Higher Groups via Displayed Univalent Reflexive Graphs in Cubical Type Theory

Master thesis by Johannes Philipp Manuel Schipp von Branitz

Date of submission: October 22, 2020

1. Review: Prof. Dr. Thomas Streicher
2. Review: Dr. Ulrik Buchholtz
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Mathematics Department
Logic

Higher Groups via Displayed Univalent Reflexive Graphs in Cubical Type Theory

Master thesis by Johannes Philipp Manuel Schipp von Branitz

1. Review: Prof. Dr. Thomas Streicher
2. Review: Dr. Ulrik Buchholtz

Date of submission: October 22, 2020

Darmstadt

Abstract

This thesis introduces displayed univalent reflexive graphs, a natural analogue of displayed categories, as a framework for uniformly internalizing composite mathematical structures in homotopy or cubical type theory. This framework is then used to formalize the definition of and equivalence of strict 2-groups and crossed modules. Lastly, foundations for the development of higher groups from the classifying space perspective in cubical type theory are laid. Most results are formalized in Cubical Agda.

Contents

1 Introduction

Homotopy type theory, HoTT in short, is a flavour of intensional Martin-Löf dependent type theory, where identities are first class citizens – between any two terms there is an inductive type of identifications between them. Identifications, now being terms of the same type themselves, give rise to types of identifications between identifications, and so on and so forth.

One fundamental idea in the use of HoTT is Voevoedsky's *univalence axiom*. Univalence states that identifications of types are 'the same as' equivalences between types. This broadens the scope of equality, allowing isomorphic structures to be identified.

The HoTT book [Uni13] gives an excellent introduction to univalent foundations. It proposes homotopy type theory as a foundation for mathematics while explaining how reasoning can be done informally within the type theory, avoiding technicalities, but preserving the ability to formalize any statement if necessary.

Buchholtz [Buc19] discusses how close HoTT is to being a valid foundation for all of mathematics. Central to that discussion is the problem of defining certain higher structures in the type theory. Some structures, like $(\infty, 1)$-categories, seem impossible to construct in standard HoTT. Overcoming these issues is an active area of research. One outcome of this research has been a variety of type theories. For instance, some have additional axioms or type formers, or two levels of types.

Cubical type theory (CTT) is one of the products of this research. Instead of inductive identity types, there are path types. A path is a function on a formal unit interval. An object parametrized by n variables of that interval can be viewed as an n-dimensional cube.

CTT resolves the problem that the univalence axiom – being an axiom – has no computational meaning in HoTT. Derivability of univalence and hence also function extensionality renders cubical type theory a candidate for a computational foundation for univalent mathematics.

There are many variants of CTT. In this paper we assume the variant implemented by Cubical Agda where the Kan composition operations are decomposed into homogeneous composition and generalized transport as described in [CHM18].

The main difference between the type theories in question lies in their definition of identity or path types. Consequently, proofs for the basic theory of identity types can be quite different in nature. However, once this initial set-up has been overcome and one moves on to internalizing known mathematical structures, it would be preferable to be able to use approximately the same style of reasoning across the different type theories.

For example, proving that general dependent paths are equivalent to non-dependent paths between transported terms, depends crucially on the Kan operations of that cubical type theory. Conversely, the standard algebraic proof that group actions are the same as split monomorphisms in the category of groups should not require a lot of modification when internalized in different type theories.

The *structure identity principle* (SIP) is the informal precept that isomorphic objects are equal (cf. [CD13] and [Acz12]). A characterization of the identity types is desirable, because it allows to reason about the structure in question while avoiding complex arguments about abstract identity types in favour of a perhaps more familiar notion of sameness.

For instance, the SIP for groups states that the type of isomorphisms between two arbitrary groups is equivalent to the type of paths between them. Univalence can be considered to be the SIP for types without additional structure.

It is natural to ask for which kinds of structure their identity principle is provable and to what extent there is a general schema for obtaining such proofs. An up-to-date description of a slight variation of the SIP for the standard notion of structure due to Escardó [Esc20] and its implementation in Cubical Agda is given in [Ang+20].

In this thesis we present a more general way of constructing composite structures and obtaining characterizations of their identity types along the way. The central notion is that of a displayed univalent reflexive graph. We use it to formalize strict 2-groups and crossed modules as structures on a group and prove the equivalence thereof. In addition to that, we use displayed univalent reflexive graphs to define higher groups from the classifying space perspective and prove that classifying spaces of 1-groups are equivalent to axiomatic groups.

A complete introduction to univalent foundations and CTT is beyond the scope of this thesis. The curious reader should consult the gentle introductions given in [Ort19] and

[Uni13]. Buchholtz [Buc19] offers a general and non-technical discussion of higher structures. Some meta-mathematical and philosophical aspects of univalent foundations are discussed by Awodey [Awo13] The technical details of CTT are presented in [Coh+18] and [CHM18].

1.1 Overview

We give a short summary of the cubical setting of this thesis in Chapter 2. In doing that, we fix notation for the rest of this work and recall results from cubical type theory to be used later on.

Taking this foundation for granted, we investigate *univalent reflexive graph structures* (URG) in Chapter 3. These made their first appearance as discrete reflexive graphs in [Rij19]. Motivated by the concept of *displayed categories* (cf. [AL19]), we define *displayed* univalent reflexive graph structures (DURG). A DURG over a URG $\mathfrak{S}_A$ on a type A is equivalent to 'a URG $\mathfrak{S}_B$ and a morphism of graphs $F : \mathfrak{S}_B \to \mathfrak{S}_A$', but with the data of the univalent relation of $\mathfrak{S}_B$ and the structure-preserving map F arranged as a family of univalent graphs indexed by A via the *fibrant replacement* of F.

We illustrate how DURGs give rise to a system in which the data of composite structures can be arranged in a uniform and modular way. They allow constructions on structures to be done systematically using one's favorite univalent relation instead of path or identity types. Unlike displayed categories, they are not limited to 1-truncated objects. In general, they serve as an abstraction which reduces redundancy and increases modularity in proofs. As a consequence, the style of reasoning becomes more classical, and less specific to the exact type theory the proof is formalized in. In that manner, we expect that the proofs formulated within the framework of DURGs, as it is implemented in CTT, can easily be carried over to HoTT.

Moving on to Chapter 4: Within the framework of DURGs, we define *strict 2-groups* and *crossed modules*. Classically, a 2-group $\mathcal{G}$ is a group object in the category of groupoids. Here, the category of groupoids can be seen as a $(2,1)$-category, or as an ordinary $(1,1)$-category. If the latter is the case, then $\mathcal{G}$ is called a *strict* 2-group. A group object in the category of groupoids is a groupoid $\mathcal{G}$ equipped with a product functor $\mathcal{G} \times \mathcal{G} \to \mathcal{G}$, which satisfies evident unit and associativity laws; in general, these hold up to coherent natural isomorphism. In a strict 2-group, however, they are proper equalities.

Identity types have structure themselves. For instance, they can be concatenated and that operation is associative up to higher identifications. This is the slogan 'types are ∞-groupoids'. It follows that pointed connected types may be viewed as higher groups. We can impose a truncation requirement on a pointed connected type to obtain the notion of an n-group.

There is no similar approach for *strict* n-groups. Hence, we internalize strict 2-groups as a split monomorphism of groups that has an additional retraction combined with a vertical composition operation.

A *crossed module* is a group action together with an equivariant homomorphism which additionally satisfies the so-called Peiffer condition. It is well known [MM10] that classically, the towers of structures that make up strict 2-groups and crossed modules are equivalent on various levels. Actions are the same as split monomorphisms and precrossed modules are the same as internal reflexive graphs in the category of groups. With the intermediate step of Peiffer graphs, we shall finally see that crossed modules are equivalent to strict 2-groups. DURGs enable us to prove each equivalence of the next level by pulling back the structure that level adds across the equivalence of the previous level while avoiding identity types of objects.

Chapter 5 lays some foundations for the theory of higher groups in cubical type theory, following [BDR18]. It can be read independently of Chapter 4. By requiring that a pointed n-truncated type also be k-connected, one obtains better behaved higher groups, known as *k-tuply groupal n-groupoids* or *(n, k)-groups*.

We transfer these ideas to cubical type theory, again, by means of DURGs. Pointedness, connectivity and truncatedness can be added to a type *simultaneously*. This gives a simple proof of the SIP for (n, k)-groups.

Our final result is that $(0, 1)$-groups are equivalent to axiomatic groups. The SIP for $(0, 1)$-groups and the first Eilenberg-MacLane space are the main ingredients thereof.

Roughly, an Eilenberg-MacLane space is a certain higher inductive type of which a specified homotopy group is isomorphic to a prescribed group. Following [LF14], we internalize the construction of the first EM space of a group, taking advantage of the general schema for higher inductive types that works in this CTT.

Chapter 6 is about the formalization[1] of the results of the previous chapters in Cubical Agda. We briefly discuss design choices and performance.

[1] `https://github.com/Schippmunk/cubical/blob/thesis/Cubical/Papers/HigherGroupsViaDURG.agda`

In the last chapter we discuss our results, comparing DURGs to displayed categories and the standard notion of structure. We point out potential, related future work.

1.2 Contributions

It was Buchholtz' idea to define displayed univalent reflexive graphs in order compose structures. The general theory of DURG structures is joint work.

The equivalence of strict 2-groups and crossed modules on the three levels presented is well known (cf. [MM10]). This is the first time these objects have been internalized into a cubical type theory and formalized in a proof assistant by means of displayed structures. Note that von Raumer [Rau15] proved the equivalence of the precategories of crossed modules (in Brown's sense) and double groupoids in HoTT. However, it is not immediate that that equivalence restricts to our case. A characterization of the identity types of the objects in question is also not part of his exposition.

The equivalence between pointed connected 1-types and groups has already been formalized in HoTT using the proof assistant Lean [BDR18]. The internalization here is novel in that cubical higher inductive types do not require any postulated terms and that some cubical arguments are simpler.

1.3 Acknowledgments

I am very grateful to Professor Thomas Streicher for supervising this project. Dr. Ulrik Buchholtz deserves my sincere gratitude for his patience, guidance and positive spirit. Working with you was a joy!

Moreover, I am thankful to Andrea Vezossi for giving me hints regarding performance issues in Agda.

I very much appreciate the love and support I have received from my dear parents Ilona and Helmut ever since, but especially during this last term of my studies.

Last but not least, I would like to thank my friends – in particular Anna, Lena, Sarah and Nutsa – for their help and support.

2 Cubical Type Theory

This chapter introduces notation for dependent type theory with path types, together with the minimal information needed to make sense of the Kan operations transport and homogeneous composition, leading to a collection of cubical groupoid laws.

We remark that a lot of technicalities are skipped here, such as face formulas, partial elements, glue types and proof of the correspondence between dependent paths and identities of transported elements. This is because those details can be treated as a black box. All of the proofs in the following chapters merely depend on the truth of these properties, not what combination of Kan operations was used to derive them. This allows us to keep the focus on intuitive cubical reasoning.

In Section 2.4, we recall function extensionality and univalence. The HoTT book gives a good account of the meaning of these principles in type theory.

In the last two sections we collect facts and lemmas about truncated and connected types, as well as the axiomatic treatment of groups in type theory. [Rij18] serves as a good reference.

2.1 Dependent Type Theory

Dependent type theory à la Martin-Löf consists of types containing terms or inhabitants, and judgemental or definitional equality, denoted $\equiv$. Judgementally equal objects cannot be distinguished by the type theory.

New types can be formed using dependent function types $(x : A) \to B\,x$, classically denoted by $\prod_{x:A} B(x)$ and called Π-types. There are also dependent pair types $(x : A) \times B\,x$, called Σ-types, which are traditionally denoted by $\sum_{x:A} B(x)$. Our notation is close to that of the programming language Agda, and we use it because the deeply nested dependent sum types are more readable this way. Elements of Π- and Σ-types, that is, dependent functions and dependent pairs, can be constructed using λ-abstraction and the pair constructor $\langle \_ , \_ \rangle$, respectively. For example,

$$(\lambda \langle a, b\rangle \mapsto a) : (\langle a, b\rangle : (a : A) \times B\, a) \to B\, a$$

is the judgement that the dependent function which takes a dependent pair $\langle a, b\rangle$ of type $(a : A) \times B\, a$ and returns its second component is of the specified type.

As we saw in the pair constructor, we use underscores to indicate the position of the arguments of special function symbols. When applying a function we often leave arguments implicit. In a function definition these are sometimes denoted by curly braces. Functions enjoy judgemental β-reduction and η-expansion, meaning that λ-abstraction computes correctly, and every function is a λ-abstraction. Function application is denoted $f\ x$, rather than $f(x)$. We commonly overload symbols, often leaving components of a Σ-type implicit. For example, $G \to_{\mathrm{Grp}} H$ denotes the type of homomorphisms between groups G and H, while $G \to H$ denotes the type of maps between the underlying types of G and H.

The type theory features a large class of *inductive types*. An inductive type consists of a list of constructors, an induction principle and their computation rules; for instance, the natural numbers form an inductive type $\mathbb{N}$ generated by $0_{\mathbb{N}}$ and the successor function $S_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$. Let $B : \mathbb{N} \to \mathrm{Type}$ be a type family over $\mathbb{N}$. The induction principle of $\mathbb{N}$ is a term

$$\mathrm{ind}_{\mathbb{N}} : (B\, 0_{\mathbb{N}}) \to ((n : \mathbb{N}) \to (B\, n) \to B\, (S_{\mathbb{N}}\, n)) \to (n : \mathbb{N}) \to (B\, n).$$

It states exactly that to prove a statement about all natural numbers, one has to prove it for $0_{\mathbb{N}}$ and provide a function that turns a $p : B\, n$ into a term of type $B\, (S\, n)$ for any $n : \mathbb{N}$. The computation rules state that $\mathrm{ind}_{\mathbb{N}}$ computes correctly: $\mathrm{ind}_{\mathbb{N}}\ p\, f\ 0_{\mathbb{N}} \equiv p$ and $\mathrm{ind}_{\mathbb{N}}\ p\, f\ (S_{\mathbb{N}} n) \equiv f\ n\, (\mathrm{ind}_{\mathbb{N}}\ p\ f\ n)$.

To be able to quantify over types while avoiding inconsistencies like Russel's paradox, the type theory features type *universes*. As is common in the literature, we take the liberty of only using one universe, called Type. The formalization can be consulted in order to see the rigorous use of universes.

2.2 Path Types

In the homotopical interpretation a type A can be seen as a space, a term $a : A$ as a point of that space, a function $f : A \to B$ as a continuous map, a path as a continuous map $[0,1] \to A$ from a to b and a type family $B : A \to \mathrm{Type}$ as a fibration. Based on this intuition is the idea that cubes are the fundamental shapes used to capture the structure of higher-dimensional mathematical objects.

The formal interval $\mathbf{I}$

In cubical type theory, the real interval is abstracted to the formal *interval type*[1] $\mathbf{I}$ with the two endpoints $\mathrm{i}_0\,\mathrm{i}_1 : \mathbf{I}$.[2] We assume a De Morgan algebra structure on $\mathbf{I}$. This means that $\mathbf{I}$ consists of countably many variable symbols and the minimum, maximum and inversion operations $\wedge : \mathbf{I} \to \mathbf{I} \to \mathbf{I}$, $\vee : \mathbf{I} \to \mathbf{I} \to \mathbf{I}$ and $\sim : \mathbf{I} \to \mathbf{I}$ such that $\mathbf{I}$ is a bounded distributive lattice with De Morgan involution $\sim$.[3] Note that we do not assume the law of excluded middle $(\sim i) \vee i = \mathrm{i}_1$, nor its counterpart, the law of noncontradiction.

Given any two terms $a\,b : A$ we can form the path type $a = b$ in A. Paths are similar to functions; given a path $p : a = b$ and an interval variable $i : \mathbf{I}$, we can judge that $p\,i : A$. Applying a path to one of the interval endpoints results in the corresponding endpoint of the path, i.e., there are judgemental equalities $p\,\mathrm{i}_0 \equiv a$ and $p\,\mathrm{i}_1 \equiv b$. Paths are construted by abstraction. If $a : A$ is a term which may depend on $i : \mathbf{I}$, then $(\lambda i \mapsto a[i]) : a[\mathrm{i}_0] = a[\mathrm{i}_1]$. As with functions, we assume judgemental β-reduction and η-expansion. That is, path abstraction computes correctly, and every path is a path abstraction.

Cubes

A term parametrized by n variables of $\mathbf{I}$ can be seen as an n-dimensional cube and it is often useful to visualize such objects diagrammatically. A 0-dimensional cube is just a point, an element of A.

$$\cdot\, a$$

A 1-dimensional cube p is a function $\mathbf{I} \to A$. It can be visualized as a line in 'direction' i.

$$\begin{tikzcd} p\,\mathrm{i}_0 \arrow[r, "p\,i"] & p\,\mathrm{i}_1 & & {} \arrow[r, "i"'] & {} \end{tikzcd}$$

[1] $\mathbf{I}$ is not formally a type, although it shares many properties of proper types. It is sometimes called a pretype.
[2] By not placing commas between multiple postulated terms of a type we stay close to Agda's notation.
[3] In other cubical type theories $\mathbf{I}$ can have a different, usually weaker structure, see for example [AHH18].

A 2-dimensional cube $p : \mathbf{I} \to \mathbf{I} \to A$ can be thought of as a homotopy of paths. Of course, the corresponding picture is a square.

$$\begin{tikzcd}
p\,\mathrm{i}_0\,\mathrm{i}_1 \arrow[r, "\lambda\, i \mapsto p\, i\, \mathrm{i}_1"] & p\,\mathrm{i}_1\,\mathrm{i}_1 \\
p\,\mathrm{i}_0\,\mathrm{i}_0 \arrow[u, "\lambda\, j \mapsto p\, \mathrm{i}_0\, j"] \arrow[r, "\lambda\, i \mapsto p\, i\, \mathrm{i}_0"'] \arrow[ur, phantom, "p"] & p\,\mathrm{i}_1\,\mathrm{i}_0 \arrow[u, "\lambda\, j \mapsto p\, \mathrm{i}_1\, j"']
\end{tikzcd}$$

The term p is also called a *filler* of the square. We note that the sides or faces of the square are paths themselves, but of dimension 1. In the third dimension, paths are homotopies of homotopies and can be visualized as cubes. The general pattern is clear. At dimension $n+1$, there are $(n+1)$-dimensional paths visualizable as hypercubes with $2(n+1)$ faces which are formed by n-dimensional cubes.

Dependent Paths

Since types are terms of a universe, this cubical structure applies to types as well. This means there is a type $A = B$: Type of paths between any two types AB : Type. Every such path $p : A = B$ is a *type line* $\mathbf{I} \to$ Type with end points $p\,\mathrm{i}_0 \equiv A$ and $p\,\mathrm{i}_1 \equiv B$. It thus makes sense to assume a type of paths between specified terms which may belong to different types.

$$\mathrm{PathP} : (A : I \to \mathrm{Type}) \to A\,\mathrm{i}_0 \to A\,\mathrm{i}_1 \to \mathrm{Type}$$

From this, the homogeneous path type can be derived.

$$\begin{aligned} \_=\_ &: (A : \mathrm{Type}) \to A \to A \to \mathrm{Type} \\ a = b &:\equiv \mathrm{PathP}\,(\lambda\,\_ \mapsto A)\; a\; b \end{aligned}$$

This is why heterogeneous equality PathP is taken to be the primitive equality type.

