a base case $x_0 : X$ and a function $x_{\mathsf{suc}} : \mathbb{N} \to X \to X$. Then, for any natural number $n : \mathbb{N}$, there is an element

$$\mathbb{N}_{\mathsf{elim}}(X, x_0, x_{\mathsf{suc}}, n) : X$$

4. **Computation rules.** The computation rules state how the elimination rules behave with regard to the introduction rules. In the present case, we have to treat the two different introduction rules which correspond to the definition of a primitive recursive function:

$$\begin{aligned} \mathbb{N}_{\mathsf{elim}}(X, x_0, x_{\mathsf{suc}}, 0) &\equiv x_0 \\ \mathbb{N}_{\mathsf{elim}}(X, x_0, x_{\mathsf{suc}}, \mathsf{suc}(n)) &\equiv x_{\mathsf{suc}}(n, \mathbb{N}_{\mathsf{elim}}(X, x_0, x_{\mathsf{suc}}, n)) \end{aligned}$$

In addition to these four sorts of rules, there might be additional rules specific to the type in question. It is for example common to specify uniqueness rules, often named η-rules, stating how should the introduction rules behave with regard to the elimination rule. For example, pairing the projections of a pair p should be a pair definitionally equal to p: $(\mathsf{pr}_1(p), \mathsf{pr}_2(p)) \equiv p$. Similarly, any function f should satisfy the equality $(\lambda x \to f(x)) \equiv f$.

The definition of natural numbers that we just gave is *synthetic* or *axiomatic* as we directly state what they are without defining them out of more primitive constructions as it is customary in set theory. This definition is an example of a certain class of types named *inductive types* which are freely generated by their constructors.

Of course, the elimination rule cannot be arbitrary or else we run the risk of being inconsistent. It has to follow from the introduction rules, which was

pointed out by Belnap (BELNAP 1962) and named *logical harmony* by Dummett (DUMMETT 1991). To a first approximation, when introduction and elimination rules are in harmony, we can infer *no more* and *no less* than what has been established by the introduction rules. In the particular case of inductive types, a more algebraic point of view on this matter is to consider them as initial algebras for some endofunctor. The elimination rule then defines the unique homomorphism out of this algebra. The picture becomes more subtle when considering the particular case of identity types whose elimination rule does not allow deducing that its sole elements are the ones introduced by the introduction rules. This is a fundamental observation which allowed to postulate new principles such as the univalence axiom and higher inductive types which constitute the novelties of homotopy type theory.

Often, we will want to use a more general elimination rule where the motive is a type dependent on the type being defined, that is $X : \mathbb{N} \to \mathcal{U}$ in the case of natural numbers. The data to provide now consists of $x_0 : X(0)$ and $x_{\mathsf{suc}} : (n : \mathbb{N}) \to X(n) \to X(\mathsf{suc}(n))$. The dependent elimination, also called the *induction* principle, then provides the following element $\mathbb{N}_{\mathsf{ind}}(X, x_0, x_{\mathsf{suc}}, n) : X(n)$ subject to the same computation rules as the ordinary elimination principle. While we can see the regular elimination principle as defining functions by recursion over the type being defined, the induction principle can be regarded as defining proofs by induction over this same type.

A.3 Propositions as types

In addition to being able to define mathematical constructions, we need to be able to formulate propositions about those constructions and be able to prove them. If we subscribe to the Brouwer-Heyting-Kolmogorov interpretation of intuitionistic logic, the meaning of a proposition is determined by its collection of proofs. Framing this interpretation in type-theoretical terms, we define propositions as types whose introduction and elimination rules reflect their counterpart in intuitionistic logic. This is known as the *propositions-as-types* paradigm (WADLER 2015) which establishes the following connection between logic and type theory:

Logic	Type theory
$\bot$	$\mathbf{0}$
$A \wedge B$	$A \times B$
$A \vee B$	$A + B$
$A \implies B$	$A \to B$
$\exists(x \in A).B(x)$	$\sum_{(x:A)} B(x)$
$\forall(x \in A).B(x)$	$(x : A) \to B(x)$

Type theory is therefore a language rich enough to accommodate and unify seemingly disparate concepts such as mathematical constructions and logical propositions. Typing judgements can thus have the following readings depend-

ing on how we interpret the types in question.

$A : \mathcal{U}$	$x : A$
A is a type	x is an element of type A
A is a proposition	x is a proof of the proposition A
A is a specification	x is a program meeting the specification A

The second interpretation is the point of view of the propositions-as-types paradigm. The third one corresponds to the interpretation of ITT as a programming language.

The richness of types even opens up the possibility of defining types which have no direct counterpart either as a set-based mathematical construction or as a logical proposition. This observation is at the root of homotopy type theory whose motivation is to classify and study types according to their intrinsic geometrical content.

The connection between types and propositions had already been noticed. There is a precise correspondence between types of the simply typed lambda calculus (STLC) and intuitionistic propositional logic (IPC) known as the Curry-Howard correspondence (Howard 1980). Under this view, the typing rules of STLC can be seen as an annotated version of the rules of natural deduction for IPC. Lambda terms can then be seen as certificates for the proof derivations they denote. The correspondence includes a correspondence between the reduction rules of the lambda calculus and proof normalisation of natural deduction. De Bruijn himself arrived independently at a similar correspondence. Lambek further extended this connection to category theory and showed that the STLC was the internal language of cartesian closed categories (Lambek and Scott 1988). The Curry-Howard correspondence has since been extended to more expressive lambda calculi (Sørensen and Urzyczyn 2006).