Example 2.1 (Paths in Σ-types)**.** Let $B : A \to$ Type be a type family over A : Type and $\langle a, b\rangle\,\langle a', b'\rangle : (a : A) \times B\,a$. The data needed to construct a path $\langle a, b\rangle = \langle a', b'\rangle$ is exactly a path $p : a = a$ together with a dependent path $q : \mathrm{PathP}\,(\lambda\, i \mapsto B(p\, i))\; b\; b'$.

Kan Operations

In the cubical setting we assume two Kan operations – *transport* and *composition* – to ensure that there are enough paths and to determine how higher paths compute.

The general transport operation of this particular type theory is

$$\mathrm{transp} : (A : \mathbf{I} \to \mathrm{Type}) \to (i : \mathbf{I}) \to (a : A\,\mathrm{i}_0) \to A\,\mathrm{i}_1.$$

The parameter i specifies where transp is the identity function. It is taken as a primitive in Cubical Agda. Normal transport can be defined in terms of transp.

$$\begin{aligned} &\mathrm{transport} : \{A B : \mathrm{Type}\} \to (A = B) \to A \to B \\ &\mathrm{transport}\ p\ a\ :\equiv\ \mathrm{transp}\ p\ \mathrm{i}_0\ a \end{aligned}$$

Homogeneous composition generalizes binary composition of paths. It states that any open box specified by u and u_0 has a lid p.

$$\begin{tikzcd} a \arrow[r, dashed, "p"] & b \\ c \arrow[u, "u"'] \arrow[r, "u_0"] & d \arrow[u, "u"'] \end{tikzcd}$$

Here u is a *partial element*, roughly meaning it only needs to be defined on the left and right side of the cube, and coincide with u_0 where the two paths meet.

Using homogeneous composition, we can show that dependent paths can be obtained from paths of transported terms and vice versa. How this works is explained in detail in [CHM18, p.18].

Proposition 2.2. *Let* $A : \mathbf{I} \to \mathrm{Type}$ *be a line of types with elements* $a_0 : A\mathrm{i}_0$ *and* $a_1 : A\mathrm{i}_1$ *at the end points. Then*

$$\begin{aligned} &(\mathrm{transport}\,(\lambda\, i\ \mapsto A\, i)\, x) = y) \\ \equiv &((\mathrm{transp}\,(\lambda\, i\ \mapsto A\, i)\,\mathrm{i}_0\, x\,) = y) \\ = &\,\mathrm{PathP}\ (\lambda\, i\ \mapsto A\, i)\; a_0\; a_1. \end{aligned}$$

With this correspondence established, we can discard transp; transport suffices for our purposes.

Note that in HoTT, a dependent path between a and b over p is usually *defined* to be a path between a transported along p, and b, i.e., the left hand side in above equation. It is sometimes convenient to have this cubical, more symmetric dependent path type – particularly when dependent paths can be constructed directly (without using transport). Of course, one could also construct dependent identity types as inductive types. Being inductive, they would still be different from cubical dependent path types.

2.3 Cubical Groupoid Laws

We make more precise the intuition of types being ∞-groupoids by establishing some basic and useful groupoid laws.

For any type A and $a : A$ we see that

$$\mathrm{refl}_a :\equiv (\lambda\, i \mapsto a) : (a = a).$$

In other words, we can use constant functions out of $\mathbf{I}$ to prove that $=$ is reflexive. The a is sometimes left implicit.

In HoTT, the approach is entirely different. Identity types in HoTT are inductive types generated by a single constructor refl and thus they obey judgemental computation rules, unlike the reflexivity term in CTT. This is one of the main reasons why the cubical style of reasoning differs from the HoTT style.

It is true that transporting along refl is the identity function.

$$\begin{aligned}&\mathrm{transportRefl} : (a : A) \to \mathrm{transport}\ \mathrm{refl}_a\ a = a\\ &\mathrm{transportRefl}\, a :\equiv \lambda\, i \mapsto \mathrm{transp}\,(\lambda\, \_ \mapsto A)\, i\, a\end{aligned}$$

Substitution can be seen as a transport in an explicitly given type family.

$$\begin{aligned}&\mathrm{subst} : (B : A \to \mathrm{Type}) \to (a = a') \to B\, a \to B\, a'\\ &\mathrm{subst}_B\ p\, b :\equiv \mathrm{transport}\,(\lambda\, i \mapsto B(p\, i))\, b\end{aligned}$$

Since path types are not inductive, they do not automatically come with an induction principle. However, *identity induction* (J) can be derived.

Proposition 2.3. *Assume a type family P parametrized by $a\ b : A$ and $p : a = b$. If $P\ a\ a\ \mathrm{refl}_a$, then $P\ a\ b\ p$ holds for all $p : a = b$.*

Functions preserve paths as the homotopy interpretation leads to expect. Let $f : A \to B$ and $a\ b : A$. Then the action of functions on paths is defined as

$$\begin{aligned} &\mathrm{ap}_f : (a = b) \to (f\ a = f\ b) \\ &\mathrm{ap}_f\ p :\equiv \lambda\, i \mapsto f\ (p\ i) \end{aligned}$$

It satisfies

$$\mathrm{ap}_f\ \mathrm{refl}_a \equiv (\lambda\, i \mapsto f\ (\mathrm{refl}_a i)) \equiv (\lambda\, i \mapsto f\ a) \equiv \mathrm{refl}_{f\ a}.$$

Function application strictly respects composition in the sense that

$$\mathrm{ap}_{g \circ f}\ p \equiv \mathrm{ap}_g (\mathrm{ap}_f\ p).$$

This makes ap_f a *strict functor*.

The De Morgan involution $\sim$ of $\mathbf{I}$ allows us to define path inversion directly. If $p : a = b$ then $p^{-1} :\equiv \lambda\, i \mapsto p(\sim i) : b = a$. Note that there is a judgemental equality $\mathrm{refl}^{-1} \equiv \mathrm{refl}$.

The fundamental operation on paths is composition. The most natural definition of homogeneous path composition is double composition. Given composable paths

$$\begin{tikzcd} a \arrow[d, "p"] & d \\ b \arrow[r, "q"] & c \arrow[u, "r"'] \end{tikzcd}, \tag{2.1}$$

we can use homogeneous composition to obtain a lid $p \cdot\cdot\, q \cdot\cdot\, r : a = d$ of that open box:

$$\begin{tikzcd} a \arrow[r, dashed, "p\cdot\cdot q\cdot\cdot r"] & d \\ b \arrow[u, "p^{-1}"] \arrow[r, "q"] & c \arrow[u, "r"'] \end{tikzcd} \tag{2.2}$$

Any homogeneous composition comes with a *filler*. In the case of double composition, this is a path of type

$$\mathrm{PathP}\ (\lambda\, i \mapsto p^{-1}\, i = r\ i)\ q\ (p \cdot\cdot\, q \cdot\cdot\, r).$$

Double path composition, with the constraint that (2.2) can be filled, is unique up to a path, in the sense that for any

$$P\,Q : (s : a = d) \times \mathrm{PathP}\,(\lambda\, i \mapsto p^{-1}\, i = r\, i)\, q\, s$$

we have that $P = Q$.

Closely related is the next lemma. It allows us to decompose fillers into double composition identities.

Lemma 2.4. *For any square*

$$\begin{tikzcd}
a \arrow[r, "s"] & d \\
b \arrow[u, "p^{-1}"] \arrow[r, "q"] & c \arrow[u, "r"]
\end{tikzcd}$$

there is a path of type

$$(\mathrm{PathP}\,(\lambda\, i \mapsto p^{-1}\, i = r\, i)\, q\, s) = (p \cdot\cdot\, q \cdot\cdot\, r = s).$$

Binary composition is defined as $q \cdot r :\equiv \mathrm{refl} \cdot\cdot\, q \cdot\cdot\, r$. This binary composition operation is what turns a type into a higher groupoid. One can prove all the usual unit, cancellation and associativity laws up to a path. Of course, these paths are subject to higher coherences.

Luckily, it does not matter if we choose to use double composition or single composition twice.

Lemma 2.5. *For composable paths as in* (2.1) *there is a path of type*

$$p \cdot\cdot\, q \cdot\cdot\, r = p \cdot q \cdot r.$$

The idea of the proof of Lemma 2.5 is to inflate the double composition with the necessary reflexivity terms:

$$p \cdot\cdot\, q \cdot\cdot\, r = \mathrm{refl} \cdot\cdot\, (p \cdot\cdot\, q \cdot\cdot\, \mathrm{refl}) \cdot\cdot\, r$$

Afterwards one applies an instance of

$$\mathrm{refl} \cdot\cdot\, p \cdot\cdot\, q = p \cdot\cdot\, q \cdot\cdot\, \mathrm{refl}.$$

The details of the homogeneous compositions needed to construct these paths can be found in the formalization.

Lemma 2.6. *Assume that the bottom and top faces of the following cube have a filler.*[4]

$$\begin{tikzcd}
 & a \arrow[rr, "p'"] \arrow[dl, equal] \arrow[dd, equal] & & b \arrow[dl, equal] \arrow[dd, equal] \\
a \arrow[rr, "p"] \arrow[dd, equal] & & b \arrow[dd, equal] & \\
 & a \arrow[rr, "q'"] \arrow[dl, equal] & & b \arrow[dl, equal] \\
a \arrow[rr, "q"] & & b &
\end{tikzcd}$$

Then the type of fillers of the front face is the same as the type of fillers of the back face. In other words,

$$(p = q) = (p' = q').$$

Proof. Let $P :\equiv p = p'$ and $Q :\equiv q = q'$. We can use the action of

$$\begin{aligned} (r : a = b) &\to \mathrm{Type} \\ r &\mapsto r = q \end{aligned}$$

on P to obtain an equality

$$(p = q) = (p' = q).$$

Similarly, we then use the action of

$$\lambda\, r \mapsto p' = r$$

on Q to obtain

$$(p' = q) = (p' = q'). \qquad \square$$

2.4 Functions, Equivalences and Univalence

In this section we introduce contractible types in order to define equivalences and isomorphisms. This leads us to function extensionality and univalence.

[4] Double lines are used to denote refl.

One of the most basic inductive types is the *unit* type $\mathbf{1}$. It has a single point constructor $* : \mathbf{1}$.

A type A is called *contractible* or a (-2)-*type*, if there is a term $\langle a, p \rangle$ of type

$$(a : A) \times ((b : A) \to a = b).$$

This is the type theoretic way of saying that A has only one point (up to homotopy). In this example, a is called the *center* of the *contraction* p.

For example, the type $\mathbf{1}$ is contractible with center of contraction $*$. The contraction is provided by the induction principle of $\mathbf{1}$. Contractible types are the simplest types from a homotopical perspective. More examples of contractible types are *singletons*.

Proposition 2.7. *For any type A and any term $a : A$, the* singleton $(a' : A) \times (a = a')$ *of A at a is contractible.*

Proof. We take the pair $\langle a, \mathrm{refl} \rangle$ to be the center of contraction. Let $\langle b, p \rangle : (b : A) \times (a = b)$. To construct a path of type $\langle a, \mathrm{refl} \rangle = \langle b, p \rangle$ it suffices to find a $q : a = b$ and a filler of the square:

$$\begin{tikzcd}
a \arrow[r, "p"] \arrow[d, equal] & b \\
a \arrow[r, equal] & a \arrow[u, "q"']
\end{tikzcd}$$

We take $q :\equiv p$. Then $\lambda\, i\, j \mapsto p(i \wedge j)$ fills the square. □

Definition 2.8. Let $f : A \to B$ be a function. The *(homotopy) fiber* of f at $b : B$ is the type

$$\mathrm{fib}_f\ b :\equiv (a : A) \times (f\ a = b).$$

The map f is called an *equivalence* or *contractible map* if its fibers are contractible. In that case we write $f : A \simeq B$. The map f is called an *isomorphism*, if it has a *quasi-inverse*, that is, a map $g : B \to A$ and proofs that $(a : A) \to g\,(f\ a) = a$ and $(b : B) \to f\,(g\ b) = b$.

One can show that a map is contractible iff it has a quasi-inverse. We often construct an isomorphism and then state that the corresponding types are equivalent.

Proposition 2.9. *Let $p : A = B$. Then*

$$\mathrm{transport}\, p : A \to B$$

is an equivalence.

Its pseudo-inverse is given by transporting along p^{-1}. This is one of the propositions we treat as a black-box. Its proof requires more details about partial elements, so we refer to the implementation.

We shall often use [Uni13, Theorem 4.7.7].

Theorem 2.10. *Let $B\,B' : A \to \mathrm{Type}$ and $f : (a : A) \to B\,a \to B'\,a$. Then f is a fiberwise equivalence iff*

$$\begin{aligned} \mathrm{tot}^f : (a : A) \times B\,a &\to (a : A) \times B'\,a \\ \langle a, b \rangle &\mapsto \langle a, f\;a \rangle \end{aligned}$$

is an equivalence.

Proposition 2.11. *Let $f : A \simeq A'$ and $g : (a : A) \to B\,a \simeq B'\,(f\,a)$. Then*

$$(a : A) \times B\,a \simeq (a' : A') \times B'\,a'.$$

Proof. According to Theorem 2.10, g induces an equivalence

$$(a : A) \times B\,a \simeq (a : A) \times B'(f\,a)$$

on total spaces. One can show that a pseudo-inverse of f induces a pseudo-inverse of

$$\begin{aligned} (a : A) \times B'(f\,a) &\to (a' : A') \times B'\,a' \\ \langle a, b \rangle &\mapsto \langle f\;a, b \rangle. \end{aligned}$$

□

Definition 2.12. Two functions $f\;g : A \to B$ are called *homotopic*, if they are pointwise equal. In other words, if

$$f \sim g :\equiv (a : A) \to (f\;a = g\,a)$$

is inhabited.

One of the prominent features of CTT is function extensionality.

Theorem 2.13. *Let $f\;g : A \to B$. Then there is a pseudo-isomorphism*

$$\begin{aligned}(f \sim g) &\simeq (f = g)\\ H &\mapsto (\lambda\, i\, a \mapsto H\, a\, i)\\ (\lambda\, a\, i \mapsto p\, i\, a) &\leftarrow\!\shortmid p.\end{aligned}$$

Function extensionality refers to the map from left to right; such a map cannot be derived in HoTT. The statement of Theorem 2.13 can be generalized to dependent functions $(a : A) \to B\,a$.

Since types A : Type are also terms, they have identity types as well.

Theorem 2.14. *Univalence holds, i.e., for any two types A and B the natural map*

$$\mathrm{idToEquiv} : (A = B) \to (A \simeq B)$$

defined by identity induction is an equivalence. Call its inverse $\mathrm{idToEquiv}^{-1}$. *These maps satisfy*

$$\mathrm{transport}\,(\mathrm{idToEquiv}^{-1} f\,)\, x = f\, x \qquad (\mathrm{idToEquiv}\text{-}\beta)$$

and

$$\mathrm{idToEquiv}^{-1}(\mathrm{idToEquiv}\, P) = P \qquad (\mathrm{idToEquiv}\text{-}\eta)$$

for all $f : A \simeq B$, $a : A$ and $P : A = B$.

The proof can be found in [Coh+18].

With univalence we can generalize function extensionality.

Lemma 2.15. *Assume that we have two functions and an equivalence arranged as follows:*

$$\begin{tikzcd}B & B' \\ A \arrow[u, "f"] \arrow[r, "\sim", "e"'] & A' \arrow[u, "f'"']\end{tikzcd}$$

If there is $p : B = B'$ and a path of type

$$(a : A) \to \mathrm{PathP}\,(\lambda\, i \mapsto p\, i)\,(f\, a)\,(f'(e\, a)),$$

then there is a path of type

$$\mathrm{PathP}\,(\lambda\, i \mapsto \mathrm{idToEquiv}^{-1}\, e\, i \to p\, i)\, f\; f'.$$

2.5 Truncated and Connected Types

A type A is called a *(mere) proposition* or (-1)-type, if all of its identity types are contractible. An equivalent condition is that for all $a\ b : A$ there is a path $a = b$. The type A is called a *set* or 0-type if its identity types are propositions. Continuing in this fashion, A is called an $(n+1)$-type or an $(n+1)$-truncated type, if all of its identity types are n-types. Intuitively, an n-type can contain non-trivial homotopical information only up to level n. We introduce

$$\mathrm{Type}^{\leq n} :\equiv (A : \mathrm{Type}) \times \mathrm{isTrunc}_n A,$$

the type of n-types. We adhere to the common abuse of notation of writing $A : \mathrm{Type}^{\leq n}$ to state that A is an n-type, leaving the second component implicit.

Lemma 2.16. *Let $B : A \to \mathrm{Type}^{\leq -2}$ be a family of contractible types over the type A. Then*

$$(a : A) \times B\, a \simeq A.$$

In particular, the identity types of $(a : A) \times B\, a$ are equivalent to those of A.

Lemma 2.17. *Let A be a contractible type with center of contraction c. Let $B : A \to \mathrm{Type}$ be a type family over A. Then*

$$(a : A) \times B\, a \simeq B\, c.$$

Lemma 2.18. *Let $B : I \to \mathrm{Type}^{\leq -1}$ be a 'line of propositions'. Then there is a term of type*

$$\mathrm{PathP}\, (\lambda\, i \ \mapsto B\, i)\, b_0\, b_1$$

for all $b_0 : B\, \mathrm{i}_0$ and $b_1 : B\, \mathrm{i}_1$.

Proof. Since $B\, \mathrm{i}_1$ is a proposition, there is a path $(\mathrm{transport}\, B\, b_0) = b_1$. Proposition 2.2 turns it into the desired dependent path. □

A propositional family $P : A \to \mathrm{Type}^{\leq -1}$ can be seen as a *subtype* of A, because $(a : A) \times P\, a$ are the terms of A which satisfy P. The following lemma establishes that the identity types of a type restrict to its subtypes. It follows directly from the characterization of paths in Σ-types, and Lemma 2.18.

Lemma 2.19. *Let $B : A \to \mathrm{Type}^{\leq -1}$ be a subtype of A. Then for all $\langle a, b\rangle\ \langle a', b'\rangle : (a : A) \times B\,a$,*

$$(\langle a, b\rangle = \langle a', b'\rangle) \simeq (a = a').$$

Using dependent paths we can express truncatedness of type families.

Definition 2.20. Let $B : A \to \mathrm{Type}$ be a type family. We say that B is a (-1)-*type over* A, if for all $a\,a' : A$ there is a term of type

$$(p : a = a') \to (b : B\,a) \to (b' : B\,a') \to \mathrm{PathP}\,(\lambda\, i \;\mapsto B(p\,i))\;b\;b'.$$

The family B is said to be a (-2)-*type over* A, if every fiber is inhabited and it is in addition a proposition over A. B is said to be an $(n+1)$-*type over* A, if for all $a\,a' : A$, and b and b' in the fibers over a and a', respectively, the family

$$\lambda\, p \;\mapsto \mathrm{PathP}\,(\lambda\, i \;\mapsto B(p\,i))\;b\;b'$$

is an n-type over A.

In view of Lemma 2.18 it is possible to show that every family $B : A \to \mathrm{Type}^{\leq n}$ is also an n-type over A.

Path induction can be used to show that any identity type of A is homotopically not more complex than the self identity type $a = a$ of any $a : A$.

Proposition 2.21. *Let $n \geq -1$ and $A : \mathrm{Type}$. If $a = a$ is an n-type for all $a : A$, then A is an $(n+1)$-type.*

The next two propositions are well-known (cf. [Uni13, Theorems 7.1.8 and 7.1.10]).