We finish this section by considering the decidability of judgements. Typing judgements of the form $x : A$ are decidable. The rules of ITT are such that x acts as a certificate from which we can establish the derivation leading to the judgement in question. It then suffices to check that this derivation is correct. The same applies to a judgement of the form $x \equiv y : A$, it suffices to compare the normal forms of x and y — up to certain rules — in order to conclude. In contrast, a judgement like A true that can be derived only if there exists some term t such that the judgement $t : A$ holds, is undecidable. It is missing a piece of information, namely the data of t, in order to decide whether it holds. However, there does not exist any algorithm able to establish whether a type is inhabited or not as this problem is undecidable. This distinction is important for proof assistants which are programs that help mathematicians check that their proofs are correct. Because of this limitation, mathematicians are still expected to produce a proof in the first place, although this process can be automated to some extent. This distinction corresponds to the one between *analytic* and *synthetic* judgements which goes back to Kant (Martin-Löf 1994). Analytic judgements distinguish themselves from synthetic judgements by the fact that they carry their own proof.

A.4 Homotopy type theory

A.4.1 The genesis

Homotopy type theory refers to ITT extended with principles consistent with the homotopy interpretation of types. Under this interpretation, types enjoy the rich structure of abstract space conferred by their identity types.

Martin-Löf identity types are the types which correspond to propositional equality under the propositions-as-types paradigm. From their proof-relevance arises interesting phenomena. We present them using the formulation due to Paulin-Mohring (PAULIN-MOHRING 1993).

Let $A : \mathcal{U}$ be a type and let $x, y : A$, we form the type $x =_A y : \mathcal{U}$ of identifications between x and y. We will often drop the type indication A when it can be inferred from the context. Identity types have a single introduction rule $\mathsf{refl}_x : x =_A x$ for any $x : A$ corresponding to the fact that any definitional equality $x \equiv y$ can be turned into a propositional one. We say that $x =_A y$ is *parametrised* by A and x and *indexed* by y in that A and x are fixed before defining the introduction rule (i.e., it is not constrained by it) while y is constrained to be definitionally equal to x by the introduction rule. We now describe the corresponding induction principle. Let $A : \mathcal{U}$ and $x : A$ be the parameters of our type. Let $B : (y : A) \to x =_A y \to \mathcal{U}$ be a type family which will play the role of the motive of elimination. In order to define a term $B(y, p)$ for any identity $p : x =_A y$, it suffices to provide a term $d : B(x, \mathsf{refl}_x)$. The induction principle therefore provides us with the term

$$=_{\mathsf{ind}} (A, x, B, d, y, p) : B(y, p)$$

satisfying the following computation rule when applied to refl_x:

$$=_{\mathsf{ind}} (A, x, B, d, x, \mathsf{refl}_x) \equiv d$$

The assumption d of the induction principle corresponds to the data needed to cover the unique case where p is refl_x.

The induction principle of identity types allows deriving a substitution principle which is a formal version of Leibniz's identity of indiscernibles. Given two elements $x, y : A$ and an identity $p : x = y$ then, for any type family $B : A \to \mathcal{U}$, there is a function $p_* : B(x) \to B(y)$. Conversely, the identity of indiscernibles implies the corresponding identity type. This logical equivalence can be turned into an equivalence of types assuming internal parametricity (ABEL, COCKX et al. 2020).

Interestingly, this induction principle does not allow proving that any identity $p : x =_A x$ is equal to refl_x! This principle, named *uniqueness of identity proofs* (UIP), is summarised as

$$\mathsf{UIP} : (A : \mathcal{U})\ (x : A)\ (p : x = x) \to p = \mathsf{refl}_x$$

Determining whether such a principle was provable in ITT remained an open question for a long time until it was settled by Hofmann and Streicher (HOFMANN

and Streicher 1998) who defined a model of ITT, the groupoid model, where it does not hold. In this model, a type $A : \mathcal{U}$ is interpreted as a groupoid whose objects are the elements of A and whose morphisms between two objects corresponding to two elements $x, y : A$ correspond to identities $x =_A y$. The structure of groupoid stems from the operations that can be defined using the induction principle of identity types. For any type $A : \mathcal{U}$ and triplet $x, y, z : A$ there is a composition operation

$$x = y \to y = z \to x = z$$

We denote $p \cdot q$ the identity resulting from the composition of two identities p and q. There is a second operation inverting identities

$$x = y \to y = x$$

We denote p^{-1} the inverse of the identity p. We can prove that those operations satisfy the following *propositional* laws for any $p : x = y$, $q : y = z$, and $r : z = t$:

$$\begin{aligned} p \cdot \mathsf{refl}_y &= \mathsf{refl}_x \cdot p = p \\ p \cdot p^{-1} &= \mathsf{refl}_x \\ p^{-1} \cdot p &= \mathsf{refl}_y \\ (p \cdot q) \cdot r &= p \cdot (q \cdot r) \end{aligned}$$

Then, considering the group $\mathbb{Z}_2$ viewed as a one-object groupoid, for example, there is an identity $p : * =_{Z_2} *$ different from refl_* which disproves the UIP.

This observation paved the way for a line of work clarifying the semantics of type theory in its *intensional* form. The tools of homotopy theory proved themselves to be adapted to this task with Gambino and Garner (Gambino and Garner 2008) showing that the axioms of identity types induce a weak factorisation system on the syntactic category of type theory and Awodey and Warren (Awodey and Warren 2009) showing that there is a Quillen model structure on this category where identity types play the role of path objects.

Similarly, the tools of higher algebra allowed Lumsdaine (Lumsdaine 2010) and Van Den Berg and Garner (Van Den Berg and Garner 2011) to generalise the Hoffman-Streicher interpretation of types to ∞-groupoids using globular operads.

A.4.2 Univalence

In his note (Voevodsky 2006), the mathematician Vladimir Voevodsky identified a missing principle in type theory. It is commonplace in mathematical practice to use the same notation for different but equivalent mathematical structures without being explicit about it. Informally, we say that we are making an *abuse of language*. Framing this principle in type-theoretical terms and identifying mathematical structures with (families of) types, given two types $A, B : \mathcal{U}$ and an equivalence of types $e : A \simeq B$, we expect that any structure T — modeled as a type family $T : \mathcal{U} \to \mathcal{U}$ — on A gives rise to the same structure on B; that is,

we want a function $T(A) \to T(B)$. In other words, we want the types A and B to be indiscernible which amounts to having an identity $A = B$. However, there is no way to obtain an identity from an equivalence of types in ITT in general. For example, the type of booleans have two automorphisms, one being the identity function and the other being the function swapping its elements, however there is no identity corresponding to this second equivalence.

Voevodsky then drafted a model of type theory in simplicial sets in which equivalence of types coincides with identities between types leading to the simplicial model as we know it (KAPULKIN and LUMSDAINE 2021). Although there is no term in ITT corresponding to this principle, it is consistent to assume it as the axiom known as the *univalence axiom*

$$\mathsf{UA} : (X\ Y : \mathcal{U}) \to (X = Y) \simeq (X \simeq Y)$$

such that $\mathsf{UA}(X, X, \mathsf{refl}_X) = \mathsf{id}_X$.

We understand this axiom as formalising the *equivalence principle*: any statement about types should be invariant under equivalence. The idea of having a logical foundation respecting the principle of equivalence is not new and was the original motivation for the FOLDS system (MAKKAI 1995).

A family of type theories with models in cubical sets have since been developed in order to give a computational meaning to UA, they are referred to as cubical type theories (ANGIULI et al. 2021; BEZEM, COQUAND and HUBER 2014; COHEN et al. 2016).

Related to the UA is the development of the univalent foundations of mathematics which refers to the practice of mathematics in mathematical foundations satisfying the univalence axiom, ITT + UA being one of them. This project was popularised by UNIVALENT FOUNDATIONS PROGRAM 2013 which is the result of the special year on Univalent Foundations of Mathematics which took place at the Institute for Advanced Study in Princeton in 2012-2013. In this book is presented homotopy type theory which is ITT + UA + HITs, also known as book HoTT, as well as a number of mathematical results in homotopy theory, category theory, set theory, and logic developed in this new foundation.

A.5 Our setting

We introduce the type theory used in this thesis. This section is self-contained and will recall some definitions which were already discussed in the preceding sections.

Our type theory is the one implemented by Agda which is an enhanced version of Martin-Löf's logical framework (MARTIN-LÖF and SAMBIN 1984). We will make heavy use of the most cutting-edge features of Agda such as inductive-inductive definitions, rewriting rules, and coinductive records to name a few. We will be concerned with two kinds of judgements. The judgement $x : A$ means that x is an element of type A while the judgement $x \equiv y : A$ means that x and y are *definitionally* equal elements of type A. There is a type universe à la Russell $\mathcal{U}_n : \mathcal{U}_{n+1}$ for $n : \mathbb{N}$. We will intentionally omit the universe level in order not to clutter the notation and simply write $\mathcal{U}$.

We introduce the basic type formers of our type theory that we will use throughout this thesis.

A.5.1 Function types

We open this section with a stylistic remark. Our notation for application departs from the one used in the preceding sections in that we will write $x\ y$ instead of $x(y)$ to lessen the number of parentheses in large lambda terms.

Let $A, B : \mathcal{U}$ be two types, we form the type of functions with domain A and codomain B denoted $A \to B$. Given a term $t : B$ which may contain an occurrence of a variable $x : A$, we introduce the function $(\lambda x \to t) : A \to B$ where x is now bound. Conversely, given a function $f : A \to B$ and a term $t : A$, we define the application of f to t denoted $f\ t : B$. The application satisfies the computation rule:

$$(\lambda x \to u)\ t \equiv u[t/x]$$

for any term $u : B$ possibly containing an occurrence of the variable $x : A$ and for any term $t : A$ where $u[t/x]$ is the capture-avoiding substitution of the occurrences of the variable x in the term u with the term t. Functions moreover satisfy the following uniqueness rule, also called η-law:

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

Function types are now generalised to *dependent* types also named Π-types. Let $A : \mathcal{U}$ be a type and let $B : A \to \mathcal{U}$ be a type family, we form the type $(x : A) \to B\ x$. The other rules are similar to ordinary function types with the difference that the variable x in the introduction and elimination rules can now appear in $B\ x$. Given an element $t : B\ x$ for any variable $x : A$ such that t may contain an occurrence of x, we introduce the function $(\lambda x \to t) : (x : A) \to B\ x$ where x is now bound. Conversely, given a function $f : (x : A) \to B\ x$ and an element $x : A$, we define the application of f to x denoted $f\ x : B\ x$. The application satisfies the computation rule:

$$(\lambda x \to t)\ y \equiv t[y/x]$$

for any term $y : A$ and where $t : B\ x$ is a family of terms indexed by the variable $x : A$ such that t possibly contains an occurrence of the variable $x : A$. Similarly, dependent function types satisfy their own η-law. In the case B is the constant type family $\lambda x \to C$ where $C : \mathcal{U}$ is a type where x is not free, the dependent function type $(x : A) \to B\ x$ coincides with the ordinary function type $A \to C$.

When writing dependent function types whose codomain is itself a dependent function type; that is, types of the form $(x_1 : A_1) \to \cdots \to (x_n : A_n) \to B$, we instead write $(x_1 : A_1) \dots (x_n : A_n) \to B$ without the intermediate arrows. Also, if we have n consecutive binders with the same type as in $(x_1 : A) \to \cdots \to (x_n : A) \to B$, we instead group them together and write $(x_1 \dots x_n : A) \to B$.

Finally, we introduce a notation for implicit arguments of functions. When the argument of a function can be inferred from the type of its application, this argument can be omitted in the notation. We use curly brackets for implicit

arguments such that $f : \{x : A\} \to B\ x$ is a function whose argument x is implicit. This notation is particularly convenient in nested function types when an argument depends on a number of previous arguments. For example, given a function $f : \{x\ y : A\}\ (z : B\ x\ y) \to C\ x\ y\ z$ and an element $z : B\ x\ y$ for x and y two elements of A, we denote $f\ z$ the application of f to the arguments x, y, and z but omitting the first two in the notation. Not all arguments can be made implicit, and we will only employ them when it is easy for the reader to infer them. Sometimes we will want to make explicit the implicit arguments in which case we write $f\ \{x\}\ \{y\}\ z$.