Proposition 2.22. *The type $\mathrm{Type}^{\leq n}$ is closed under taking Σ-types.*

In particular, subtypes of sets are sets.

Proposition 2.23. *Being truncated is a property. In other words, the type $\mathrm{isTrunc}_n A$ is a proposition.*

Proposition 2.24. *If B is an n-type and A is any type, then the type of functions $A \to B$ is an n-type.*

This uses function extensionality and is proved by induction on n. We see that universally quantified statements about equalities in sets are propositions.

Theorem 2.25. *For any type A and $n \geq -2$ there is a type $\|A\|_n : \mathrm{Type}^{\leq n}$, the n-*truncation *of A, together with a map $|\_|_n : A \to \|A\|_n$ satisfying the universal property that for any n–type B the precomposition map*

$$\lambda\, g \;\mapsto g \circ |\_|_n : (\|A\|_n \to B) \to (A \to B)$$

is an equivalence.

The n-truncation of A can be seen as the 'best approximation' of A as an n-type. The truncation can be defined via the *hub and spoke* construction, or as a higher inductive type. For a reference of the former, see [Uni13, Section 7.3]. The definition as a HIT is simple and beautiful. For example, the propositional truncation $\|A\|_{-1}$ of A is generated by the constructors

$$|\_|_{-1} : A \to \|A\|_{-1}$$

and

$$\mathrm{squash} : (a\, b : A) \to a = b.$$

Dual to n-truncated types, there are types which do not have any non-trivial homotopical information below level n.

Definition 2.26. A type A is *n-connected*, if its n-truncation $\|A\|_n$ is contractible. We collect the connected types in

$$\mathrm{Type}^{>n} \;:\equiv\; (A : \mathrm{Type}) \times \mathrm{isConn}_n A.$$

A map $f : A \to B$ between any two types A and B is called *n-connected* if all of its fibers are n-connected.

Lemma 2.27. *For any $A : \mathrm{Type}^{>n+1}$, any map of type $\mathbf{1} \to X$ is n-connected.*

2.6 Groups

The *axiomatic definition a group* in CTT is the same as in HoTT. Namely, a group consists of a set with a specified unit element, a binary composition operation which has inverses, such that the evident unit, cancellation and associativity laws hold. The nested Σ-type of types with this structure and these properties is called Grp. A *homomorphism of groups* is a function on the underlying types that respects the addition operation. We collect the group homomorphisms between $G\,H : \mathrm{Grp}$ in $G \to_{\mathrm{Grp}} H$. If the underlying map of a group homomorphism f is an equivalence, then f is called an *isomorphism of groups* and we write $f : G \simeq_{\mathrm{Grp}} H$.

As an immediate consequence of Proposition 2.22 and Proposition 2.24, we see that maps between groups are homotopically not more complex than groups themselves.

Proposition 2.28. *The types $G \to H$, $G \to_{\mathrm{Grp}} H$ and $G \simeq_{\mathrm{Grp}} H$ are sets for any two groups G and H.*

Proposition 2.29. *Any homomorphism $f : G \to_{\mathrm{Grp}} H$ induces a group structure* $\ker f$, *called the* kernel of f, *on the type*

$$(g : G) \times f\,g = 1.$$

This comes with a forgetful, injective homomorphism

$$\lambda\,\langle g, p\rangle \mapsto g : \ker f \to_{\mathrm{Grp}} G.$$

The next proposition is an extensionality principle for the type of groups. It characterizes the paths between homomorphisms of groups.

Proposition 2.30. *For all $f\;g : G \to_{\mathrm{Grp}} H$ there is an equivalence*

$$(f \sim g) \simeq (f = g).^5$$

Proof. Function extensionality states that $f \sim g$ is equivalent to the paths between the underlying maps of f and g. Since being a homomorphism is a property, the statement follows from Lemma 2.19 and the functorial action of the projection to the first component of a homomorphism. □

[5] Here $f \sim g$ means homotopies of the underlying functions of f and g, and $f = g$ is the identity type in $G \to_{\mathrm{Grp}} H$.

We state the SIP for groups.

Theorem 2.31. *Let G and H be groups. Then there is an equivalence*

$$(G \simeq_{\mathrm{Grp}} H) \simeq (G = H).$$

A proof can be found in [Rij18, Section 14].

We give two more definitions relevant for Chapter 4. The definition of a split monomorphism works in any category. We are only concerned with the special case of the category of groups.

Definition 2.32. Given two homomorphisms $f : G \to_{\mathrm{Grp}} H$ and $g : H \to_{\mathrm{Grp}} G$, we say that $\langle f, g \rangle$ is a *section-retraction-pair*, if it satisfies the *split condition*

$$\mathrm{isSecRet}\langle f, g \rangle :\equiv g \circ_{\mathrm{Grp}} f = \mathrm{id}_H^{\mathrm{Grp}} .$$

If $\mathrm{isSecRet}\langle f, g \rangle$ is inhabited, we say that f is a *split monomorphism*, g a *split epimorphism*, g a *retraction* of f and f a *section* of g.

Definition 2.33. Let G and H be groups. A *group action* of G on H consists of a *left action structure* $\_\alpha\_ : G \to H \to H$ satisfying the following axioms for all $g : G$ and $h\,h' : H$.

$$\begin{aligned} g\,\alpha\,(hh') &= (g\,\alpha\,h)(g\,\alpha\,h') && \text{(pointwise homomorphism)} \\ 1\,\alpha\,h &= h && \text{(identity law)} \\ (gg')\,\alpha\,h &= g\,\alpha\,(g'\,\alpha\,h) && \text{(associativity)} \end{aligned}$$

We call the Σ-type collecting these axioms $\mathrm{isAction}\,\alpha$.

Since we are only interested in split monomorphisms in the category of groups and actions of groups on groups, we no longer qualify these with the word ‘group’.

It is easy to see that $\mathrm{isSecRet}\langle f, g \rangle$ and $\mathrm{isAction}\,\alpha$ are propositions.

Proposition 2.34. *An action α of G on H gives rise to a new group – the* semidirect product *$H \rtimes_\alpha G$ – with carrier type $H \times G$ and group operation*

$$\langle h, g \rangle \cdot_\alpha \langle h', g' \rangle :\equiv \langle h(g\,\alpha\,h'), gg' \rangle.$$

For more information on semidirect products, see [DF04, Section 5.5].

3 Displayed Structures

This chapter is concerned with the process of 'displaying' some structure over another. In the first section we shall give a thorough motivation and explanation as to how this works. The second section contains a summary of the relevant aspects of displayed categories as introduced by Ahrens and Lumsdaine in [AL19]. In the third and fourth sections we define URGs and DURGs, respectively. The fifth section is devoted to general operations such as forming total spaces and lifting DURG structures. The chapter concludes with a theorem about obtaining equivalences between DURG structures using an equivalence of the base types.

3.1 Motivation

Many mathematical structures are composite ones. Starting with some base structure one can sequentially add properties and (higher) structure.

For example, groups can be constructed by starting with a base type, adding a multiplication and inversion structure, and finally group axioms – associativity, unit and inverse laws.

For constructing new categories by adding extra data to objects or morphisms of an existing category, there exists the framework of *displayed categories*.

'A displayed category over a category $\mathscr{C}$ is equivalent to "a category $\mathscr{D}$ and a functor $F : \mathscr{D} \to \mathscr{C}$", but instead of having a single collection of "objects of $\mathscr{D}$", with a map to the objects of $\mathscr{C}$, the objects are given as a family indexed by objects of $\mathscr{C}$, and similarly for the morphisms.'

This is analogous to how a family of sets indexed by a set C is equivalent to a set D with a function to C. To emphasize the analogy, one could call a family of sets over a set a *displayed set*.

While category theorists might not like this perspective, we can consider a category to be a type with some extra structure. The choice of morphisms might depend on the context. For example, on the collection of sets, taking morphisms to be either functions or set inclusions yields distinct category structures. Both structures can be used to prove theorems about sets, but the resulting theorems usually depend on the chosen structure.

One can distinguish different purposes of adding structure to mathematical objects. In the examples above, the category structures were chosen to *highlight* a particular aspect of the nature of the collection of sets. In contrast to that, one could argue that equipping a set with a particular group structure usually serves the purpose of understanding the group structure itself, or how the resulting group behaves with respect to other groups – not what the nature of the elements of the underlying set is like.

The obvious analogue to displayed sets in type theory, a *displayed type* over a type C, would be a type indexed by the terms of C; in other words, a *type family* over C. This data is equivalent to that of a type X together with a function $X \to C$. The transformation of this data is known as *fibrant replacement*.

Proposition 3.1. *For any type C there is an equivalence*

$$\begin{aligned}(X : \mathrm{Type}) \times (X \to C) &\simeq (C \to \mathrm{Type})\\ \langle X, F\rangle &\mapsto (\lambda\, c \ \mapsto \mathrm{fib}_F\, c)\\ \langle (c : C) \times D\, c, \pi_1\rangle &\mapsfrom D\end{aligned}$$

A univalent reflexive graph (URG) is a type A together with a binary relation, a witness that the relation is reflexive, and a witness that the relation is equivalent to the identity types of A (see Definition 3.5). We see that a URG is a type combined with a representation of its intrinsic structure – its identity types.[1]

It is evident that a *displayed* univalent reflexive graph (DURG) (see Definition 3.13) over a URG on a type A should be a type family $B : A \to \mathrm{Type}$ with a reflexive relation $\cong^{\mathrm{D}}_p$ indexed by proofs $p : a \cong a'$ in A. This relation should be univalent at any chosen reflexivity proof $\rho : a \cong a$ in A. We remark that no reference to the identity types of A is necessary.

In the internalization of mathematics into type theory a common task is to characterize the identity types of the newly defined type. Common challenges faced during this process are that identity types of composite structures quickly become very complex and that it is

[1] Reflexive graphs in HoTT first received attention by Egbert Rijke and Bas Spitters in [Rij19]. They were used to develop the theory of reflexive coequalizers.

hard to repeat such characterizations in a structured way. Additionally, for a successful implementation in a proof assistant it is vital that code can be reused.

We shall see how DURGs can live up to these challenges. One fundamental observation is that a DURG structure on a type family induces a URG structure – and hence a characterization of identity types – on the total space (see Theorem 3.16). Furthermore, we show that two DURG structures over the same type can be combined (see Corollary 3.21) and that one of them can be lifted to be displayed over the other (cf. Proposition 3.20). These operations greatly reduce redundancy.

Now, let us recall the basics of displayed categories and explore the details of DURGs.

3.2 Displayed Categories

Definition 3.2. Given a category $\mathscr{C}$, a *displayed category* $\mathscr{D}$ *over* $\mathscr{C}$ consists of

1. for each object $c : \mathscr{C}$, a type $\mathscr{D}_c$ of 'objects over c';
2. for each morphism $f : a \to b$ of $\mathscr{C}$, $x : \mathscr{D}_a$ and $y : \mathscr{D}_b$, a set of 'morphisms from x to y over f', denoted $x \to_f y$;
3. for each $c : \mathscr{C}$ and $x : \mathscr{D}_c$, a morphism $1_x : x \to_{1_c} x$;
4. for all morphisms $f : a \to b$ and $g : b \to c$ in $\mathscr{C}$ and objects $x : \mathscr{D}_a$ and $y : \mathscr{D}_b$ and $z : D_c$, a sequential composition function
$$\cdot : (x \to_f y) \times (y \to_g z) \to (x \to_{f \cdot g} z);$$
5. such that the following dependent properties over equalities of morphisms in $\mathscr{C}$ are satisfied for all $\bar{f} : x \to_f y, \bar{g} : y \to_g z$ and $\bar{h} : z \to_h w$.
   a) $\bar{f} \cdot 1_y =_* \bar{f}$
   b) $1_x \cdot \bar{f} =_* \bar{f}$
   c) $\bar{f} \cdot (\bar{g} \cdot \bar{h}) =_* (\bar{f} \cdot \bar{g}) \cdot \bar{h}$

The asterisks in the axioms indicate *dependent* equalities over equalities of morphisms in $\mathscr{C}$. For instance, if $\bar{f} : x \to_f y$, then $\bar{f} \cdot 1_y : x \to_{f \cdot 1_b} y$, so the displayed right unit axiom is over the ordinary axiom $f \cdot 1_b = f$ of $\mathscr{C}$.

To every displayed category $\mathscr{D}$ over $\mathscr{C}$ there is an associated *total category* $\int_{\mathscr{C}} \mathscr{D}$. It is a category on the total space $(c : \mathscr{C}) \times \mathscr{D}_c$ of objects of $\mathscr{D}$ over $\mathscr{C}$. This comes with an evident forgetful functor $\pi_1 : \int_{\mathscr{C}} \mathscr{D} \to \mathscr{C}$.

A category is called *univalent*, if the natural map idtoiso : $(x = y) \to \mathrm{iso}(a, b)$ is an equivalence. A displayed category $\mathscr{D}$ over $\mathscr{C}$ is called *univalent*, if the displayed isomorphisms characterize the displayed identity types over idtoiso in $\mathscr{C}$.

One particularly useful theorem states that univalence of a total category follows from the univalence of the displayed category.

Theorem 3.3. *If $\mathscr{D}$ is a displayed univalent category over a univalent category $\mathscr{C}$, then the total category $\int_{\mathscr{C}} \mathscr{D}$ is univalent.*

3.3 Univalent Reflexive Graphs

Definition 3.4. A *graph structure* or *binary relation* on a type A is a function $\_ \cong \_ : A \to A \to \mathrm{Type}$. The *type of graphs* is

$$(A : \mathrm{Type}) \times (A \to A \to \mathrm{Type})$$

If there is a term $\rho : (a : A) \to a \cong a$, then $\cong$ is called a *reflexive* graph structure.

Here we use the symmetric symbol $\cong$, because we are only interested in univalent and hence symmetric graphs.

The relation given by identity types on a type is the least reflexive relation on that type. This allows us to define a map

$$\begin{aligned} &\mathrm{idToRel} : (a\, a' : A) \to (a = a') \to a \cong a. \\ &\mathrm{idToRel}\; a\, a'\, p :\equiv \mathrm{subst}\,(\lambda\, z \;\mapsto a \cong z)\, p\, (\rho\, a) \end{aligned}$$

for any a reflexive graph structure $\langle \cong, \rho \rangle$.[2]

[2] We equivalently could have used J rather than subst.

Definition 3.5. The reflexive graph structure $\langle\cong,\rho\rangle$ is called *univalent*, if $\mathrm{idToRel}\, a\, a'$ is an equivalence for any $a\, a' : A$.

The *type of univalent reflexive graph structures* on A is the expected Σ-type. We drop the A subscript when the relation $\cong$ and its reflexivity and univalence witnesses are clear from the context.

Example 3.6. Group isomorphisms, together with the identitity morphisms form a reflexive relation on the type Grp of groups. By the SIP for groups, this relation is univalent. Hence, we have a URG structure $\mathfrak{S}_{\mathrm{Grp}}$ on Grp.

Example 3.7. Every univalent 1-precategory naturally induces a univalent reflexive graph structure on its base type.

To employ some more examples of DURG structures we can use a reformulation of [Rij18, Theorem 10.2.3].

Proposition 3.8. *Let $\cong: A \to A \to \mathrm{Type}^{\leq -1}$ be a reflexive relation. If $\cong$ 'implies identity' – meaning there is $f : (a \cong b) \to (a = b)$ – then all $\cong$-singletons are contractible.*

Proof. Let $a : A$. Since $\cong$ takes values in propositions, the map idToRel is a left inverse to f. Consequently, the total function of f is a left inverse to the total function of idToRel. In other words, $(a' : A) \times a \cong a'$ is a retract of $(a' : A) \times a = a'$, a contractible type. Hence, the former type is also contractible. □

Example 3.9. We previously noted that a URG structure on a type can be understood as an observational equality. We elaborate on this by defining an observational equality on the natural numbers, and showing that it satisfies the axioms of a URG. We define the following relation on $\mathbb{N}$:

$$\begin{aligned} \mathrm{Eq}_{\mathrm{N}} : \mathbb{N} \to \mathbb{N} &\to \mathrm{Type}^{\leq -1} \\ \mathrm{Eq}_{\mathrm{N}}\, 0\, 0 &:\equiv \mathbf{1} \\ \mathrm{Eq}_{\mathrm{N}}\, 0\, (\mathrm{suc}\, n) &:\equiv \emptyset \\ \mathrm{Eq}_{\mathrm{N}}\, (\mathrm{suc}\, n)\, 0 &:\equiv \emptyset \\ \mathrm{Eq}_{\mathrm{N}}\, (\mathrm{suc}\, n)\, (\mathrm{suc}\, m) &:\equiv \mathrm{Eq}_{\mathrm{N}}\, n\, m \end{aligned}$$

Clearly, $\mathrm{Eq}_{\mathbb{N}}$ takes values in $\mathrm{Type}^{\leq -1}$, is reflexive and implies identity. According to Proposition 3.8, it follows that the relational singletons $(m : \mathbb{N}) \times \mathrm{Eq}_{\mathbb{N}}\, n\, m$ are contractible. Hence, $\mathrm{Eq}_{\mathbb{N}}$ defines a URG structure on the natural numbers.

At least when dealing with types that are not sets, showing that the map idToRel is an equivalence becomes difficult. A slight variant of the *fundamental theorem of identity types* (cf. [Rij18, Theorem 9.2.2], [Uni13, Theorem 5.8.4]) is often useful to show that a reflexive graph is univalent.

Theorem 3.10. *For any reflexive graph structure $\langle \cong, \rho \rangle$ on A, the following statements are equivalent.*

- *$\langle \cong, \rho \rangle$ is univalent.*
- *Every* relational singleton *w.r.t. $\cong$ is contractible. In other words, for every $a : A$, the type*

$$\mathrm{singl}_a^{\cong} \equiv (a' : A) \times a \cong a'$$

  is contractible.

Proof. In the context of $a : A$, we define the total function of idToRel:

$$\begin{aligned} \mathrm{tot}_a^{\mathrm{idToRel}} : (a' : A) \times a = a' &\to (a' : A) \times a \cong a' \\ \langle a', p \rangle &\mapsto \langle a', \mathrm{idToRel}\, a\, a'\, p \rangle \end{aligned}$$

If $\langle \cong, \rho \rangle$ is univalent, then $\mathrm{tot}^{\mathrm{idToRel}}$ is an equivalence (Theorem 2.10). The singleton $(a' : A) \times a = a'$ is contractible, and so is $(a' : A) \times a \cong a'$.

Conversely, assume all $\cong$ singletons are contractible. Let $a : A$. Then both the domain and codomain of $\mathrm{tot}_a^{\mathrm{idToRel}}$ are contractible. It follows that $\mathrm{tot}_a^{\mathrm{idToRel}}$ is an equivalence. By Theorem 2.10, if the total function $\mathrm{tot}_a^{\mathrm{idToRel}}$ is an equivalence, then $\mathrm{idToRel}\, a$ is a fiberwise equivalence. This is a reformulation of the univalence of $\langle \cong, \rho \rangle$. □

Example 3.11. Theorem 3.10 provides us with many more examples of URG structures.

1. Every type has a natural URG structure given by its identity types.

2. Every universe Type has a URG structure. The binary relation $\simeq$ is given by equivalences. Identity equivalences witness reflexivity of $\simeq$. Univalence of $\simeq$ is equivalent to Type being a univalent universe.[3]

The univalence axiom of URG structures implies of course that up to homotopy there is exactly one small URG structure on any given type.