A.5.2 Σ-types

Σ-types are the types of dependent pairs. Let $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, we form the type $\sum_{(x:A)} B\ x : \mathcal{U}$. A pair of elements $x : A$ and $y : B\ x$ defines an element $(x, y) : \sum_{(x:A)} B\ x$. Conversely, given a pair $p : \sum_{(x:A)} B\ x$, we can project its two components as $\mathsf{pr}_1\ p : A$ and $\mathsf{pr}_2\ p : B\ (\mathsf{pr}_1\ p)$ such that, for any $x : A$ and $y : B\ x$, $\mathsf{pr}_1\ (x, y) \equiv x$ and $\mathsf{pr}_2\ (x, y) \equiv y$ which constitute the two β-laws of Σ-types. Moreover, Σ-types satisfy the η-law $p \equiv (\mathsf{pr}_1\ p, \mathsf{pr}_2\ p)$ for any $p : \sum_{(x:A)} B\ x$.

When defining a mathematical structure as a type endowed with a number of operations each satisfying some laws, we implicitly formalise it as a nested sigma type. For example, if we have to write the definition of an associative magma, we will say that it is the data of

- A type $X : \mathcal{U}$.
- A binary operation $m : X \times X \to \mathcal{U}$.
- A proof that m is associative; that is, for any triplet $x, y, z : X$, a proof of $m\ (m\ x\ y)\ z = m\ x\ (m\ y\ z)$.

but implicitly we have to think of this definition as being formalised as the type

$$\sum_{(X:\mathcal{U})}\ (\sum_{(m:X\times X\to\mathcal{U})}\ (x\ y\ z : X) \to m\ (m\ x\ y)\ z = m\ x\ (m\ y\ z))$$

We allow coinductive definitions as implemented in Agda (ABEL, PIENTKA et al. 2013), in which case we drop the η-law as it could lead to non-terminating type-checking. For example, the type Stream_A of stream of type A is defined as the data of

- An element of type A.
- A stream Stream_A.

A.5.3 Inductive types

Most of the types that we will be using fall within the scope of inductive types which, intuitively, are freely generated by their introduction rules that we name *constructors*. Their elimination rules then naturally follow from their constructors and consist of a clause specifying what to do for each of the constructors. We introduce a few inductive types which will be used throughout this thesis.

Empty type The empty type $\bot$ has no constructor. Given a type family $A : \bot \to \mathcal{U}$, defining a function $f : (x : \bot) \to A\ x$ requires no clause corresponding to the fact that there is no constructor inhabiting this type. Its induction principle is

$$\bot\text{-elim} : (A : \bot \to \mathcal{U})\ (x : \bot) \to A\ x$$

Unit type The unit type $\mathbf{1}$ has a single constructor $\star$. Given a type family $A : \mathbf{1} \to \mathcal{U}$, defining a function $f : (x : \mathbf{1}) \to A\ x$ requires a single clause corresponding to its only constructor

$$f\ \star :\equiv f_\star$$

where $f_\star : A\ \star$.

Natural numbers The type of natural numbers $\mathbb{N}$ have two constructors:

$$\begin{aligned} 0 &: \mathbb{N} \\ \mathsf{suc} &: \mathbb{N} \to \mathbb{N} \end{aligned}$$

Given a type family $A : \mathbb{N} \to \mathcal{U}$, a function $f : (n : \mathbb{N}) \to A\ n$ is defined by the following equations

$$\begin{aligned} f\ 0 &:\equiv f_0 \\ f\ (\mathsf{suc}\ n) &:\equiv f_{\mathsf{suc}}\ n\ (f\ n) \end{aligned}$$

where

$$\begin{aligned} f_0 &: A\ 0 \\ f_{\mathsf{suc}} &: (n : \mathbb{N}) \to A\ n \to A\ (\mathsf{suc}\ n) \end{aligned}$$

Sum types Given two types A and B, we form the type $A + B$ of the disjoint union of A and B. It has two constructors:

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

Given a family $C : A + B \to \mathcal{U}$, a function $f : (x : A + B) \to C\ x$ is defined by the following equations

$$\begin{aligned} f\ (\mathsf{inl}\ a) &:\equiv f_{\mathsf{inl}}(a) \\ f\ (\mathsf{inr}\ b) &:\equiv f_{\mathsf{inr}}(b) \end{aligned}$$

where $f_{\mathsf{inl}} : (a : A) \to C\ (\mathsf{inl}\ a)$ and $f_{\mathsf{inl}} : (b : B) \to C\ (\mathsf{inr}\ b)$.

A.5.4 Identity types

Identity types model the notion of *propositional* identity. Let $A : \mathcal{U}$ be a type. Given two elements $x, y : A$, we form their identity type $x =_A y : \mathcal{U}$ dropping the type annotation and writing $x = y$ instead when it can be inferred from the context. Its sole constructor is $\mathsf{refl}_x : x = x$ for any element $x : A$. Identity types $x =_A y$ depart from ordinary inductive types in that they are inductively defined *family* of types indexed by y. The sole constructor of identity types then specifies an inhabitant of the fibre over x.

In order to define the elimination principle for identity types, we consider a type family $B : (y : A) \to x = y \to \mathcal{U}$. Note how B depends on y being variable owing to the fact that y is the index of this inductively defined family of types. Now, in order to define an element of $B\ y\ p$ for any element $y : A$ and identity $p : x = y$, it suffices to define an element $d : B\ x\ \mathsf{refl}_x$. Intuitively, we imagine p to be a path whose y endpoint is free in such a way that we can contract p to refl_x then use d. Moreover, if p was definitionally equal to refl_x in the first place, we just return d.