Proposition 3.12. *For any A : Type, the type of* small *URG structures on A is contractible.*

Proof. Let $\mathfrak{S} :\equiv \langle \cong, \rho, \mathrm{uni} \rangle$ be any URG structure on A and $\mathfrak{S}' :\equiv \langle =, (\lambda\, a \mapsto \mathrm{refl}_a), \mathrm{uni}' \rangle$ be the one given by identity types. To construct a path $\mathfrak{S}' = \mathfrak{S}$ of a three-component structure we have to come up with three paths – one for each component – with the latter paths depending on the previous ones. A path p between the relations $=$ and $\cong$ is determined by function extensionality and uni. We now construct a term

$$q : \mathrm{PathP}\,(\lambda\, i \mapsto (a : A) \to p\; i\; a\; a)\,(\lambda\, a \mapsto \mathrm{refl}_a)\,\rho\,.$$

Let $a : A$ and put

$$\begin{aligned} u &: (a = a) \simeq (a \cong a) \\ u &:\equiv \langle \mathrm{idToRel}\; a\, a, \mathrm{uni} \rangle. \end{aligned}$$

We have that

$$\begin{aligned} &\mathrm{transport}\,(\lambda\, i \mapsto p\; i\; a a)\,\mathrm{refl}_a \\ =\,&\mathrm{subst}\,(\lambda\, a' \mapsto a \cong a')\,\mathrm{refl}_a\,(\rho\; a) && (\mathrm{idToEquiv}\text{–}\beta) \\ =\,&\mathrm{refl}_a. && (\mathrm{transportRefl}) \end{aligned}$$

Proposition 2.2 turns this path of a transported term into a parametrized dependent path of type

$$\mathrm{PathP}\,(\lambda\, i \mapsto p\; i\; a\; a)\,\mathrm{refl}_a\,(\rho\; a).$$

Function extensionality applied to that path yields q. The univalence axiom of URG structures is a proposition, hence Lemma 2.18 produces a dependent path between uni and uni′ over p and q. □

[3] Actually, in the cubical library, contractibility of the $\simeq$-singletons is used to prove univalence, because it follows from unglue being an equivalence.

3.4 Displayed Univalent Reflexive Graphs

Definition 3.13. Let $\mathfrak{S}_A :\equiv \langle \cong, \rho, \mathrm{uni} \rangle$ be a URG structure on A, and $B : A \to \mathrm{Type}$ be a type family over A. A *displayed univalent graph structure on B over $\mathfrak{S}_A$* consists of

- a relation, called *displayed relation*, over the relation $\cong$ of $\mathfrak{S}_A$, that is, a term

$$\_ \cong^{\mathrm{D}}_{\_} \_ : \{a\, a' : A\} \to B\, a \to a \cong a' \to B\, a' \to \mathrm{Type}$$

- a witness of reflexivity relative to ρ:

$$\rho^{\mathrm{D}} : \{a : A\} \to (b : B\, a) \to b \cong^{\mathrm{D}}_{\rho\, a} b$$

- and a witness of univalence relative to ρ^{D} and ρ,

$$\mathrm{uni}^{\mathrm{D}} : \{a : A\} \to \mathrm{isUnivalent}\,(\lambda\, b\, b' \mapsto b \cong^{\mathrm{D}}_{\rho\, a} b'\,)\, \rho^{\mathrm{D}}.$$

Since a displayed relation already carries a subscript of the kind $p : a \cong a$, we usually drop the $\_^{\mathrm{D}}$ superscript.

We remark that the fundamental theorem of identity types is equally useful for the construction of DURGs as for URGs. This is because it proves that to give a term $\mathrm{uni}^{\mathrm{D}}$ as in Definition 3.13 is to show that for all $a : A$ and every $b : B\, a$ the relational singleton

$$(b' : B\, a) \times b \cong^{\mathrm{D}}_{\rho\, a} b'$$

is contractible.

As a first example, we see that subtypes can be given a simple DURG structure. This is reminiscent of [AL19, Example 3.6]. It aids in imposing axioms on a given structure.

Proposition 3.14. *Let $P : A \to \mathrm{Type}^{\leq -1}$ be a propositional family, and $\mathfrak{S}_A :\equiv \langle \cong, \rho, \mathrm{uni} \rangle$ be a URG structure on the type A. Then there is a DURG structure on P over $\mathfrak{S}_A$ with displayed relation*

$$p \cong_q p' :\equiv \mathbf{1}.$$

Proof. Reflexivity over $\mathfrak{S}_A$ is witnessed by $*$. In view of Theorem 3.10 it now suffices to show that for any $a : A$ and $b : P\,a$ the $\cong_{\rho\,a}$-singleton

$$(b' : P\,a) \times b \cong_{\rho\,a} b'$$

is contractible. By definition, this is just

$$(b' : P\,a) \times \mathbf{1}. \tag{3.1}$$

The unit type is contractible, so we can further reduce (3.1) to $P\,a$. By assumption, $P\,a$ is a proposition and inhabited by $b : P\,a$. Consequently, $P\,a$ is contractible. Note that we did not use univalence of $\mathfrak{S}_A$. □

The types $\mathrm{isConn}_k A$ and $\mathrm{isTrunc}_n A$ are propositions. Whence, Proposition 3.14 yields a DURG structure.

Example 3.15. Connectedness and truncatedness can be displayed over universes, i.e., for any $n\,k : \mathbb{N}$ there are DURG structures on the type families

$$\lambda\,(A : \mathrm{Type}) \mapsto \mathrm{isConn}_k A$$

and

$$\lambda\,(A : \mathrm{Type}) \mapsto \mathrm{isTrunc}_n A.$$

3.5 Operations on Displayed Univalent Reflexive Graphs

As promised, we can form a ‘total URG structure’ for any displayed URG structure – a univalent reflexive graph structure on the total space of the type family. This is the analogue of Theorem 3.3.

Theorem 3.16. *Let $\mathfrak{D}_A^B$ be a DURG over $\mathfrak{S}_A$. Then there is an associated* total URG structure *of $\mathfrak{D}_A^B$, written $\int \mathfrak{D}_A^B$, on the type $(a : A) \times B\,a$ with the binary relation being*

$$\langle a, b\rangle \cong_\Sigma \langle a', b'\rangle :\equiv (e : a \cong a') \times b \cong_e b'.$$

Proof. The relation $\cong_\Sigma$ is reflexive by

$$\lambda\,\langle a, b\rangle \mapsto \langle \rho\, a, \rho^{\mathrm{D}} b\rangle.$$

To show that $\cong_\Sigma$ is univalent, it suffices to show (Theorem 3.10) that all $\cong_\Sigma$-singletons are contractible. Let $\langle a, b\rangle : (a : A) \times B\,a$. Then

$$\begin{aligned}
&(\langle a', b'\rangle : (a : A) \times B\,a) \times (e : a \cong a') \times b \cong_e b' \\
\simeq&(\langle a', e\rangle : (a' : A) \times a \cong a') \times (b' : Ba') \times b \cong_e b' \\
\simeq&(b' : B\,a) \times b \cong_{\rho\,a} b'.
\end{aligned}$$

In the first step we reassociated and swapped independent Σ-types. In the second step we used contractibility of the relational singleton of $\mathfrak{S}_A$ at a. Thus, we are left with the $\cong_{\rho\,a}$-singleton at b, which is also contractible by assumption.

One subtlety is that the center of contraction Theorem 3.10 does in general not coincide with $\langle a, \rho\, a\rangle$. Hence, to apply the assumption of contractibility of $\cong_{\rho\,a}$, the contraction of the relational singleton of $\mathfrak{S}_A$ at a needs to be recentered. □

It is worth noting that $\int \mathfrak{D}_A^B$ is in particular a characterization of the identity types of $(a : A) \times B\,a$.

We use the sketch

$$\begin{tikzcd}
B \arrow[d] \\
A
\end{tikzcd}$$

to illustrate a DURG structure on a type family B over A. The arrow pointing down is justified by the first projection $(a : A) \times B\,a \to A$.

As a corollary to the total space construction we obtain products of URG structures by constructing constant DURG structures.

Corollary 3.17. *Let $\mathfrak{S}_A$ and $\mathfrak{S}_B$ be URG structures on types A and B, respectively. Then there is a DURG structure $\mathfrak{D}_A^B$ on the constant family $\lambda\,\_ \mapsto B$ over A. The displayed relation is given by*

$$b \cong_p b' :\equiv b \cong b',$$

for $p : a \cong a'$. Reflexivity and univalence of the new relation follow immediately from $\mathfrak{S}_B$.

Definition 3.18. Let $\mathfrak{S}_A$ and $\mathfrak{S}_B$ be URG structures on A and B, and $\mathfrak{D}_A^B$ the constant DURG structure as in Corollary 3.17. We define the *product URG structure*

$$\mathfrak{S}_A \times_{\mathfrak{S}} \mathfrak{S}_B :\equiv \int \mathfrak{D}_A^B.$$

The corresponding sketch to Corollary 3.17 and Definition 3.18 is:

$$\begin{tikzcd} & B \arrow[dd] & \\ A \quad B \quad \mapsto & & \mapsto \quad A \times B \\ & A & \end{tikzcd}$$

The next important operation is that of reassociating towers of DURG structures.

Theorem 3.19. *Let $\mathfrak{S}_A$ be a URG structure on A, $\mathfrak{D}_A^B$ a DURG structure on $B : A \to \mathsf{Type}$ over $\mathfrak{S}_A$, and $\mathfrak{D}_B^C$ a DURG structure on $C : ((a : A) \times B\,a) \to \mathsf{Type}$ over $\int \mathfrak{D}_A^B$. Then there is a DURG structure $\mathfrak{D}_A^D$ on the type family*

$$D :\equiv \lambda\, a \mapsto (b : B\,a) \times C\langle a, b\rangle$$

over $\mathfrak{S}_A$.

We have this picture in mind:[4]

$$\begin{tikzcd} C \arrow[dd] & & \\ & & B \times C \arrow[dd] \\ B \arrow[dd] & \mapsto & \\ & & A \\ A & & \end{tikzcd}$$

[4] We draw our towers to associate to the bottom. A diagram $C \to A \times B$ would indicate a URG structure on $(a : A) \times B\,a$ which has not necessarily been obtained form Theorem 3.16. Moreover, note that C depends on B even if the notation $B \times C$ might suggest otherwise. This is to stay consistent with the naming convention in the next chapter. In our applications it is always obvious whether or not the right factor depends on the left.

Proof. We define the displayed relation of $\mathfrak{D}_A^D$ to be

$$\langle b, c\rangle \cong_{p_a} \langle b', c'\rangle :\equiv (p_b : b \cong_{p_a} b') \times c \cong_{\langle p_a, p_b\rangle} c'$$

for any $p_a : a \cong a$ in A. Reflexivity over $\langle b, c\rangle$ is witnessed by $\langle \rho^{\mathrm{D}} b, \rho^{\mathrm{D}} c\rangle$. As per usual, we apply Theorem 3.10 to reduce univalence of the displayed relation to contractibility of all relational singletons. Let $a : A$, and $\langle b, c\rangle : (b : B\,a) \times C\langle a, b\rangle$. We need to prove that

$$(\langle b', c'\rangle : (b' : B\,a) \times C\langle a, b'\rangle) \times (p_b : b \cong_{\rho\,a} b') \times c \cong_{\langle \rho\,a, p_b\rangle} c'$$

is contractible. Swapping and reassociating factors appropriately, we obtain the equivalent type

$$(\langle b', p_b\rangle : (b' : B\,a) \times b \cong_{\rho\,a} b') \times (c' : C\langle a, b'\rangle) \times c \cong_{\langle \rho\,a, p_b\rangle} c'. \tag{3.2}$$

From the univalence assumption on URG structures, and Theorem 3.10, it follows that

$$(b' : B\,a) \times b \cong_{\rho\,a} b'$$

is contractible. We may use our favourite center of contraction $\langle b, \rho^{\mathrm{D}} b\rangle$. This proves that (3.2) is equivalent to

$$(c' : C\langle a, b\rangle) \times c \cong_{\langle \rho\,a, \rho^{\mathrm{D}} b\rangle} c'.$$

This is the relational singleton of $\mathfrak{D}_B^C$ at c, hence contractible. □

A useful operation to reduce redundancy in defining DURG structures is the following lifting operation.

Proposition 3.20. *Let $\mathfrak{S}_A$ be a URG structure on A, $\mathfrak{D}_A^B$ and $\mathfrak{D}_A^C$ be DURG structures over $\mathfrak{S}_A$. Then B can be lifted to be displayed over $\int \mathfrak{D}_A^C$.*

$$\begin{tikzcd}
B \arrow[dr] & & C \arrow[dl] & & B \arrow[d] \\
& A & & \mapsto & C \arrow[d] \\
& & & & A
\end{tikzcd}$$

Proof. The new type family $B' : (a : A) \times C\,a \to \mathrm{Type}$ ignores the C-component, and so does its DURG structure. The displayed relation is given by

$$b \cong_{p_a,p_c} b' :\equiv b \cong_{p_a} b'.$$

Of course, reflexivity and univalence follow immediately from the same properties of $\mathfrak{D}^B_A$. □

We can combine the lifting and the reassociating of DURG structures to obtain the combinator: If two structures can be displayed over a type, then so can their product.

Corollary 3.21. *Let $\mathfrak{S}_A$ be a URG structure on A, and $\mathfrak{D}^B_A$ and $\mathfrak{D}^C_A$ be DURG structures over $\mathfrak{S}_A$. Then there is a DURG structure over $\mathfrak{S}_A$ on the family*

$$\lambda\,(a : A) \mapsto B\,a \times C\,a.$$

$$\begin{tikzcd}
B \arrow[dr] & & C \arrow[dl] & & B \times C \arrow[d] \\
& A & & \mapsto & A
\end{tikzcd}$$

Proof. First, lift $\mathfrak{D}^B_A$ to be displayed over $\int \mathfrak{D}^C_A$. Then apply Theorem 3.19 to the resulting tower. □

3.6 Constructing Equivalences Using Displayed Univalent Reflexive Graphs

In this section we set up some tools which can be used in order to obtain equivalences between composite structures.

Definition 3.22. Between two graphs $\langle A, \cong \rangle$ and $\langle A', \cong' \rangle$, a *relational isomorphism* consists of functions

$$\begin{tikzcd}
A \arrow[r, shift left, "f"] & A' \arrow[l, shift left, "g"]
\end{tikzcd}$$

such that $g(f\,a) \cong a$ and $f(g\,a') \cong' a'$ for all $a : A$ and $a' : A'$. We denote the Σ-*type of relational isomorphisms* by $\mathrm{RelIso}\langle A, \cong \rangle \langle A', \cong' \rangle$.

One could define an *isomorphism of URG structures* as an isomorphism of the underlying relation that also preserves the reflexivity and univalence witesses, as well as *morphisms of URG structures* by dropping the left and right inverse axioms. However, the next proposition shows that an isomorphism of the underlying graphs suffices to induce an equivalence on the underlying types. Morphisms of URG structures are not of interest here.

Proposition 3.23. *Let $f : \mathfrak{S}_A \to \mathfrak{S}_B$ be a relational isomorphism between the underlying graphs of URG structures $\mathfrak{S}_A$ and $\mathfrak{S}_B$. Then A and B are equivalent.*

Proof. We show that the *relational* inverse f^{-1} of f is also an inverse in the ordinary sense. Let $a : A$. We need to prove that $f^{-1}(f\ a) = a$. By assumption that f is a relational isomorphism we have a $p : f^{-1}(f\ a) \cong a$. The relation $\cong$ on A is univalent, so there is a map $U : f^{-1}(f\ a) \cong a \to f^{-1}(f\ a) = a$. Clearly, $U\,p : f^{-1}(f\ a) = a$. A symmetric argument proves that f^{-1} is also a right inverse. □

Assume now that there are types and maps as follows:

$$\begin{tikzcd} \mathsf{Type} & \mathsf{Type} \\ A \arrow[u, "B"] \arrow[r, "f"] & A' \arrow[u, "B'"'] \end{tikzcd} \tag{3.3}$$

We may consider the type family B' to be defined over A, by precomposing with f, i.e., we define

$$f^*B' :\equiv \lambda\, a \mapsto B'(f\ a).$$

To construct an equivalence of the types $(a : A) \times B\,a$ and $(a : A) \times f^*B\,a$, it suffices to give a fiberwise equivalence

$$g : (a : A) \to (B\,a) \simeq (f^*B\,a).$$

If f is an equivalence, then f and g together induce an equivalence

$$(a : A) \times B\,a \simeq (a' : A') \times B'\,a \tag{3.4}$$

between the original total types. Compared to constructing an equivalence as in (3.4) directly, the advantage of this method is that we merely need to set up tools for proving that maps are inverse to each other in type families over A. This concept, of course, transfers to DURG structures (Theorem 3.25).

Definition 3.24. Let $f : A \to A'$ be a function, $\mathfrak{S}_A$ and $\mathfrak{S}_{A'}$ URG structures on A and A', and $\mathfrak{D}^B_A$, and $\mathfrak{D}^{B'}_{A'}$ be DURG structures over $\mathfrak{S}_A$ and $\mathfrak{S}_{A'}$, respectively. A *fiberwise relational isomorphism of* $\mathfrak{D}^B_A$ *and* $\mathfrak{D}^{B'}_{A'}$ *over* f is a function of type

$$(a : A) \to \mathrm{RelIso}\,\langle B\,a, \cong_{\rho\,a}\rangle\,\langle f^*B'\,a, \cong'_{\rho(f\,a)}\rangle.$$

Here $\cong$ and $\cong'$ are the displayed relations of $\mathfrak{D}^B_A$ and $\mathfrak{D}^{B'}_{A'}$ respectively.

Note that f^*B' a priori does not carry a DURG structure over $\mathfrak{S}_A$; it is merely a type family with a binary relation on every fiber. Direct application of univalence of the relation of $\mathfrak{D}^{B'}_{A'}$ suffices to make the total equivalence work.

Theorem 3.25. *Let* $f, \mathfrak{S}_A, \mathfrak{S}_{A'}, \mathfrak{D}^B_A$ *and* $\mathfrak{D}^{B'}_{A'}$ *be as in Definition 3.24. Assume furthermore that* f *is an equivalence, and that there is a fiberwise relational isomorphism between* $\mathfrak{D}^B_A$ *and* $\mathfrak{D}^{B'}_{A'}$ *over* f*. Then the total spaces* $(a : A) \times B\,a$ *and* $(a' : A') \times B'\,a'$ *are equivalent.*

We visualize the input to Theorem 3.25 via this diagram:

$$\begin{tikzcd}
B \arrow[d] & B' \arrow[d] \arrow[dl, dashed] \\
A \arrow[r, "\sim"] & A'
\end{tikzcd}$$

The dashed arrow indicates that B' is pulled back along the equivalence, but not viewed as a displayed URG over $\mathfrak{S}_A$.

Proof. In the same way as in the proof of Proposition 3.23, we get an equivalence $B\,a \simeq f^*B'\,a$ for every $a : A$. We can apply Proposition 2.11 to bundle up f and the fiberwise equivalence to get an equivalence of the desired type. □

Consider the special case where $A \equiv A'$ and $\mathfrak{S}_A \equiv \mathfrak{S}_{A'}$. Then we take f to be the identity equivalence on A, and no pullback is necessary. Note that Proposition 3.23 can be seen as a special case of Theorem 3.25 when $A \equiv \mathbf{1} \equiv A'$.