To summarise, the induction principle has type:

$$\begin{aligned} =_{\mathsf{ind}} : {} & \{A : \mathcal{U}\}\ \{x : A\}\ (B : \{y : A\} \to x = y \to \mathcal{U}) \\ & \to (d : B\ \mathsf{refl}_x) \\ & \to \{y : A\}\ (p : x = y) \\ & \to B\ p \end{aligned}$$

subject to the equation

$$=_{\mathsf{ind}}\ B\ d\ \mathsf{refl}_x \equiv d$$

This induction principle reduces to the following substitution principle when B does not depend on the particular identity:

$$\mathsf{transport} : \{A : \mathcal{U}\}\ (B : A \to \mathcal{U})\ \{x\ y : A\}\ (p : x = y) \to B\ x \to B\ y$$

The use of transport is ubiquitous in ITT and can often obfuscate the crux of a proof. We will therefore adopt the notation

$$\mathsf{transport}^B\ p\ u$$

in place of $\mathsf{transport}\ B\ p\ u$ in order to highlight the term being transported which is u in the present case. When B is the identity function $\mathsf{id}_{\mathcal{U}}$, we will even omit B and write $\mathsf{transport}\ p\ u$.

We recall a number of definitions relative to HoTT which make use of identity types and which will be used throughout this thesis.

Inverse Let A be a type, let $x, y : A$, and let $p : x = y$. There exists an identity $y = x$ that we denote p^{-1}. It is defined by induction on p.

Composition Let A be a type, let $x, y, z : A$, let $p : x = y$, and let $y = z$, then there exists an identity $x = z$ that we denote $p \cdot q$. It is defined by induction on p or q. The composition satisfies the following *propositional* laws for any $p : x = y$, $q : y = z$, and $r : z = t$:

$$p \cdot \mathsf{refl}_y = \mathsf{refl}_x \cdot p = p$$
$$p \cdot p^{-1} = \mathsf{refl}_x$$
$$p^{-1} \cdot p = \mathsf{refl}_y$$
$$(p \cdot q) \cdot r = p \cdot (q \cdot r)$$

This defines a structure of groupoid on types.

n-Types We recall that types can be classified into n-types: types which are contractible above the dimension n. We will in particular use the explicit definition of the following three particular cases.

Definition A.5.1 (Contractible type)**.** A type A is contractible if there exists an element $x : A$ such that, for all elements $y : A$, we have $x = y$. The element x is named the *centre of contraction*.

$$\mathsf{is\text{-}contr}\ A :\equiv \textstyle\sum_{(x:A)} (y : A) \to x = y$$

Definition A.5.2 (Proposition)**.** A type A is a proposition if any two elements $x, y : A$ are equal.

$$\mathsf{is\text{-}prop}\ A :\equiv (x\ y : A) \to x = y$$

Definition A.5.3 (Set)**.** A type is a set if, for any two elements $x, y : A$ and any two identities $p, q : x = y$ then $p = q$.

$$\mathsf{is\text{-}set}\ A :\equiv \{x\ y : A\}\ (p\ q : x = y) \to p = q$$

Moreover, it can be shown that $\mathsf{is\text{-}contr}(A)$, $\mathsf{is\text{-}prop}(A)$, and $\mathsf{is\text{-}set}(A)$ are all propositions.

We now give a definition of equivalence of types based on contractible functions; we refer the reader to Chapter 4 of the HoTT book (Univalent Foundations Program 2013) for other equivalent definitions.

Definition A.5.4 (Homotopy fibre)**.** The homotopy fibre of a function $f : A \to B$ at a point $y : B$ is the subtype of A which has image y under f.

$$\mathsf{fib}_f\ y :\equiv \textstyle\sum_{(x:A)} f\ x = y$$

We define equivalences as functions whose fibres are contractible.

Definition A.5.5 (Equivalence)**.** A function $f : A \to B$ is an equivalence if for every element $y : B$, the type $\mathsf{fib}_f\ y$ is contractible.

$$\mathsf{is\text{-}equiv}\ f :\equiv (y : B) \to \mathsf{is\text{-}contr}\ (\mathsf{fib}_f\ y)$$

This, in turn, allows us to define equivalences of types.

Definition A.5.6 (Equivalence of types)**.** An equivalence of types $A \simeq B$ is a function $A \to B$ together with a proof that it is an equivalence. We therefore define

$$A \simeq B :\equiv \sum_{(f:A\to B)} (\mathsf{is\text{-}equiv}\ f)$$

We will often use the more practical definition of *quasi inverse*. It is logically equivalent to an equivalence of types, but it is not as well-behaved as it is in general not a proposition. However, from a quasi inverse we can always define an equivalence of types. When defining an equivalence we will then often just provide a quasi inverse.

Definition A.5.7 (Quasi inverse)**.** A function $f : A \to B$ has a quasi-inverse if there exists a function $g : B \to A$ such that

$$f \circ g = \mathsf{id}_A$$
$$g \circ f = \mathsf{id}_B$$

We will denote e^{-1} the inverse of an equivalence e. We finally recall the univalence axiom. First, notice that for any two types A and B, there is a function

$$\mathsf{id\text{-}to\text{-}equiv} : A = B \to A \simeq B$$

defined by induction which associates to any identity $p : A = B$ the function

$$\lambda x \to \mathsf{transport}^{\mathsf{id}_{\mathcal{U}}}\ p\ x$$

together with a proof that it is an equivalence — easily proven by an induction on p.

Definition A.5.8 (Univalence axiom)**.** For any two types A and B, the function id-to-equiv is an equivalence. In particular, we have

$$(A = B) \simeq (A \simeq B)$$

It is well known that univalence implies function extensionality.

Definition A.5.9 (Function extensionality)**.** Let $A : \mathcal{U}$ be a type and let $B : A \to \mathcal{U}$ be a family of types. Let $f, g : (x : A) \to B\ x$ be two functions, if f and g are pointwise equal then $f = g$.

$$\mathsf{funext} : \{A : \mathcal{U}\}\ \{B : A \to \mathcal{U}\}\ \{f\ g : (x : A) \to B\ x\}$$
$$\to (p : (x : A) \to f\ x = g\ x) \to f = g$$

The definition uses the univalence axiom and can be found in Section 4.9 of the HoTT book.