Alternatively, consider the case where B and B' are propositional families. Then, to construct a fiberwise relational isomorphism between $\mathfrak{D}^B_A$ and $\mathfrak{D}^{B'}_{A'}$ over f, it suffices to give fiberwise *functions*

$$(a : A) \to B\,a \to B'(f\ a)$$

and

$$(a : A) \to B'(f\ a) \to B\, a.$$

If $\mathfrak{D}_A^B$ and $\mathfrak{D}_{A'}^{B'}$ were constructed using Proposition 3.14 this reduces the type-checking time,[5] since the displayed relation on either side is given by $\mathbf{1}$, rather than an arbitrary contractible type, allowing to choose $*$ for the center of contraction every time.

[5] Agda's abstract keyword nullifies this advantage; see Chapter 6.

4 Equivalence of Strict 2-Groups and Crossed Modules

The first two sections of this chapter introduce strict 2-groups and crossed modules. In the three subsequent sections DURG structures are used to show the equivalence between group actions and split monomorphisms, precrossed modules and internal reflexive graphs in the category of groups, crossed modules and Peiffer reflexive graphs, and Peiffer reflexive graphs and strict 2-groups.

4.1 Strict 2-Groups

In the introduction we stated that a 2-group is a group object in the category of groupoids. Such a group object consists of a groupoid $\mathscr{G}$ together with morphisms of groupoids as follows.

$$\begin{tikzcd} * \arrow[r, "1"] & \mathscr{G} \arrow[loop above, "-1"] & \mathscr{G} \times \mathscr{G} \arrow[l, "\cdot"'] \end{tikzcd} \tag{4.1}$$

dc A groupoid can be seen as an ordinary category in which every morphism is an isomorphism. From that point of view, morphisms of groupoids are functors. These functors also have to satisfy the obvious inverse, unit and associativity laws.

We may prepend the word ‘strict’ to this definition and obtain the more rigid *strict* 2-groups – internal groups in the 1-category of groupoids with the functors in (4.1) satisfying the coherence laws up to equality, not just natural isomorphism.

We would like to work with an alternative, equivalent definition of strict 2-groups, namely that of internal categories in the category of groups. Such an internal category consists of groups G_0 and G_1, together with group homomorphisms arranged as follows:

$$\begin{tikzcd}
G_1 \times_{G_0} G_1 \arrow[r, "\circ"] & G_1 \arrow[r, shift left=2, "\sigma"] \arrow[r, shift right=2, "\tau"'] & G_0 \arrow[l, "\iota" description]
\end{tikzcd}$$

Here the pullback $G_1 \times_{G_0} G_1$ is the limit of the cospan $\sigma : G_1 \to G_0 \leftarrow G_1 : \tau$.[1] The morphisms σ, τ, ι and $\circ$ are the *source*, *target*, *identity-assigning* and *composition* morphisms, respectively. They are subject to the coherence laws expressed as commutativity of the following diagrams.

- The source and target of identity morphisms are as expected.

$$\begin{tikzcd}
G_0 \arrow[r, "\iota"] \arrow[dr, "1"'] & G_1 \arrow[d, "\sigma"] \\
& G_0
\end{tikzcd}
\qquad
\begin{tikzcd}
G_0 \arrow[r, "\iota"] \arrow[dr, "1"'] & G_1 \arrow[d, "\tau"] \\
& G_0
\end{tikzcd}$$

- The source and target of composite morphisms behave as expected.

$$\begin{tikzcd}
G_1 \times_{G_0} G_1 \arrow[r, "\circ"] \arrow[d, "\pi_2"'] & G_1 \arrow[d, "\sigma"] \\
G_1 \arrow[r, "\sigma"'] & G_0
\end{tikzcd}
\qquad
\begin{tikzcd}
G_1 \times_{G_0} G_1 \arrow[r, "\circ"] \arrow[d, "\pi_1"'] & G_1 \arrow[d, "\tau"] \\
G_1 \arrow[r, "\tau"'] & G_0
\end{tikzcd}$$

- Composition is associative.

$$\begin{tikzcd}
G_1 \times_{G_0} G_1 \times_{G_0} G_1 \arrow[r, "\langle 1{,}\circ \rangle"] \arrow[d, "\langle \circ{,}1 \rangle"'] & G_1 \times_{G_0} G_1 \arrow[d, "\circ"] \\
G_1 \times_{G_0} G_1 \arrow[r, "\circ"'] & G_0
\end{tikzcd}$$

- Composition satisfies left and right unit laws.

$$\begin{tikzcd}
G_0 \times_{G_0} G_1 \arrow[r, "\langle \iota{,}1 \rangle"] \arrow[dr, "\pi_2"'] & G_1 \times_{G_0} G_1 \arrow[d, "\circ"] & G_1 \times_{G_0} G_0 \arrow[l, "\langle 1{,}\iota \rangle"'] \arrow[dl, "\pi_1"] \\
& G_1 &
\end{tikzcd}$$

[1] This way, $\circ$ composes like function composition, not sequential composition.

In what way are internal groups in the category of groupoids and internal categories in the category of groups equivalent? G_0 corresponds to the objects of $\mathscr{G}$, G_1 to the morphisms of $\mathscr{G}$. The composition operation $\circ$ on $G_1 \times_{G_0} G_1$ corresponds to the composition of arrows in the morphisms of $\mathscr{G}$. The group operation of G_0 is represented by the map $\cdot$ on the object level; that of G_1 by the map $\cdot$ on the morphism level. The correspondence between the remaining structure should be clear.

Example 4.1. Every group G gives a strict 2-group. We take $G_0 :\equiv G$ and G_1 to be the minimal set of morphisms, namely only identity morphisms for all objects of G.

A third perspective unites the previous approaches. A strict 2-group can be seen as a very special 2-category, namely a connected, strict 2-groupoid on one object. The groups G_0 and G_1 are then the 1, and 2-arrows, respectively.

As for all 2-categories, computations in a strict 2-group can be drawn using *pasting diagrams*. In such a diagram we use $*$ to resemble the generic point, '$\rightarrow$' for 1-arrows or elements of G_0 and '$\Rightarrow$' for 2-arrows.

Let us visualize the homomorphism property of $\circ$. Suppose we have composable $a\,c : G_1$ and $b\,d : G_1$. We can first form the products $a \cdot_1 b$ and $c \cdot_1 d$ in G_1. These will again be composable, becacuse σ and τ are homomorphisms. The pasting diagram of $(c \cdot_1 d) \circ (a \cdot_1 b)$ is:

$$\begin{tikzcd}
{*} \arrow[r, bend left=60, ""{name=A1, below}] \arrow[r, "f"', ""{name=A2}] & {*} \arrow[r, bend left=60, ""{name=B1, below}] \arrow[r, "g"', ""{name=B2}] & {*} \\
 & \circ & \\
{*} \arrow[r, "f", ""{name=C1, below}] \arrow[r, bend right=60, ""{name=C2}] & {*} \arrow[r, "g", ""{name=D1, below}] \arrow[r, bend right=60, ""{name=D2}] & {*}
\arrow[Rightarrow, from=A1, to=A2, "a"]
\arrow[Rightarrow, from=B1, to=B2, "b"]
\arrow[Rightarrow, from=C1, to=C2, "c"]
\arrow[Rightarrow, from=D1, to=D2, "d"]
\end{tikzcd} \tag{4.2}$$

The map $\circ$ being a homomorphism means exactly that (4.2) is equal to first composing using $\circ$ and then forming the product in G_1. The corresponding picture is:

$$\begin{tikzcd}
{*} \arrow[r, bend left=60, ""{name=A1, below}] \arrow[r, "f", ""{name=A2}, ""{name=A3, below}] \arrow[r, bend right=60, ""{name=A4}] & {*} & \cdot_1 & {*} \arrow[r, bend left=60, ""{name=B1, below}] \arrow[r, "g", ""{name=B2}, ""{name=B3, below}] \arrow[r, bend right=60, ""{name=B4}] & {*}
\arrow[Rightarrow, from=A1, to=A2, "a"]
\arrow[Rightarrow, from=A3, to=A4, "c"]
\arrow[Rightarrow, from=B1, to=B2, "b"]
\arrow[Rightarrow, from=B3, to=B4, "d"]
\end{tikzcd}$$

Because of this convention, $\circ$ is often called *vertical composition*, the group operation of G_1 *horizontal composition* and the homomorphism property of $\circ$ *interchange law*.

4.2 Crossed Modules

Definition 4.2. A *crossed module* consists of groups G and H, a homomorphism $\varphi : H \to G$ and an action $\alpha : G \times H \to H$ such that the diagrams

$$\begin{tikzcd}
G \times H \arrow[r, "\alpha"] \arrow[d, "\mathrm{id}_G \times \varphi"'] & H \arrow[d, "\varphi"] \\
G \times G \arrow[r, "\mathrm{Ad}"'] & G
\end{tikzcd}
\qquad
\begin{tikzcd}
H \times H \arrow[d, "\varphi \times \mathrm{id}_H"'] \arrow[dr, "\mathrm{Ad}"] & \\
G \times H \arrow[r, "\alpha"'] & H
\end{tikzcd}$$

commute. Here Ad denotes the adjoint action. Commutativitiy of the first diagram translates to G-equivariance of φ, i.e.,

$$\varphi(g \,\alpha\, h) = g(\varphi\, h)g^{-1}.$$

The second diagram signifies that the action α has to satisfy the so-called *Peiffer identity*

$$(\varphi\, h) \,\alpha\, h' = hh'h^{-1}$$

for all $h, h' \in H$.

We remark that the group H can act on itself in two different ways. One is the usual conjugation action $h \cdot h' :\equiv hh'h^{-1}$; the second one is by mapping h down to G and then using the action α of G on H. The Peiffer rule states that these two actions coincide.

Another way to interpret the Peiffer identity is to rewrite it to

$$hh' = ((\varphi\, h) \,\alpha\, h')h.$$

Then it looks like a 'twisted commutativity law' for H. Indeed, if H is any abelian group, then there is a unique crossed module structure on $G :\equiv \{e\}$ and H.

Example 4.3. Every group G gives a crossed module

$$\begin{tikzcd}
\mathrm{G} \arrow[r, dashed, shift left, "!"] & \{e\} \arrow[l, hook, shift left, "!"]
\end{tikzcd} .$$

We use the dashed arrow to indicate an action. The exclamation marks express that there is only one possible choice for the object at hand: the trivial action and the trivial homomorphism, respectively.

More generally, if H is any normal subgroup of G, then the inclusion homomorphism $H \hookrightarrow G$ together with the conjugation action Ad give a crossed module

$$\begin{tikzcd}
G \arrow[r, dashed, shift left, "\mathrm{Ad}"] & H \arrow[l, hook', shift left] .
\end{tikzcd}$$

In that sense, crossed modules generalize normal subgroups.

Crossed modules encode strict 2-groups in terms of two ordinary groups together with additional structure and properties. A crossed module defines a strict 2-group via the right inclusion[2] and projection of the semidirect product:

$$\begin{tikzcd}
G_0 \arrow[r, dashed, bend left, "\alpha"] & H \arrow[l, "\varphi"] \arrow[r, mapsto] & G_0 \arrow[r, "\iota_2" description] & H \rtimes_\alpha G_0 \arrow[l, shift right=3, "\pi_2"'] \arrow[l, shift left=3, "\tau_\varphi"]
\end{tikzcd}$$

Conversely, a strict 2-group produces an action of G_0 on the group of 2-arrows with source the neutral element of G_1:

$$\begin{tikzcd}
G_0 \arrow[r, "\iota" description] \arrow[rr, dashed, bend left=60, "\mathrm{Ad}_{\iota_2}"] & G_1 \arrow[l, shift right=3, "\sigma"'] \arrow[l, shift left=3, "\tau"] & \ker\sigma \arrow[l, hook', "\iota'"'] \arrow[ll, bend left=60, "\tau \circ_{\mathrm{Grp}} \iota'"] & G_0 \arrow[l, mapsto] \arrow[r, "\iota" description] & G_1 \arrow[l, shift right=3, "\sigma"'] \arrow[l, shift left=3, "\tau"]
\end{tikzcd}$$

The rest of the chapter is devoted to proving that these maps actually define an equivalence of types.

4.3 Group Actions and Split Monomorphisms

In this section we establish the equivalence between actions and split monomorphisms. With the SIP for groups already proved, we can take the URG structure $\mathfrak{S}_{\mathrm{Grp}}$ for the bottom

[2] The reason we draw the second projection on the left is the following: G_0 stays constant when going back and forth along the equivalence of strict 2-groups, so it has to be at the bottom of the algebraic hierarchy. Hence, the two groups involved appear as $\langle G_0, G_1 \rangle$, not the other way around. Consequently, ι is directed forwards, and σ and τ backwards.

level. By the URG product $\times_{\mathfrak{S}}$, this gives rise to a URG structure on the type of pairs of groups. On top of pairs of groups, there are two towers. One adds a left action structure (LAS) followed by the axioms turning it into an action. The other successively adds a morphism forth (F) and back (B), and afterwards the split condition.

$$\begin{tikzcd}
\text{isAction} \arrow[d] & & \text{isSecRet} \arrow[d] \\
\text{LAS} \arrow[dr] & & \mathrm{F} \times \mathrm{B} \arrow[dl] \\
& \text{Grp} \arrow[d] & \\
& \text{Grp} &
\end{tikzcd}$$

Groups

We define

$$\mathfrak{S}_{\mathrm{Grp}^2} :\equiv \mathfrak{S}_{\mathrm{Grp}} \times_{\mathfrak{S}} \mathfrak{S}_{\mathrm{Grp}},$$

the product URG structure on the type of pairs of groups.

Proposition 4.4. *Over $\mathfrak{S}_{\mathrm{Grp}^2}$ we can display:*

1. *homomorphisms in the forth direction, i.e., there is a DURG structure $\mathfrak{D}^{\mathrm{F}}_{\mathrm{Grp}^2}$ on the type family*

$$\lambda \langle G, H \rangle \mapsto G \rightarrow_{\mathrm{Grp}} H;$$

2. *homomorphisms in the back direction, i.e., there is a DURG structure $\mathfrak{D}^{\mathrm{B}}_{\mathrm{Grp}^2}$ on the type family*

$$\lambda \langle G, H \rangle \mapsto H \rightarrow_{\mathrm{Grp}} G;$$

3. *and pairs of homomorphisms back and forth, i.e., there is a DURG structure $\mathfrak{D}^{\mathrm{F}\times\mathrm{B}}_{\mathrm{Grp}^2}$ on the type family*

$$\lambda \langle G, H \rangle \mapsto (G \rightarrow_{\mathrm{Grp}} H) \times (H \rightarrow_{\mathrm{Grp}} G).$$

Proof. 1. The displayed relation assumes a context of groups and homomorphisms arranged as follows:

$$\begin{tikzcd}
G \arrow[r, "f"] \arrow[d, "p"', "\wr"] & H \arrow[d, "q", "\wr"'] \\
G' \arrow[r, "f'"'] & H'
\end{tikzcd}$$

Here p and q are group isomorphisms. We define the displayed relation of f and f' over $\langle p, q \rangle$ to be commutativity of the above diagram. Explicitly,

$$f \cong_{\langle p,q \rangle} f' :\equiv (g : G) \to q(f\ g) = f'(p\ g). \tag{4.3}$$

Next, we need to show that (4.3) is reflexive w.r.t. the reflexivity term of the URG structure on pairs of groups. That reflexivity term is simply the pair $\langle \mathrm{id}_G^{\mathrm{Grp}}, \mathrm{id}_H^{\mathrm{Grp}} \rangle$ of identity homomorphisms. Hence, given $f : G \to_{\mathrm{Grp}} H$ and $g : G$, condition (4.3) becomes $f\ g = f\ g$. This is solved by refl.

By Theorem 3.10, contractibility of all relational singletons of the relation (4.3) implies its univalence. Let $f : G \to_{\mathrm{Grp}} H$ be given. We need to prove that

$$(f' : G \to_{\mathrm{Grp}} H) \times f \cong_{\langle \mathrm{id}_G^{\mathrm{Grp}}, \mathrm{id}_H^{\mathrm{Grp}} \rangle} f' \tag{4.4}$$

is contractible. Extensionality for homomorphisms (Proposition 2.30) implies that that (4.4) is equivalent to

$$(f' : G \to_{\mathrm{Grp}} H) \times (f = f'),$$

which is the singleton at f. The claim follows because singletons are contractible and equivalences preserve truncation levels.

2. The situation is entirely symmetric to that of the previous case, so we omit an explicit proof.

3. By the above, we have morphisms forth and back displayed over pairs of groups. We apply Corollary 3.21 to combine $\mathfrak{D}^{\mathrm{F}}_{\mathrm{Grp}^2}$ and $\mathfrak{D}^{\mathrm{B}}_{\mathrm{Grp}^2}$. This gives a DURG structure $\mathfrak{D}^{\mathrm{F} \times \mathrm{B}}_{\mathrm{Grp}^2}$ on

$$\lambda\, \langle G, H \rangle \mapsto (G \to_{\mathrm{Grp}} H) \times (H \to_{\mathrm{Grp}} G). \qquad \square$$

It should be noted that in general any DURG structure over $\mathfrak{S}_A \times_{\mathfrak{S}} \mathfrak{S}_A$ gives rise to a symmetric one by transporting along the swap equivalence. However, this cannot be used to obtain the second case from the first one in Proposition 4.4 without further adjustments, because we allow the two factors, both called Grp, to live in different universes.

We call the URG structure on the total space consisting of pairs of homomorphisms forth and back

$$\mathfrak{S}_{\mathrm{Grp}^2 \times \mathrm{F} \times \mathrm{B}} :\equiv \int \mathfrak{D}^{\mathrm{F} \times \mathrm{B}}_{\mathrm{Grp}^2}.$$

Lemma 4.5. *The split condition can be displayed over $\mathfrak{S}_{\mathrm{Grp}^2 \times \mathrm{F} \times \mathrm{B}}$, i.e., there is a DURG structure $\mathfrak{D}^{\mathrm{isSecRet}}_{\mathrm{Grp}^2 \times \mathrm{F} \times \mathrm{B}}$ on the type family*

$$\lambda \, \langle \langle G_0, G_1 \rangle, \iota, \sigma \rangle \mapsto \mathrm{isSecRet}\langle \iota, \sigma \rangle.^{3}$$

Proof. The type $\mathrm{isSecRet}\langle \iota, \sigma \rangle$ is a proposition. Hence, Proposition 3.14 immediately produces the desired DURG structure. □

Taking the total space produces a URG structure

$$\mathfrak{S}_{\mathrm{SplitMono}} :\equiv \int \mathfrak{D}^{\mathrm{isSecRet}}_{\mathrm{Grp}^2 \times \mathrm{F} \times \mathrm{B}}$$

on the Σ-type SplitMono consisting of tuples $\langle \langle G_0, G_1 \rangle, \iota, \sigma \rangle$ such that $\mathrm{isSecRet}\langle \iota, \sigma \rangle$.

Actions

To display actions over pairs of groups we first add a left action structure and in a second step the action axioms.

Lemma 4.6. *Left action structures can be displayed over pairs of groups, i.e., there is a DURG structure $\mathfrak{D}^{\mathrm{LAS}}_{\mathrm{Grp}^2}$ on the type family*

$$\lambda \, \langle G, H \rangle \mapsto G \to H \to H.$$

[3] When forming new type families over an existing nested Σ-type, we keep track of how the elements are associated, because sometimes the families need to be reassociated later on.

Proof. This proof is similar to that of Proposition 4.4. The displayed relation assumes (after uncurrying) a context of (underlying types of) groups G and H, maps α and β, and equivalences p and q, as arranged in this diagram:

$$\begin{tikzcd}
G \times H \arrow[r, "\alpha"] \arrow[d, "\langle p,q\rangle"', "\wr"] & H \arrow[d, "q", "\wr"'] \\
G' \times H' \arrow[r, "\beta"'] & H'
\end{tikzcd}$$

The displayed relation of α and β is defined to be commutativity of the above diagram. Explicitly,

$$\alpha \cong_{\langle p,q\rangle} \beta \ :\equiv\ (g : G) \to (h : H) \to q(g\ \alpha\ h) = (p\ g)\ \beta\ (q\ h).$$

Reflexivity is immediate and two applications of function extensionality transform the relational singleton at α to the ordinary singleton at α. □

We give the name

$$\mathfrak{S}_{\mathrm{Grp}^2 \times \mathrm{LAS}} :\equiv \int \mathfrak{D}^{\mathrm{LAS}}_{\mathrm{Grp}^2}$$

to the resulting URG structure on the type of pairs of groups with a left action structure.

Lemma 4.7. *The action axioms can be displayed over* $\mathfrak{S}_{\mathrm{Grp}^2 \times \mathrm{LAS}}$*, i.e., there is a DURG structure* $\mathfrak{D}^{\mathrm{isAction}}_{\mathrm{Grp}^2 \times \mathrm{LAS}}$ *on the type family*

$$\lambda\, \langle\langle G, H\rangle, \alpha\rangle \ \mapsto \mathrm{isAction}\, \alpha.$$

Proof. The type isAction α is a proposition. Proposition 3.14 produces the desired DURG structure. □

Equivalence

If α is an action of G_0 on H, a split monomorphism can be obtained by constructing the semidirect product with respect to α and showing that its second projection has a section.

Conversely, a section-retraction-pair $\iota : G_0 \leftrightarrow H : \sigma$ induces a conjugation action of G_0 on $\ker\sigma$ by precomposing with ι.

$$\begin{tikzcd}
 & G_0 \arrow[r, dashed, "\alpha"] & H & {} \arrow[r, mapsto, "\text{Proposition 4.8}"] & {} & G_0 \arrow[r, shift right, "\iota_2"'] & H \rtimes_\alpha G_0 \arrow[l, shift right, "\pi_2"'] \\
G_0 \arrow[r, shift right, "\iota"'] \arrow[rr, dashed, bend left, "\mathrm{Ad}_\iota"] & G_1 \arrow[l, shift right, "\sigma"'] & \ker\sigma \arrow[l, hook'] & {} & {} \arrow[l, mapsto, "\text{Proposition 4.9}"'] & G_0 \arrow[r, shift right, "\iota"'] & G_1 \arrow[l, shift right, "\sigma"']
\end{tikzcd}$$

Proposition 4.8. *Let $\langle G_0, H, \alpha, \mathrm{isAct}\rangle$[4] be an action of G_0 on H. Then the second inclusion and projection of the semidirect product $H \rtimes_\alpha G_0$ form a section-retraction-pair.*

Proof. The second projection is the homomorphism

$$\pi_2 : (H \rtimes_\alpha G_0) \to_{\mathrm{Grp}} G_0,$$
$$\pi_2 = \langle \lambda\, \langle h, g\rangle \mapsto g, \lambda\, \langle h, g\rangle\, \langle h', g'\rangle \mapsto \mathrm{refl}_{gg'}\rangle$$

and the second inclusion is the obvious homomorphism $\iota_2 : G_0 \to_{\mathrm{Grp}} H \rtimes_\alpha G_0$. The homomorphism ι_2 consists of the map $\lambda\, g \mapsto \langle 1, g\rangle$ and a term witnessing that

$$\langle 1, gg'\rangle = \langle 1(g\, \alpha\, 1), gg'\rangle$$

for any $g\, g' : G_0$.

It is clear that $\pi_2 \circ \iota_2 \sim \mathrm{id}_{G_0}$. Proposition 2.30 about group morphism extensionality yields a path

$$\pi_2 \circ_{\mathrm{Grp}} \iota_2 = \mathrm{id}^{\mathrm{Grp}}_{G_0}. \qquad \square$$

Proposition 4.9. *Let $\langle G_0, G_1, \iota, \sigma, \mathrm{split}^\sigma_\iota\rangle$ be a split monomorphism with retraction σ. Then G_0 acts on* $\ker\sigma$ *by conjugation after ι.*

Proof. We define the left action structure of G_0 on $\ker\sigma$ by

$$g\ \alpha\ \langle h, p\rangle = \langle (\iota\, g)h(\iota\, g^{-1}), q\rangle$$

[4] To keep the amount of parentheses manageable, the ordering of pairs is suppressed when we prove a lemma about a composite Σ-type.

Here q is a proof that $\sigma((\iota\, g)h(\iota\, g^{-1})) = 1$. The term q can be constructed using the group axioms, homomorphism properties of σ, ι, and $p : \sigma\, h = 1$.

We prove that α satisfies the action axioms.

1. Let $g : G_0$. We show that $\lambda\, h \mapsto (g\ \alpha\, h)$ is a homomorphism. By Lemma 2.19 it suffices to show that

$$(\iota\, g)hh'(\iota g^{-1}) = (\iota\, g)h(\iota\, g^{-1})(\iota\, g)h'(\iota g^{-1}),$$

   whenever there are p and p' such that $\langle h, p\rangle$ and $\langle h', p'\rangle$ are terms in $\ker \sigma$. This, however, amounts to no more than a sequence of applications of associativity and cancellation in H, so we skip the details.

2. Again, Lemma 2.19 reduces the identity axiom for actions to showing that for any $\langle h, p\rangle : \ker \sigma$ the identity

$$(1\ \alpha\ \langle h, p\rangle)_1 = h$$

   holds. This is done using that homomorphisms preserve the identity, together with a sequence of cancellations and identity laws in G_1.

3. By the same argument we skip this simple proof of associativity of α. □

As we see, the group G_0 remains constant when going back and forth. This suggests to apply Theorem 3.19 twice to $\mathfrak{D}^{\text{isAction}}_{\text{Grp}^2 \times \text{LAS}}$ to obtain $\mathfrak{D}^{\text{Grp} \times \text{LAS} \times \text{isAction}}_{\text{Grp}}$ – a DURG structure on the type family

$$\lambda\, (G_0 : \text{Grp}) \mapsto (H : \text{Grp}) \times (\alpha : G_0 \to H \to H) \times \text{isAction}\, \alpha.$$

In the same fashion we obtain a DURG structure $\mathfrak{D}^{\text{Grp} \times (\text{F} \times \text{B}) \times \text{isSecRet}}_{\text{Grp}}$ on

$$\lambda\, (G_0 : \text{Grp}) \mapsto (G_1 : \text{Grp}) \times (\langle \iota, \sigma\rangle : (G_0 \to_{\text{Grp}} G_1) \times (G_1 \to_{\text{Grp}} G_0)) \times \text{isSecRet}\langle \iota, \sigma\rangle.$$

Lemma 4.10. *There is a fiberwise relational isomorphism between* $\mathfrak{D}^{\text{Grp} \times \text{LAS} \times \text{isAction}}_{\text{Grp}}$ *and* $\mathfrak{D}^{\text{Grp} \times (\text{F} \times \text{B}) \times \text{isSecRet}}_{\text{Grp}}$ *over the type of groups.*

Proof. Fix a group G_0. We show that the maps constructed in Proposition 4.8 and Proposition 4.9 are relationally inverse to each other.

Claim: Let $\langle H, \alpha, \mathrm{isAct}\rangle$ be an action of G_0 on H. Then there is $\varphi : \ker \pi_2 \simeq_{\mathrm{Grp}} H$ such that

$$\begin{tikzcd}
G_0 \times \ker \sigma \arrow[r, "\mathrm{Ad}_{\iota_2}"] \arrow[d, "{\langle \mathrm{id}, \varphi\rangle}"'] & \ker \pi_2 \arrow[d, "\varphi"] \\
G_0 \times H \arrow[r, "\alpha"'] & H
\end{tikzcd}$$

commutes.

The map φ extracts the H-component of $\ker \pi_2$, that is

$$\varphi \langle\langle h, g\rangle, p\rangle :\equiv h.$$

Consider the function

$$\psi :\equiv \lambda (h : H) \mapsto \langle\langle h, 1\rangle, \mathrm{refl}\rangle.$$

We prove that ψ is a pseudo-inverse of φ. Clearly, $\varphi \circ \psi \sim \mathrm{id}_H$. To see that ψ is also a left inverse to φ, let $\langle\langle h, g\rangle, p\rangle : \ker \pi_2$. It is necessary to construct a path of type

$$\langle\langle h, 0\rangle, \mathrm{refl}\rangle = \langle\langle h, g\rangle, p\rangle.$$

This can be done component-wise. Reflexivity provides a path for the first component, and a path over refl is just an ordinary path. Hence, we use $p^{-1} : 1 = g$ for the second component. Being an element of the kernel is a proposition, so we can apply Lemma 2.18 to obtain a path for the last component over $\langle \mathrm{refl}, p^{-1}\rangle$.

To see that φ is a homomorphism, let $\langle\langle h, g\rangle, p\rangle$ and $\langle\langle h, g\rangle, p\rangle$ be in $\ker \pi_2$. The desired path of type

$$h(g \, \alpha \, h') = hh'$$

can be constructed by first reducing g to 1 using p, and then applying the identity axiom for actions.

Commutativity of the square is equivalent to pure existence of a path

$$q : 1(g \, \alpha \, h) = (gg') \, \alpha \, (g^{-1} \, \alpha \, 1^{-1})$$

for every $g : G_0$ and $\langle\langle h, g'\rangle, p\rangle : \ker \sigma$. Such a term q can be constructed using multiple applications of identity laws for groups, as well as the homomorphism axiom for actions.

This finishes the proof of the first claim.

Claim: Conversely, let $\langle G_1, \iota, \sigma, \mathrm{split}_\iota^\sigma \rangle$ be a split monomorphism with domain G_0. There is an isomorphism of groups φ such that both diagrams

$$\begin{tikzcd}
\ker\sigma \rtimes_{\mathrm{Ad}_\iota} G_0 \arrow[r, "\pi_2"] \arrow[d, "\varphi"'] & G_0 \arrow[d, "\mathrm{id}"] & G_0 \arrow[r, "\iota_2"] \arrow[d, "\mathrm{id}"'] & \ker\sigma \rtimes_{\mathrm{Ad}_\iota} G_0 \arrow[d, "\varphi"] \\
G_1 \arrow[r, "\sigma"'] & G_0 & G_0 \arrow[r, "\iota"'] & G_1
\end{tikzcd}$$

commute.

We define the map

$$\varphi \, \langle \langle h, g \rangle, p \rangle \, :\equiv \, h \, \iota \, g.$$

A pseudo-inverse of φ is given by

$$\psi \, :\equiv \, \lambda \, h \, \mapsto \langle \langle h \, \iota(\sigma \, h^{-1})), p \rangle, \sigma \, h \rangle.$$

Here, p is any proof that $\sigma(h \, \iota(\sigma \, h^{-1})) = 1$. It can be easily verified that ψ actually is a pseudo-inverse. This crucially uses $\mathrm{split}_\iota^\sigma$. We refer the doubtful reader to the formalization.

To prove that the above square commutes is to give paths

$$g = \sigma(h \, \iota \, g),$$

and

$$1 \, \iota \, g = \iota \, g,$$

for all $g : G_0$ and $\langle h, p \rangle : \ker\sigma$. Since $p : \sigma \, h = 1$, this is obvious.

This finishes the proof of the second claim. □

Theorem 4.11. *There is an equivalence*

$$\mathrm{Action} \simeq \mathrm{SplitMono}\,.$$

Proof. Lemma 4.10 and Theorem 3.25 establish an equivalence between the types in question up to associating the Σ-types, and associating Σ-types is an equivalence. □

4.4 Precrossed Modules and Internal Reflexive Graphs

A precrossed module consists of an action with an additional morphism (in the back direction), which is equivariant w.r.t. that action. An internal reflexive graph is a split monomorphism together with a second retraction. We construct an equivalence of the types of these structures by pulling back along the equivalence between actions and split monomorphisms from the previous section. Pictorially, we are in this situation:

$$\begin{tikzcd}
\mathrm{isEquivariant} \arrow[d] & & & & & \mathrm{isSecRet} \arrow[d] \\
\mathrm{B} \arrow[drr] & & \mathrm{B} \times \mathrm{isEquivariant} \arrow[d] & \mathrm{B} \times \mathrm{isSecRet} \arrow[d] \arrow[dl, dashed] & & \mathrm{B} \arrow[dll] \\
& & \mathrm{Action} \arrow[r, "\sim"] & \mathrm{SplitMono} & &
\end{tikzcd}$$

Precrossed Modules

Lemma 4.12. *Group homomorphisms can be displayed over actions, i.e., there is a DURG structure* $\mathfrak{D}^{\mathrm{B}}_{\mathrm{Action}}$ *on the type family*

$$\lambda \, \langle\langle\langle G_0, H\rangle, \alpha\rangle, \mathrm{isAct}\rangle \mapsto H \to_{\mathrm{Grp}} G_0$$

over Action.

Proof. We apply the lifting operation from Proposition 3.20 once to $\mathfrak{D}^{\mathrm{B}}_{\mathrm{Grp}^2}$, to obtain $\mathfrak{D}^{\mathrm{B}}_{\mathrm{Grp}^2 \times \mathrm{LAS}}$. We apply it again and have $\mathfrak{D}^{\mathrm{B}}_{\mathrm{Action}}$. □

We define the URG structure

$$\mathfrak{S}_{\mathrm{Action} \times \mathrm{B}} :\equiv \int \mathfrak{D}^{\mathrm{B}}_{\mathrm{Action}}$$

on the Σ-type Action × B with components an action and an additional homomorphism.

Definition 4.13. Let $\langle G_0, H, \alpha, \mathrm{isAct}, \varphi\rangle : \mathrm{Action} \times \mathrm{B}$ be an action with a homomorphism $\varphi : H \to_{\mathrm{Grp}} G_0$. We say that φ is *equivariant* w.r.t. α, if there is a term of type

$$\mathrm{isEquivariant}\langle \alpha, \varphi\rangle :\equiv (g : G_0) \to (h : H) \to \varphi(g\, \alpha\, h) = g(\varphi\, h)g^{-1}.$$

We define the *type of precrossed modules* to be

$$\mathrm{PreXModule} :\equiv (\langle\langle\langle\langle G_0, H\rangle, \alpha\rangle, \mathrm{isAct}\rangle, \varphi\rangle : \mathrm{Action} \times \mathrm{B}) \times \mathrm{isEquivariant}\langle \alpha, \varphi\rangle.$$

The type $\mathrm{isEquivariant}\langle \alpha, \varphi\rangle$ is a proposition, because H is a set. Hence, the family of equivariance proofs over Action × B carries a DURG structure $\mathfrak{D}^{\mathrm{isEquivariant}}_{\mathrm{Action} \times \mathrm{B}}$. As usual, we take the total space

$$\mathfrak{S}_{\mathrm{PreXModule}} :\equiv \int \mathfrak{D}^{\mathrm{isEquivariant}}_{\mathrm{Action} \times \mathrm{B}}.$$

Internal Reflexive Graphs in the Category of Groups

Lemma 4.14. *Morphisms in the back direction can be displayed over split monomorphisms, i.e., there is a DURG structure $\mathfrak{D}^{\mathrm{B}}_{\mathrm{SplitMono}}$ on the type family*

$$\lambda\, (\langle\langle\langle G_0, G_1\rangle, \langle \iota, \sigma\rangle\rangle, \mathrm{split}^{\sigma}_{\iota}\rangle : \mathrm{SplitMono}) \mapsto G_1 \to_{\mathrm{Grp}} G_0.$$

Proof. We lift (Proposition 3.20) the structure $\mathfrak{D}^{\mathrm{B}}_{\mathrm{Grp}^2}$ once to obtain $\mathfrak{D}^{\mathrm{B}}_{\mathrm{Grp}^2 \times \mathrm{F} \times \mathrm{B}}$. We lift it again and have $\mathfrak{D}^{\mathrm{B}}_{\mathrm{SplitMono}}$. □

This yields a URG structure

$$\mathfrak{S}_{\mathrm{SplitMono} \times \mathrm{B}} :\equiv \int \mathfrak{D}^{\mathrm{B}}_{\mathrm{SplitMono}}$$

on the type

$$\mathrm{SplitMono} \times \mathrm{B} :\equiv (\langle\langle\langle G_0, G_1\rangle, \langle \iota, \sigma\rangle\rangle, \mathrm{split}^{\sigma}_{\iota}\rangle : \mathrm{SplitMono}) \times G_1 \to_{\mathrm{Grp}} G_0.$$

Lemma 4.15. *The split condition can be displayed over $\mathfrak{S}_{\mathrm{SplitMono} \times \mathrm{B}}$. In other words, there is a DURG structure $\mathfrak{D}^{\mathrm{isSecRet}}_{\mathrm{SplitMono} \times \mathrm{B}}$ on*

$$\lambda\, (\langle\langle\langle\langle G_0, G_1\rangle, \langle \iota, \sigma\rangle\rangle, \mathrm{split}^{\sigma}_{\iota}\rangle, \tau\rangle) \mapsto \mathrm{isSecRet}\langle \iota, \tau\rangle.$$

Proof. Note that here we cannot simply use the lifting operation from Proposition 3.20 on isSecRet, because of the additional components and different composition of Σ-types. However, $\mathrm{isSecRet}\langle\iota,\tau\rangle$ is still a proposition, so Proposition 3.14 yields the desired DURG structure. □

Finally, we have a URG structure

$$\int \mathfrak{D}^{\mathrm{isSecRet}}_{\mathrm{SplitMono}\times\mathrm{B}}$$

on the type

$$\mathrm{IntReflGraph} :\equiv (\langle\langle\langle\langle G_0, G_1\rangle, \langle\iota,\sigma\rangle\rangle, \mathrm{split}^{\sigma}_{\iota}\rangle, \tau\rangle : \mathrm{SplitMono}\times\mathrm{B}) \times \mathrm{isSecRet}\langle\iota,\tau\rangle.$$

Equivalence

In this section, let $\mathscr{F}$ denote the isomorphism of Action and SplitMono constructed in Theorem 4.11.