We define a last operation which relates pairs of identities and identities of pairs.

Definition A.5.10. Let $A : \mathcal{U}$ and let $B : A \to \mathcal{U}$ be a type family. Let $x, y : A$, let $u : B\ x$, and let $v : B\ y$. If we have an identity $p : x = y$ along with an identity $\mathsf{transport}^B\ p\ u = v$ then there is an identity $(x, u) = (y, v)$ in the type $\sum_{(x:A)} B\ x$.

$$\begin{aligned}\mathsf{pair{=}} : &\{A : \mathcal{U}\}\ \{B : A \to \mathcal{U}\}\ \{x\ y : A\}\ \{u : B\ x\}\ \{v : B\ y\} \\ &\to (p : x = y)\ (q : \mathsf{transport}^B\ p\ u = v) \\ &\to (x, u) = (y, v)\end{aligned}$$

Moreover, pair= is an equivalence. The proof is done by induction on p and q.

Appendix B

Extended abstract (French version)

La définition de structures algébriques sur des types arbitraires en théorie des types homotopiques (HoTT) s'est révélée hors de portée jusqu'à présent. Cela est dû au fait que les types sont, en général, des espaces plutôt que de simples ensembles, et que les égalités d'éléments d'un type se comportent comme des homotopies. Les lois équationnelles des structures algébriques doivent donc être énoncées de manière cohérente. Cependant, en mathématiques ensemblistes, la présentation de ces données de cohérence se fait à l'aide de structures algébriques sur des ensembles, telles que les opérades ou les préfaisceaux, qui ne sont par conséquent pas soumises à des conditions de cohérence supplémentaires. Reproduire cette approche en HoTT conduit à une situation de dépendance circulaire puisque ces structures doivent être définies de manière cohérente dès le départ.

Dans cette thèse, nous brisons cette circularité en étendant la théorie des types d'un univers de monades polynomiales cartésiennes qui, de manière cruciale, satisfont leurs lois définitionnellement. Cette extension permet de présenter les types et leurs structures supérieures sous forme de types opétopiques qui sont des collections infinies de cellules dont la géométrie est décrite par les opétopes.

Opétopes

Nous ouvrons cette thèse en donnant une définition des opétopes dans une théorie des types similaire à celle du livre HoTT au chapitre 1. Les opétopes sont des formes géométriques introduites par Baez et Dolan afin de donner une définition des n-catégories. Les opétopes de dimension $n + 1$ ont pour source une collection d'opétopes de dimension n et ont pour cible un unique opétope de dimension n. Nous définissons les opétopes comme séquences d'arbres bien fondés satisfaisant certaines propriétés qui sont capturées par leur typage. L'approche opétopique est particulièrement adaptée au contexte de la théorie des types car ces arbres sont aisément définissables comme types inductifs.

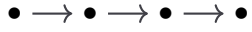
Plus précisément, notre construction est basée sur une séquence de monades polynomiales cartésiennes, une notion qui devient centrale dans les chapitres suivants.

Nous décrivons informellement quelques exemples d'opétopes en basse dimension afin d'en saisir l'intuition.

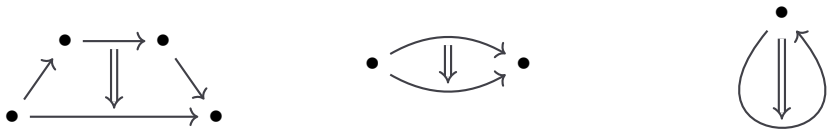
0-opétope Il y a un seul opétope de dimension 0 que l'on représente par un point.

1-opétope Il y a un seul opétope de dimension 1 que l'on représente par une flèche. Elle a une unique source et une cible qui sont tous deux des 0-opétopes.

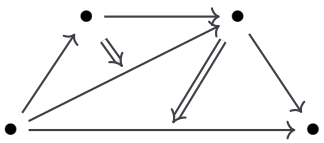
À partir de cette dimension, on peut former des composites formels de n-opétopes aussi appelés *schémas de composition*. En dimension 1 ce ne sont que des chaînes de 1-opétopes comme celle-ci :



2-opétopes Les 2-opétopes ont une source constituée d'une chaîne de 1-opétopes (possiblement vide) et ont pour cible l'unique 1-opétope. Les diagrammes suivants illustrent des exemples de 2-opétopes :

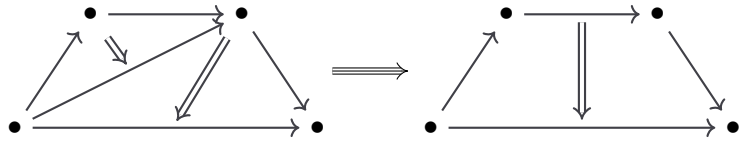


Les 2-opétopes peuvent être assemblés pour former des schémas de composition de 2-opétopes en collant la cible d'un 2-opétope à l'une des sources d'un second 2-opétope comme illustré sur le diagramme suivant :



3-opétopes Nous présentons un dernier exemple d'opétope dont la source est un schéma de composition de 2-opétopes et dont la cible est un 2-opétope *parallèle* en cela que le schéma source et l'opétope cible ont tous deux la même

source et la même cible.



Ce chapitre est l'occasion pour le lecteur de se familiariser avec les opétopes en théorie des types qui sont des ensembles et qui échappent donc aux considérations de cohérence avant d'aborder les types opétopiques dont la complexité peut obscurcir la simplicité conceptuelle des opétopes.

Monades polynomiales cartésiennes

Nous étendons ensuite la théorie des types d'un univers de monades polynomiales cartésiennes $\mathcal{M}$ clos sous certains constructeurs de monades au chapitre 2 qui nous servira à définir les types opétopiques au chapitre 3.

Les monades polynomiales cartésiennes constituent la fondation de notre système. Elles sont au cœur de la construction dite de Baez-Dolan (Baez et Dolan 1998) qui a été introduite dans le cadre des opérades dans le but de définir les n-catégories. Les monades polynomiales cartésiennes peuvent être considérées comme des présentations de théories algébriques fortement régulières. Ce sont des théories algébriques dont les équations sont contraintes de telle sorte que les variables doivent apparaître des deux côtés sans répétition et dans le même ordre (Leinster 2004). À partir de maintenant, nous ne préciserons pas systématiquement que nos monades polynomiales sont cartésiennes même si elles le sont toujours afin d'alléger la notation.

Les monades polynomiales se prêtent bien à une définition en théorie des types en tant que familles indexées. Les éléments de notre univers $\mathcal{M}$ sont considérés comme des codes pour nos monades. Pour toute monade $M : \mathcal{M}$, la donnée définissant son endofoncteur sous-jacent est spécifiée par les familles de types et la fonction de typage suivantes :

$$\begin{aligned}
&\mathsf{Idx}_M : \mathcal{U}\\
&\mathsf{Cns}_M : \mathsf{Idx}_M \to \mathcal{U}\\
&\mathsf{Pos}_M : \{i : \mathsf{Idx}_M\} \to \mathsf{Cns}_M\ i \to \mathcal{U}\\
&\mathsf{Typ}_M : \{i : \mathsf{Idx}_M\}\ (c : \mathsf{Cns}_M\ i) \to \mathsf{Pos}_M\ c \to \mathsf{Idx}_M
\end{aligned}$$

où $\mathcal{U}$ représente l'univers des types.

Ces données peuvent être considérées comme une description de la signature d'une théorie algébrique : les éléments de Idx, que nous appelons *indices*, sont les sortes de la théorie, et pour chaque index $i : \mathsf{Idx}$, le type $\mathsf{Cns}\ i$ est le types des symboles d'opérations dont la sorte de "sortie" est i. Le type $\mathsf{Pos}\ c$ est alors le type des "positions d'entrée" de l'opération c auxquelles sont affectées un index via la fonction Typ.

Dans cette thèse, nous utiliserons un langage graphique pour représenter les constructeurs de nos monades polynomiales. Un constructeur est représenté

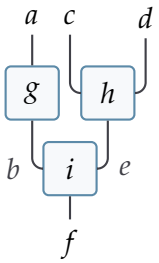
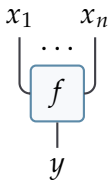


FIGURE B.1 : A P-tree

par une *corolle* dont la sortie pointe vers le bas et dont les entrées pointent vers le haut. Dans la représentation suivante d'une corolle, le constructeur est étiqueté f, les entrées sont étiquetées x_i pour $1 \leq i \leq n$, et la cible est étiquetée y.



Pour tout foncteur polynomial P, ses constructeurs peuvent arrangés en arbres appelés P-arbres (Figure B.1) à condition que, pour chaque arête interne, l'indice de la cible et l'indice de la source des constructeurs correspondants soient les mêmes. La structure supplémentaire conférant à un foncteur polynomial M une structure de monade polynomiale cartésienne est spécifiée, en partie, par les opérations suivantes :

$$\eta_M : (i : \mathsf{Idx}_M) \to \mathsf{Cns}_M\ i$$
$$\mu_M : \{i : \mathsf{Idx}_M\}\ (c : \mathsf{Cns}_M\ i)\ (d : (p : \mathsf{Pos}_M\ c) \to \mathsf{Cns}_M\ (\mathsf{Typ}_M\ c\ p)) \to \mathsf{Cns}_M\ i$$

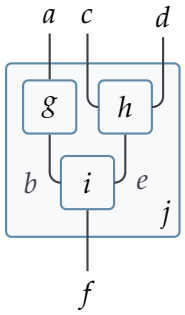
Ces opérations doivent satisfaire certaines lois que nous ne détaillons pas dans cette section, mais nous commentons les deux opérations.

Pour chaque indice $i : \mathsf{Idx}_M$, il existe un constructeur unaire $\eta_M\ i : \mathsf{Cns}_M\ i$ dont l'index d'entrée est i. Nous pouvons représenter ce constructeur comme suit :

En ce qui concerne l'opération μ, ses arguments comprennent un constructeur c ainsi qu'une famille de constructeurs d indexée par les positions de c. Nous considérons ces données comme spécifiant un arbre de profondeur 2 tel que

celui représenté à gauche de la figure suivante. L'opération μ envoie alors cet arbre sur un constructeur partageant les mêmes entrées et la même sortie tel que celui représenté à droite. Le fait que les positions des entrées ainsi que leur typage soient préservés par l'opération μ est dû au fait que nos monades sont cartésiennes.

Une autre façon de représenter un arbre ainsi qu'un constructeur partageant les mêmes sources et cibles consiste à dessiner l'arbre à l'intérieur du constructeur en question.



En outre, l'opération μ est associative et unitaire, les unités étant spécifiées par l'opération η. Nous soulignons que nous mettrons en œuvre notre univers de monades polynomiales en étendant la théorie des types de manière que ces lois soient définitionnelles.

Contrairement à notre définition des opétopes qui n'implique que des ensembles, les types opétopiques sont à valeurs dans des types arbitraires. Par conséquent, nous ne pouvons plus énoncer les lois équationnelles du chapitre 1 de façon cohérente. Nous définissons donc notre univers de monades polynomiales afin que ces lois soient satisfaites par définition. Les constructeurs sous lesquels notre univers est clos nous permettent alors de définir, en particulier, la construction tranche de Baez-Dolan sur laquelle repose notre définition des types opétopiques.