We apply Theorem 3.19 to $\mathfrak{D}^{\mathrm{isEquivariant}}_{\mathrm{Action}\times\mathrm{B}}$, the DURG structure of equivariance proofs over Action × B, to obtain a DURG structure $\mathfrak{D}^{\mathrm{B}\times\mathrm{isEquivariant}}_{\mathrm{Action}}$ on the family

$$\lambda\,\langle\langle\langle G_0, H\rangle, \alpha\rangle, \mathrm{isAct}\rangle \mapsto (\varphi : H \to_{\mathrm{Grp}} G_0) \times \mathrm{isEquivariant}\langle\alpha,\varphi\rangle.$$

Similarly, from $\mathfrak{D}^{\mathrm{isSecRet}}_{\mathrm{SplitMono}\times\mathrm{B}}$ we obtain a DURG structure $\mathfrak{D}^{\mathrm{B}\times\mathrm{isSecRet}}_{\mathrm{SplitMono}}$ on the family

$$\lambda\,\langle\langle\langle G_0, G_1\rangle, \langle\iota,\sigma\rangle\rangle, \mathrm{split}^{\sigma}_{\iota}\rangle \mapsto (\tau : G_1 \to_{\mathrm{Grp}} G_0) \times \mathrm{isSecRet}\langle\iota,\tau\rangle.$$

We want to construct a fiberwise relational isomorphism of $\mathfrak{D}^{\mathrm{B}\times\mathrm{isEquivariant}}_{\mathrm{Action}}$ and $\mathfrak{D}^{\mathrm{B}\times\mathrm{isSecRet}}_{\mathrm{SplitMono}}$ over $\mathscr{F}$. Hence, fix an action $\langle G_0, H, \alpha, \mathrm{isAct}\rangle$ and its corresponding split monomorphism $\langle G_0, H \rtimes_\alpha G_0, \iota_2, \pi_2, \mathrm{split}^{\pi_2}_{\iota_2}\rangle$. The maps transforming the equivariant homomorphism into the second retraction and vice versa are illustrated in the following diagram:

$$\begin{tikzcd}
G_0 \arrow[rr, dashed, bend left, "\alpha"] & & H \arrow[ll, "\varphi"] & {} \arrow[r, mapsto, "\text{Lemma 4.16}"] & {} & G_0 \arrow[r, "\iota_2" description] & H \rtimes_\alpha G_0 \arrow[l, shift right=2, "\pi_2"'] \arrow[l, shift left=2, "\tau_\varphi"]
\end{tikzcd}$$

$$\begin{tikzcd}
G_0 \arrow[r, "\iota_2" description] \arrow[rr, dashed, bend left=50, "\mathrm{Ad}_{\iota_2}"] & H \rtimes_\alpha G_0 \arrow[l, shift right=2, "\pi_2"'] \arrow[l, shift left=2, "\tau"] & \ker \pi_1 \arrow[l, hook, "\iota_1"'] \arrow[ll, bend left=50, "\tau \circ_{\mathrm{Grp}} \iota_1"] & {} & {} \arrow[l, mapsto, "\text{Lemma 4.17}"'] & G_0 \arrow[r, "\iota_2" description] & H \rtimes_\alpha G_0 \arrow[l, shift right=2, "\pi_2"'] \arrow[l, shift left=2, "\tau"]
\end{tikzcd}$$

Lemma 4.16. *Let $\varphi : H \to_{\mathrm{Grp}} G_0$ be an α-equivariant homomorphism. Then*

$$\begin{aligned} \tau_\varphi : H \rtimes_\alpha G_0 &\to G_0 \\ \langle h, g \rangle &\mapsto (\varphi\, h) g \end{aligned}$$

defines a retraction of ι_2.

Proof. The map τ_φ is a homomorphism, because

$$(\varphi(h(g \,\alpha\, h')))gg' = (\varphi\, h)(\varphi(g \,\alpha\, h'))gg' = (\varphi\, h)g(\varphi\, h')g^{-1}gg' = (\varphi\, h)g(\varphi\, h')g'$$

holds for all $\langle h, g\rangle\, \langle h', g'\rangle : H \rtimes_\alpha G_0$. This uses the homomorphism property and equivariance of φ. By Proposition 2.30, proving that τ_φ is split amounts to verifying that

$$(\varphi\, 1)g = g. \qquad \square$$

Lemma 4.17. *Let $\tau : H \rtimes_\alpha G_0 \to_{\mathrm{Grp}} G_0$ be a retraction of ι_2. Then there is an α-equivariant homomorphism $\varphi : H \to_{\mathrm{Grp}} G_0$.*

Proof. We define

$$\varphi :\equiv \tau \circ_{\mathrm{Grp}} \iota_1.$$

We briefly show α-equivariance. Let $g : G_0$ and $h : H$. Then

$$\begin{aligned}\varphi(g\,\alpha\,h) &\equiv \tau\langle g\,\alpha\,h, 1\rangle \\ &= \tau\langle 1(g\,\alpha\,h)((g1)\,\alpha\,1), g1g^{-1}\rangle \\ &= \tau(\langle 1, g\rangle \cdot_\alpha \langle h, 1\rangle \cdot_\alpha \langle 1, g^{-1}\rangle) \\ &= \tau\langle 1, g\rangle\,\tau\langle h, 1\rangle\,\tau\langle 1, g^{-1}\rangle \\ &= \tau(\iota_2\,g)\,\tau(\iota_1\,h)\,\tau(\iota_2\,g) \\ &= g(\varphi\,h)g^{-1}.\end{aligned}$$

□

Lemma 4.18. *There is a fiberwise relational isomorphism between* $\mathfrak{D}^{\mathrm{B}\times\mathrm{isEquivariant}}_{\mathrm{Action}}$ *and* $\mathfrak{D}^{\mathrm{B}\times\mathrm{isSecRet}}_{\mathrm{SplitMono}}$ *over* $\mathscr{F}$.

Proof. For the first direction, let $\varphi : H \to_{\mathrm{Grp}} G_0$ be an α-equivariant homomorphism. We need to verify that $\tau_\varphi \circ \iota_1 \sim \varphi$.

Unfolding the definitions, we find that the round-trip function $\tau_\varphi \circ \iota_1$ acts as

$$(\tau \circ \iota_1)h \equiv (\varphi\,h)1.$$

Conversely, let τ be another retraction of ι_2. We claim that τ is homotopic to its round-trip version

$$\tau_\varphi \equiv \lambda\,\langle h, g\rangle \mapsto \tau\langle h, 1\rangle g.$$

It is easy to see that

$$\begin{aligned}\tau\langle h, 1\rangle g &= \tau\langle h, 1\rangle\,\tau\langle 1, g\rangle \\ &= \tau\langle h(1\,\alpha\,1), 1g\rangle \\ &= \tau\langle h, g\rangle.\end{aligned}$$

Since the split property and equivariance condition are propositions, these two claims imply the result. □

Theorem 4.19. *There is an equivalence*

$$\mathrm{PreXModule} \simeq \mathrm{IntReflGraph}.$$

Proof. By Theorem 3.25, the fiberwise relational isomorphism of Lemma 4.18 induces an isomorphism of the underlying total spaces. Up to reassociating Σ-types, these total spaces are PreXModule and IntReflGraph, respectively. □

4.5 Crossed Modules and Peiffer Graphs

This secion establishes the equivalence of the type of crossed modules and that of Peiffer graphs. Peiffer graphs were first defined in [MM10] in the context of a semiabelian category. Mantovani and Metere also proved that in *good* semiabelian categories, such as the category of groups, the Peiffer condition is sufficient to identify the crossed modules among the precrossed modules.

In one picture, the current setting is:

$$\begin{tikzcd}
\mathrm{PFXM} \arrow[d] & \mathrm{PFG} \arrow[d] \arrow[dl, dashed] \\
\mathrm{PreXModule} \arrow[r, "\sim"] & \mathrm{IntReflGraph}
\end{tikzcd}$$

Here PFXM and PFG denote the Peiffer condition for crossed modules and internal reflexive graphs, respectively. As the word ‘condition’ suggests, being Peiffer is a mere proposition. This makes it easy to construct DURG structures and hence a fiberwise relational isomorphism over the equivalence from the previous section.

Crossed Modules

Definition 4.20. A *crossed module* is a precrossed module $\langle G_0, H, \alpha, \mathrm{isAct}, \varphi, \mathrm{isEqui}\rangle$ which additionally satisfies the *Peiffer condition for precrossed modules*

$$\mathrm{isPeiffer}\langle \alpha, \varphi\rangle :\equiv (h\,h' : H) \to (\varphi\, h)\, \alpha\, h' = hh'h^{-1}. \tag{PFXM}$$

Here, H is the object of the action α. We define the *type of crossed modules*

$$\mathrm{XModule} :\equiv (\langle\langle\langle\langle\langle G_0, H\rangle, \alpha\rangle, \mathrm{isAct}\rangle, \varphi\rangle, \mathrm{isEqui}\rangle) \times \mathrm{isPeiffer}\langle \alpha, \varphi\rangle.$$

Being a universally quantified statement about an equality in a group, (PFXM) is a proposition and can therefore be displayed over precrossed modules. Hence, we have an evident DURG $\mathfrak{D}^{\mathrm{PFXM}}_{\mathrm{PreXModule}}$.

Peiffer Graphs

Definition 4.21. Let $\langle G_0, G_1, \iota, \sigma, \mathrm{split}_{\iota}^{\sigma}, \tau, \mathrm{split}_{\iota}^{\tau} \rangle$ be an internal reflexive graph. The *Peiffer condition for internal reflexive graphs* is

$$(a\, b : G_1) \to (\iota(\sigma\, b))\, a\, (\iota(\sigma\, a^{-1}))(\iota(\sigma\, b^{-1}))\, b\, (\iota(\tau\, a)) = b\, a. \tag{PFG}$$

Let us visualize the Peiffer condition in terms of modified[5] pasting diagrams. After bringing $\iota(\sigma\, b)$ and $\iota(\tau\, a)$ to the other side of the equation, the left-hand side is:

$$\begin{tikzcd}
* \arrow[r, bend left, "\sigma\, a"{name=U}] \arrow[r, bend right, "\tau\, a"'{name=D}] \arrow[from=U, to=D, Rightarrow, "a"] & * & * \arrow[l, Rightarrow, "\iota(\sigma\, a^{-1})"'] & * \arrow[l, Rightarrow, "\iota(\sigma\, b^{-1})"'] \arrow[r, bend left, "\sigma\, a"{name=U2}] \arrow[r, bend right, "\tau\, b"'{name=D2}] \arrow[from=U2, to=D2, Rightarrow, "b"] & *
\end{tikzcd}$$

The right-hand side is:

$$\begin{tikzcd}
* & * \arrow[l, Rightarrow, "\iota(\sigma\, b^{-1})"'] \arrow[r, bend left, "\sigma\, b"{name=U}] \arrow[r, bend right, "\tau\, b"'{name=D}] \arrow[from=U, to=D, Rightarrow, "b"] & * \arrow[r, bend left, "\sigma\, a"{name=U2}] \arrow[r, bend right, "\tau\, a"'{name=D2}] \arrow[from=U2, to=D2, Rightarrow, "a"] & * & * \arrow[l, Rightarrow, "\iota(\tau\, a^{-1})"']
\end{tikzcd}$$

Of course, (PFG) is also a mere proposition. This gives rise to a DURG $\mathfrak{D}^{\mathrm{PFG}}_{\mathrm{IntReflGraph}}$. Taking its total space

$$\int \mathfrak{D}^{\mathrm{PFG}}_{\mathrm{IntReflGraph}},$$

we have characterized the identity types of the type PeifferGraph.

Equivalence

In this subsection, let $\chi :\equiv \langle G_0, H, \alpha, \mathrm{isAct}, \varphi, \mathrm{isEqui} \rangle$ be a precrossed module and $\mathcal{G} :\equiv \langle G_0, H \rtimes_{\alpha} G_0, \iota_2, \pi_2, \mathrm{split}_{\iota_2}^{\pi_2}, \tau, \mathrm{split}_{\iota_2}^{\tau_\varphi} \rangle$ the corresponding internal reflexive graph.

Lemma 4.22. *Assume χ satisfies* (PFXM). *Then $\mathcal{G}$ satisfies* (PFG).

[5] The modification is that identity arrows $\iota g : G_1$ are drawn as '⇒' (and their inverses as '⇐'), rather than vertical double arrows.

Proof. Let $a\,b : H \rtimes_\alpha G_0$ with $a \equiv \langle h, g\rangle$ and $b \equiv \langle h', g'\rangle$. We have to show that

$$\iota_2(\pi_2\, b) \cdot_\alpha a \cdot_\alpha \iota_2(\tau_\varphi\, a^{-1}) \cdot_\alpha \iota_2(\pi_2\, b^{-1}) \cdot_\alpha \iota_2(\tau_\varphi\, a) = b \cdot_\alpha a. \tag{4.5}$$

We compute the identities

$$\begin{aligned} \iota_2(\pi_2\, b) &= \langle 1, g'\rangle \\ \iota_2(\pi_2\, b^{-1}) &= \langle 1, g'^{-1}\rangle \\ \iota_2(\tau_\varphi\, a) &= \langle 1, (\varphi\, h)g\rangle. \end{aligned}$$

Using these identities and simplifying, we see that

$$\iota_2(\pi_2\, b) \cdot_\alpha a \cdot_\alpha \iota_2(\tau_\varphi\, a^{-1}) \cdot_\alpha \iota_2(\pi_2\, b^{-1}) \cdot_\alpha \iota_2(\tau_\varphi\, a) = \langle (g'\,\alpha\, h)((g'(\varphi\, h^{-1})g'^{-1})\,\alpha\, h'), g'g\rangle.$$

In the first component we apply equivariance of φ and see that

$$(g'\,\alpha\, h)((g'(\varphi\, h^{-1})g'^{-1})\,\alpha\, h') = (g'\,\alpha\, h)((\varphi(g'\,\alpha\, h^{-1}))\,\alpha\, h').$$

We now use the Peiffer identity (PFXM) and obtain

$$(g'\,\alpha\, h)((\varphi(g'\,\alpha\, h^{-1}))\,\alpha\, h') = (g'\,\alpha\, h)(g'\,\alpha\, h^{-1})h'(g'\,\alpha\, h^{-1})^{-1} = h'(g'\,\alpha\, h).$$

On the other hand, $\langle h'(g'\,\alpha\, h), g'g\rangle \equiv b \cdot_\alpha a$. Concatenating these identities gives a proof of (4.5). □

Lemma 4.23. *Assume that* $\mathcal{G}$ *satisfies* (PFG). *Then* χ *satisfies* (PFXM).

Proof. Let $h\,h' : H$. We want to prove that

$$(\varphi\, h)\,\alpha\, h' = hh'h^{-1}. \tag{4.6}$$

We apply (PFG) to $\langle h^{-1}, 1\rangle$ and $\langle h', 1\rangle$ and get

$$\iota_2(\pi_2\langle h', 1\rangle)\langle h^{-1}, 1\rangle\, \iota_2(\tau_\varphi\, \langle h^{-1}, 1\rangle^{-1})\, \iota_2(\pi_2\, \langle h', 1\rangle^{-1})\langle h', 1\rangle\, \iota_2(\tau_\varphi\, \langle h^{-1}, 1\rangle) = \langle h', 1\rangle \cdot_\alpha \langle h^{-1}, 1\rangle. \tag{4.7}$$

Clearly,

$$\iota_2(\pi_2\langle h', 1\rangle) = \langle 1, 1\rangle$$

and

$$\iota_2(\tau_\varphi\langle h^{-1}, 1\rangle) = \langle 1, \varphi\, h^{-1}\rangle.$$

Accordingly, (4.7) implies

$$\langle h^{-1}, 1\rangle \cdot_\alpha \langle 1, \varphi\, h\rangle \cdot_\alpha \langle h', 1\rangle \cdot_\alpha \langle 1, \varphi\, h^{-1}\rangle = \langle h', 1\rangle \cdot_\alpha \langle h^{-1}, 1\rangle.$$

Carrying out the multiplication on both sides and simplifying further, leaves us with

$$\langle h^{-1}((\varphi\, h)\, \alpha\, h'), 1\rangle = \langle h' h^{-1}, 1\rangle. \tag{4.8}$$

Projecting to the first component, and multiplying h on the left shows that (4.6) follows from (4.8). □

Theorem 4.24. *There is a fiberwise relational isomorphism between the DURGs $\mathfrak{D}^{\mathrm{PFXM}}_{\mathrm{PreXModule}}$ and $\mathfrak{D}^{\mathrm{PFG}}_{\mathrm{IntReflGraph}}$. Hence, there is an equivalence*

$$\mathrm{XModule} \simeq \mathrm{PeifferGraph}\,.$$

Proof. Since both Peiffer conditions are propositions, it suffices to show that they are logically equivalent. This is done in the previous two lemmas. □

4.6 Peiffer Graphs and Strict 2-Groups

A strict 2-group is an internal reflexive graph together with a vertical composition operation. Thus, we compare the Peiffer condition to the type of vertical composition operations that can be defined on an internal reflexive graph.

$$\begin{tikzcd}
\mathrm{PFG} \arrow[d] & \mathrm{VertComp} \arrow[d] \\
\mathrm{IntReflGraph} \arrow[r, "\mathrm{id}"] & \mathrm{IntReflGraph}
\end{tikzcd}$$

In that spirit, let us fix an internal reflexive graph

$$\mathcal{G} :\equiv \langle G_0, G_1, \iota, \sigma, \mathrm{split}^{\sigma}_{\iota}, \tau, \mathrm{split}^{\tau}_{\iota}\rangle$$

throughout the entire section.

Vertical Compositions

Definition 4.25. We say that two arrows $b\,a : G_1$ are *composable*, if $\sigma\,b = \tau\,a$. If $p : \text{isComposable}\ b\,a$, we call $\langle b, a, p\rangle$ a *composable triple*.
The *type of vertical compositions* $\mathscr{V}$ on $\mathscr{G}$ is the Σ-type consisting of the following components:

1. A composition operation

$$\_\circ_{\_}\_ : (b\,a : G_1) \to \text{isComposable}\ b\,a \to G_1$$

   (where the order of the last two arguments of $\circ$ is swapped, i.e., we write $b \circ_p a$, for $p : \text{isComposable}\ b\,a$);

2. witnesses that σ and τ respect $\circ$, i.e., for every composable triple $\langle b, a, p\rangle$ terms of type $\sigma(b \circ_p a) = \sigma\,a$ and $\tau(b \circ_p a) = \tau\,b$;

3. a proof that $\circ$ is a homomorphism, i.e., for all composable triples $\langle b, a, p\rangle$ and $\langle b', a', p'\rangle$, and every $\widetilde{p} : \text{isComposable}\,(b\,b')\,(aa')$, a term of type

$$(b\,b') \circ_{\widetilde{p}} (aa') = (b \circ_p a)(b' \circ_{p'} a');$$

4. a witness of associativity, i.e., a term of type

$$\begin{aligned}
&(c\,b\,a : G_1)\\
&(p^c_b : \text{isComposable}\ c\,b)\\
&(p^b_a : \text{isComposable}\ b\,a)\\
&(p^c_{ba} : \text{isComposable}\ c\,(b \circ_{p^b_a} a))\\
&(p^{cb}_a : \text{isComposable}\,(c \circ_{p^c_b} b)\,a)\\
\to\, &c \circ_{p^c_{ba}} (b \circ_{p^b_a} a) = (c \circ_{p^c_b} b) \circ_{p^{cb}_a} a;
\end{aligned}$$

5. left and right unit laws, that is, terms of type

$$(a : G_1) \to (p : \text{isComposable}\,(\iota(\tau\,a))\,a) \to \iota(\tau\,a) \circ_p a = a$$

   and

$$(b : G_1) \to (p : \text{isComposable}\ b\,(\iota(\sigma\,b)) \to b \circ_p \iota(\sigma\,b) = b.$$

The alert reader will have noticed that some of the parameters in the above types are superfluous. If $\langle b, a, p\rangle$ and $\langle b', a', p'\rangle$ are composable triples, then there is a term witnessing that bb' and aa' are composable. The same applies to p^{h}_{ba} and p^{hb}_{a} in the associativity axiom, as well as the composability condition in the unit laws. Those arguments exist, because they are *practical* when using equational reasoning patterns in chains of equations. The type $\mathscr{V}$ is equivalent to the one without these extra arguments.

The next proposition is the key ingredient to proving that $\mathscr{V}$ is a mere proposition. It expresses the vertical composition in terms of the multiplication in G_1 and rests on the homomorphism property of the vertical composition operation.

Proposition 4.26. *Let $\circ$ be (the operation of) a vertical composition on $\mathscr{G}$. Then*

$$b \circ_p a = b(\iota(\sigma\, b^{-1}))a$$

holds for all composable triples $\langle b, a, p\rangle$.