En plus de cet univers qui constitue le cœur de notre ajout à la théorie des types, nous définissons deux extensions supplémentaires. Tout d'abord, pour toutes monades M et N, nous définissons un univers de morphismes cartésiens de monades de M vers N dénoté $M \to_m N$. Ensuite, pour toute monade M, nous définissons un univers de monades polynomiales au dessus de M dénoté $\mathcal{M}\downarrow_M$. Ces deux extensions permettent d'établir des résultats plus avancés au chapitre 3.

$$X_0 : \mathsf{Fam}_M$$
$$X_{>0} : O_{M/X_0}$$

FIGURE B.2 : Définition de O_M

Construction de Baez-Dolan

Notre univers de monades est clos sous la construction tranche de Baez-Dolan (BAEZ et DOLAN 1998). Étant donné une monade $M : \mathcal{M}$ et une famille de types $X : \mathsf{Fam}_M$ avec $\mathsf{Fam}_M :\equiv \mathsf{Idx}_M \to \mathcal{U}$, nous définissons une nouvelle monade M/X. Son type d'indices est $\sum_{(i,y):\sum_{i:\mathsf{Idx}_M} X\ i} \sum_{c:\mathsf{Cns}_M\ i} \overrightarrow{X}\ c$, le types des constructeurs de M dont les entrées et la sortie sont décorées par des éléments de X, où $\overrightarrow{X}\ c :\equiv (p : \mathsf{Pos}_M\ c) \to X\ (\mathsf{Typ}_M\ c\ p)$ est le type de décorations des entrées de c avec des éléments bien typés de X.

Nous nommons ces quadruplets *cadres* et les dénotons $(i, y) \triangleleft (c, x)$. Nous considérons les familles de type $\mathsf{Fam}_{M/X}$ comme des familles de remplissages pour ces cadres mettant en relation une configuration d'entrées spécifiée par (c, x) avec la sortie (i, y).

Les constructeurs de M/X indexés par un cadre $(i, y) \triangleleft (c, x)$ sont alors des arbres bien fondés de cadres qui se multiplient en $(i, y) \triangleleft (c, x)$ en utilisant la multiplication μ_M pour réduire les constructeurs et en oubliant les décorations des arêtes internes. La corolle ayant $(i, y) \triangleleft (c, x)$ pour seul nœud est une unité pour la monade M/X. La grande force de l'approche opétopique est que de tels arbres sont très commodément définis comme types inductifs. Cependant, leur indexation utilisant la structure des monades, nous avons besoin que les lois des monades soient définitionnelles afin de ne pas rencontrer de problèmes de cohérence.

Étant donné un constructeur $c : \mathsf{Cns}_{M/X}\ i$, son type de positions $\mathsf{Pos}_{M/X}\ c$ est le type de *chemins* de la racine de c à ses différents nœuds. La fonction de typage attribue alors le cadre correspondant à un nœud de c à une position spécifiée.

Enfin, la multiplication $\mu_{M/X}$ prend un arbre $c : \mathsf{Cns}_{M/X}\ i$ ainsi qu'une famille d'arbres $d : \overrightarrow{\mathsf{Cns}_{M/X}}\ c$ indexés par les positions de c et substitue les arbres spécifiés par d aux nœuds de c. Nous définissons également cette multiplication de manière à ce qu'elle soit associative et unitaire définitionnellement.

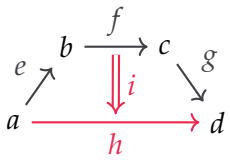
Types opétopiques

Nous tirons parti de notre univers de monades polynomiales pour définir les types opétopiques au chapitre 3. Étant donné une monade M, un type M-opétopique $X : O_M$ (Figure B.2) est une collection coinductive de cellules opétopiques. On note X_n la $(n+1)$-ième famille de cellules d'un type opétopique X et on note $X_{>n}$ le type opétopique des familles X_m pour $m > n$.

Nous qualifions un type opétopique $X : O_M$ de *fibrant* s'il possède la

propriété coinductive suivante. Premièrement, pour tout constructeur décoré $(c, x) : \sum_{c:\mathsf{Cns}_M\ i} \overrightarrow{X_0}\ c$, il existe un unique composite $y : X_0\ i$ ainsi qu'un remplissage de type $X_1\ ((i, y) \triangleleft (c, x))$. En d'autres termes, le type $\sum_{y:X_0\ i} X_1\ ((i, y) \triangleleft (c, x))$ est contractile. Deuxièmement, le type opétopique $X_{>0}$ est fibrant. Autrement dit, pour tout schéma de composition de n-cellules, il existe un unique composite de n-cellules partageant le même cadre. Ces types opétopiques représentent des algèbres cohérentes.

À titre d'exemple, un type opétopique fibrant garantit l'existence des cellules rouges affichées sur le diagramme suivant étant donné le schéma de composition affiché en noir :



De plus, la donnée rouge formée de la paire comprenant la cellule h et la cellule i témoignant que h est un composite de son schéma de composition source est *unique*. Par unique, nous entendons que le type de paires dont (i, h) est membre est contractile. La notion de contractilité est fondamentale en HoTT et est très naturellement définie.

Applications

Nous clôturons cette thèse par la définition de structures algébriques supérieures cohérentes et nous motivons ces définitions en prouvant un certain nombre de résultats attendus. Les types opétopiques fibrants nous permettent de définir, en particulier, les infini-groupoïdes et les (infini, 1)-catégories. De manière cruciale, leur structure supérieure est univalente en ce sens qu'elle coïncide avec celle induite par leurs types d'identités. Ensuite, nous comparons notre définition de type opétopique fibrant à la définition d'algèbre cohérente de Baez et Dolan, et nous montrons qu'elles sont équivalentes sous certaines hypothèses. Puis, nous établissons que les types opétopiques indexés par la monade identité et dont la famille de 1-cellules est un ensemble sont équivalents aux précatégories telles que définies dans le livre HoTT. Nous nous tournons vers les monades dépendantes auxquelles nous avons fait allusion plus tôt afin d'établir les résultats restants. Nous commençons par définir les fibrations de types opétopiques qui sont une généralisation des types opétopiques fibrants, puis nous montrons que les types opétopiques fibrants sont clos sous les sommes dépendantes. Nous définissons également la fibration universelle des types opétopiques. Enfin, nous définissons les (op)fibrations de Grothendieck que nous appliquons à la définition de Lurie d'une adjonction comme bifibration au-dessus de l'intervalle.