The classical proof can be found in [Por08, p. 6].

Proof. It is easy to see that

$$b \circ_p a = b(\iota(\sigma\, b^{-1}))(\iota(\sigma\, b)) \circ (\iota(\sigma\, b))(\iota(\sigma\, b^{-1}))a.$$

We suppress the witnesses that the arrows being composed are composable. Using the homomorphism property of $\circ$ once, we see that

$$(\iota(\sigma\, b^{-1}))(\iota(\sigma\, b)) \circ (\iota(\sigma\, b))(\iota(\sigma\, b^{-1}))a = (g(\iota(\sigma\, b^{-1})) \circ (\iota(\sigma\, b))(\iota(\sigma\, b^{-1})))(\iota(\sigma\, b) \circ a).$$

Another application of the interchange law gives

$$(b(\iota(\sigma\, b^{-1})) \circ (\iota(\sigma\, b))(\iota(\sigma\, b^{-1})))(\iota(\sigma\, b) \circ a) = (b \circ \iota(\sigma\, b))(\iota(\sigma\, b^{-1}) \circ \iota(\sigma\, b^{-1}))(\iota(\sigma\, b) \circ a).$$

Using the right unit law on the left and right compositions, we obtain

$$(b \circ \iota(\sigma\, b))(\iota(\sigma\, b^{-1}) \circ \iota(\sigma\, b^{-1}))(\iota(\sigma\, b) \circ a) = b(\iota(\sigma\, b^{-1}) \circ \iota(\sigma\, b^{-1}))a.$$

Note that $\iota(\sigma(\iota(\sigma\, b^{-1}))) = \iota(\sigma\, b^{-1})$. Hence, $\iota(\sigma\, b^{-1}) \circ \iota(\sigma\, b^{-1}) = \iota(\sigma\, b^{-1})$. Combining this chain of equations yields the desired identity. □

Proposition 4.27. *The type $\mathscr{V}$ of vertical compositions on $\mathscr{G}$ is a proposition.*

Proof. To construct a path between two composition operations $\circ$ and $\circ'$, we use function extensionality three times and then apply Proposition 4.26. This gives

$$b \circ_p a = b(\iota(\sigma\, b^{-1}))a = b \circ'_p a$$

for any composable triple $\langle b, a, p\rangle$. Note that the other axioms are universally quantified statements about identities in groups, hence propositions. Lemma 2.18 produces the desired dependent paths. □

Finally, we have a DURG $\mathfrak{D}^{\mathrm{VertComp}}_{\mathrm{IntReflGraph}}$ and a characterization $\int \mathfrak{D}^{\mathrm{VertComp}}_{\mathrm{IntReflGraph}}$ of the identity types of the type S2G of strict 2-groups.

Equivalence

Proposition 4.28. *Assume that* $\mathcal{G}$ *satisfies* (PFG). *Then*

$$b \circ_p a :\equiv b(\iota(\sigma\, b^{-1}))a$$

defines a vertical composition on $\mathcal{G}$.

Proof. The way $\circ$ is defined, it already satisfies all axioms of a vertical composition without the additional assumption (PFG), except for the interchange law. The proof of this claim amounts to a simple verification. The details can be found in the formalization. Thus, we only prove here that $\circ$ satisfies the interchange law

$$bb'(\iota(\sigma\,(bb')^{-1}))aa' = b(\iota(\sigma\, b^{-1}))ab'(\iota(\sigma\, b'^{-1}))a'$$

for composable b, a and b', a' in G_1. Since cancelling is an equivalence, it suffices to show that

$$b'(\iota(\sigma\,(bb')^{-1}))a = (\iota(\sigma\, b^{-1}))ab'(\iota(\sigma\, b'^{-1})). \tag{4.9}$$

Clearly, the Peiffer identity is equivalent to

$$(a\, b' : G_1) \to a(\iota(\tau\, a^{-1}))(\iota(\sigma\, b'^{-1}))b' = (\iota(\sigma\, b'^{-1}))b'a(\iota(\tau\, a^{-1})). \tag{4.10}$$

Inverting both sides, (4.10) implies

$$b'^{-1}(\iota(\sigma\, b'))(\iota(\tau\, a))a^{-1} = (\iota(\tau\, a))a^{-1}b'^{-1}\,\iota(\sigma\, b').$$

We substitute $a \mapsto a^{-1}$ and $b' \mapsto b'^{-1}$. After simplifying we have the equivalent identity

$$b'(\iota(\sigma\, b'^{-1}))(\iota(\tau\, a^{-1}))(a^{-1}) = (\iota(\tau\, a^{-1}))ab'(\iota(\sigma\, b'^{-1})). \tag{4.11}$$

(4.11) can easily be transformed to (4.9). □

Proposition 4.29. *If*

$$\_\circ\_ : (b\, a : G_1) \to \mathrm{isComposable}\ b\, a \to G_1$$

satisfies the axioms of a vertical composition, then the graph $\mathcal{G}$ satisfies (PFG).

Proof. Let $b\, b'\, a : G_1$ such that b and a are composable. The interchange law and definition of $\circ$ gives

$$b\, b'(\iota(\sigma\, b'^{-1}))(\iota(\sigma\, b^{-1}))a a' a'^{-1}. \tag{4.12}$$

We may substitute $a' :\equiv \iota(\sigma\, b')$ in (4.12) and cancel b on the left, as well as a' on the right. Then we have exactly

$$b'(\iota(\sigma\, b'^{-1}))(\iota(\sigma\, b^{-1}))a = (\iota(\sigma\, b^{-1}))a\, b'(\iota(\sigma\, b'^{-1})). \tag{4.13}$$

We invert both sides of (4.13) and have

$$a^{-1}(\iota(\sigma\, b))(\iota(\sigma\, b'))(\iota(\sigma\, b'^{-1})) = (\iota(\sigma\, b'))b'^{-1}a^{-1}(\iota(\sigma\, b)). \tag{4.14}$$

In (4.14) we substitute $b :\equiv b^{-1}, b' :\equiv b'^{-1}$ and $a :\equiv a^{-1}$. The result is

$$a(\iota(\sigma\, b^{-1}))(\iota(\sigma\, b'^{-1}))b' = (\iota(\sigma\, b'^{-1}))b' a(\iota(\sigma\, b^{-1})). \tag{4.15}$$

We now put $b :\equiv \iota(\tau\, a)$ and $b' :\equiv b$. Then

$$a(\iota(\tau\, a^{-1}))(\iota(\sigma\, b^{-1}))b = (\iota(\sigma\, b'^{-1}))b' a(\iota(\tau\, a^{-1})).$$

The Peiffer identity follows immediately. □

Theorem 4.30. *There are equivalences*

$$\mathrm{XModule} \simeq \mathrm{PeifferGraph} \simeq \mathrm{S2G}\,.$$

Proof. The first equivalence is from the previous section. The results of this section establish a fiberwise relational isomorphism between the Peiffer condition on graphs, and $\mathcal{V}$. An invocation of Theorem 3.25 gives the second equivalence. □

5 Higher Groups in Cubical Type Theory

This chapter introduces pointed types as DURG structures, followed by two equivalent definitions of pointed homotopies used to prove pointed function extensionality. After proving a theorem about truncated pointed fibrations, we move on to the definition of (n,k)-groups. We display homomorphisms over pairs of higher groups, characterizing their identity type along the way. This sets up the proof that the type of (n,k)-groups is an $(n+1)$-type. This is followed by a very short introduction to homotopy groups. A good reference is [Uni13, Section 8]. The last two sections introduce the first Eilenberg-MacLane space and use it to prove that pointed connected 1-types are equivalent to groups in the ordinary sense.

5.1 Pointed Types

Recall that we have a URG structure $\mathfrak{S}_{\mathrm{Type}}$ on the universe Type of interest, where two types are related if there is an equivalence between them. Reflexivity of this relation is witnessed by the identity equivalence of a type.

Proposition 5.1. *Pointedness can be displayed over universes, i.e., there is a DURG structure $\mathfrak{D}^*_{\mathrm{Type}}$ on the type family*

$$\lambda\,(A : \mathrm{Type}) \mapsto A.$$

Proof. Given two types $A B$: Type related by the equivalence $e : A \simeq B$, we define the displayed relation as

$$a \cong_e b \;:\equiv\; e\,a = b$$

for any $a : A$ and $b : B$.

Reflexivity of this relation over a fixed type is trivial.

The $\cong$-singleton over the identity equivalence of A at $a : A$ is exactly the ordinary singleton at a, and hence contractible. □

Note that displaying points over types is already beyond the scope of displayed categories. We define the types of pointed types

$$\mathrm{Type}_* :\equiv (A : \mathrm{Type}) \times A.$$

Taking the total space

$$\mathfrak{S}_{\mathrm{Type}_*} :\equiv \int \mathfrak{D}^*_{\mathrm{Type}}$$

we see that *pointed equivalences* characterize the identity type of pointed types.

Definition 5.2. A *pointed family* over a pointed type $\langle A, *_A \rangle$ is a type family $B : A \to \mathrm{Type}$ together with a base point $*_B : B\, *_A$. A *pointed section* f consists of a dependent function (also called f) of type $(a : A) \to B\, a$ combined with a witness $f_* : f(*_A) = *_B$ that f preserves the basepoint. We call the *type of pointed sections* $(a : \langle A, *_A \rangle) \to_* \langle B\, a, *_B \rangle$.

We usually leave the basepoint implicit and write $(a : A) \to_* B$. The concept of a homotopy between functions can be transferred to pointed sections.

Definition 5.3. Let $f\ g : (a : A) \to_* B\, a$. Define the two kinds of *pointed homotopies* between f and g as

$$f \sim_* g :\equiv (a : A) \to_* \langle f\, a = g\, a, f_* \cdot {g_*}^{-1} \rangle$$

and

$$f \sim^{\mathrm{P}}_* g :\equiv (H : f \sim g) \times \mathrm{PathP}\, (\lambda\, i \mapsto H *_A i = *_B)\, f_*\, g_*.$$

Unraveling the definition of pointed Π-types, we see that

$$f \sim_* g \equiv (H : f \sim g) \times (H *_A = f_* \cdot {g_*}^{-1}).$$

Thus, both definitions only differ in how they require the homotopy H to act on the path which witnesses that f preserves the basepoint. Pictorially, these are the squares the homotopies need a filler of:

$$\begin{tikzcd}
f *_A \arrow[r, "f_* \cdot {g_*}^{-1}"] \arrow[d, equal] & g *_A \arrow[d, equal] & g *_A \arrow[r, "g_*"] & *_B \arrow[d, equal] \\
f *_A \arrow[r, "H *_A"'] & g *_A & f *_A \arrow[u, "H *_A"] \arrow[r, "f_*"'] & *_B
\end{tikzcd} \tag{5.1}$$

Proposition 5.4. *For pointed sections $f\ g : (a : A) \to_* B\,a$, the two kinds of pointed homotopies are equivalent as types. In symbols, this reads*

$$(f \sim_* g) \simeq (f \sim_*^{\mathrm{P}} g).$$

Proof. Let $H : f \sim g$. To show that the types of the fillers of the squares in (5.1) are equivalent is to construct a path

$$\mathscr{P} : (H \ast_A = f_* \cdot {g_*}^{-1}) = (\mathrm{PathP}\,(\lambda\, i \mapsto H \ast_A i = \ast_B)\, f_*\, g_*).$$

To put the focus on the cubical argument, we give generic names to the relevant variables, such that the squares become

$$\begin{tikzcd} x \arrow[r, "q \cdot r^{-1}"] \arrow[d, equal] & y \arrow[d, equal] \\ x \arrow[r, "p"] & y \end{tikzcd} \tag{5.2}$$

and

$$\begin{tikzcd} y \arrow[r, "r"] & z \arrow[d, equal] \\ x \arrow[u, "p"'] \arrow[r, "q"] & z \end{tikzcd} . \tag{5.3}$$

Path inversion is an equivalence, since it is its own pseudo-inverse. It follows that the fillers of (5.2) are equivalent to

$$q \cdot r^{-1} = p.$$

Since $p = (p \cdot r) \cdot r^{-1}$, Lemma 2.6 shows that

$$(q \cdot r^{-1} = p) = (q \cdot r^{-1} = (p \cdot r) \cdot r^{-1}).$$

Path concatenation on both sides is an equivalence. Whence, there is a path of type

$$(q \cdot r^{-1} = (p \cdot r) \cdot r^{-1}) = (p^{-1} \cdot q = p^{-1} \cdot (p \cdot r)).$$

It is easy to construct paths $p^{-1} \cdot q = p^{-1} \cdot q \cdot \mathrm{refl}$ and $p^{-1} \cdot (p \cdot r) = r$. It follows from Lemma 2.5 and another application of Lemma 2.6 that

$$(p^{-1} \cdot q = p^{-1} \cdot (p \cdot r)) = (p^{-1} \cdot\cdot\, r \,\cdot\cdot\, \mathrm{refl} = r).$$

According to Lemma 2.4, the type $p^{-1} \cdot\cdot\, r \cdot\cdot\, \mathsf{refl} = r$ is equivalent to the fillers of (5.3). The concatenation of these paths, some of which are obtained from univalence, can be taken to be $\mathscr{P}$. Next, we define the function

$$\begin{aligned} \varphi : (f \sim_* g) &\to (f \sim_*^{\mathrm{P}} g) \\ \langle p_1, p_2 \rangle &\mapsto \langle p_1, \mathsf{transport}\, \mathscr{P} \rangle \end{aligned}$$

Transporting along a path is an equivalence by Proposition 2.9. Hence, φ is the total function of a fiberwise equivalence. This implies that φ is an equivalence as well (Theorem 2.10). □

Theorem 5.5. *For any two pointed sections $f\ g : (a : A) \to_* B\, a$, we have* pointed function extensionality*, that is*

$$(f \sim_*^{\mathrm{P}} g) \simeq (f = g)$$

and hence

$$(f \sim_* g) \simeq (f = g).$$

Proof. We define

$$\begin{aligned} \mathsf{funExtP}_* : (f \sim_*^{\mathrm{P}} g) &\to (f = g) \\ \langle H, H_* \rangle &\mapsto \lambda\, i \mapsto \langle \lambda\, a \mapsto H\, a\, i, H_* i \rangle. \end{aligned}$$

The first component, which generates a path of the underlying maps of f and g, is the same one as in the usual, unpointed function extensionality. The second one is by construction of $\sim_*^{\mathrm{P}}$ exactly a path between f_* and g_*, as needed. Its inverse is given by

$$\mathsf{funExtP}_*^{-1}\, p :\equiv \langle \lambda\, a\, i \mapsto (p\, i)_1\, a, \lambda\, i \mapsto (p\, i)_2 \rangle.$$

To see that this is an inverse, let $\langle H, H_* \rangle : (f \sim_*^{\mathrm{P}} g)$. The image of this term is $\lambda\, i \mapsto \langle \lambda\, x \mapsto H\, x\, i, H_* i \rangle$. We have to provide a path of type

$$\mathsf{funExtP}_*^{-1}(\lambda\, i \mapsto \langle \lambda\, a \mapsto H\, a\, i, H_* i \rangle) = \langle H, H_* \rangle. \tag{5.4}$$

The left-hand side is judgementally equal to

$$\langle \lambda\, a\, i \mapsto H\, a\, i, \lambda\, i \mapsto H_*\, i \rangle.$$

However, this is just $\langle H, H_* \rangle$, so refl solves (5.4). Conversely, let $p : f = g$. We have to construct a path of type

$$\mathsf{funExtP}_*\, \langle \lambda\, a\, i \mapsto (p\, i)_1\, a, \lambda\, i \mapsto (p\, i)_2 \rangle = p \tag{5.5}$$

The left-hand side is by definition

$$\lambda\, i \mapsto \langle \lambda\, a \mapsto (p\, i)_1 a, (p\, i)_2 \rangle.$$

However, this simplifies to p. In total, refl solves (5.5). □

We move on to prove [BDR18, Theorem 3].

Theorem 5.6. *For integers $n \geq -1$ and $k \geq -2$ let $\langle X, *_X \rangle : \mathrm{Type}_*^{>k+1}$ be a $(k+1)$-connected, pointed type and let $Y : X \to \mathrm{Type}_*^{\leq n+k}$ be a fibration of $(n+k)$-truncated, pointed types. Then the type of pointed sections, $(x : X) \to_* Y\, x$, is n-truncated.*

Proof. We prove this by induction on n. For the base case we have to show that the type of pointed sections is a mere proposition. Since it is pointed by the trivial section

$$\bar{s} :\equiv \langle \lambda\, x \mapsto *_{Y\,x}, \mathrm{refl} \rangle,$$

it must be contractible. We take $\bar{s}$ to be the center of contraction. Let s be another section. According to pointed function extensionality, it suffices to define a pointed homotopy $\bar{s} \sim_* s$. We define the k-connected map

$$\begin{aligned} f : \mathbf{1} &\to X \\ * &\mapsto *_X. \end{aligned}$$

It follows from the assumptions on the fibration Y, that $P : \lambda\, x \mapsto s\, x = \bar{s}\, x$ is a family of k-truncated types over X. Next, we apply the elimination principle for pointed connected types [Uni13, Lemma 7.5.7] to f and P. Together with the induction principle of $\mathbf{1}$ this yields an equivalence

$$e : (\bar{s} \sim s) \simeq (\mathbf{1} \to (\bar{s}\, *_X = s\, *_X)) \simeq (\bar{s}\, *_X = s\, *_X).$$

Recall that

$$\bar{s} \sim_* s \equiv (H : \bar{s} \sim s) \times (H\, *_X = \bar{s}_* \cdot {s_*}^{-1}).$$

It is easy to see that

$$H :\equiv e^{-1}(\mathrm{refl} \cdot {s_*}^{-1}) : \bar{s} \sim s$$

and that, viewing e as an isomorphism, its right inverse property at $\mathrm{refl} \cdot {s_*}^{-1}$ is of type $H\, *_X = \bar{s}_* \cdot {s_*}^{-1}$. This completes the base case.

To prove the result for $n+1$, assuming the n case as the induction hypothesis, it suffices to show that for any pointed section s, its self-identity type is n-truncated, according to Proposition 2.21. Pointed function extensionality proves that

$$(s = s) \simeq (s \sim_* s).$$

We construct an equivalence

$$(s \sim_* s) \simeq (x : X) \to_* \langle s \sim s, \mathrm{refl} \rangle.$$

Since it follows from the induction hypothesis that the right hand side is an n-truncated family, doing so will complete the proof.

Elements of $s \sim_* s$ consist of a homotopy $H : s \sim s$ and a path $p : H *_X = s_* \cdot {s_*}^{-1}$. Elements of the type on the right are pairs of a homotopy H of the same kind, and a path $H *_X = \mathrm{refl}$. It is now obvious that a constant map in the first component and an application of the cancellation law for paths in the second component is an equivalence of the desired type. □

5.2 Homotopy Groups

We introduce loop spaces and homotopy groups.

Definition 5.7. The *loop space* of a pointed type $\langle A, * \rangle$ is the pointed type

$$\Omega\langle A, * \rangle :\equiv \langle * = *, \mathrm{refl}_* \rangle.$$

For any $n \geq 1$, the n-tuple loop space $\Omega^n \langle A, * \rangle$ has the structure of a higher group with delooping the connected component of A at $*$. We can use the set truncation to obtain a set-level group.

Definition 5.8. For any pointed type $\langle A, * \rangle$ and $n \geq 1$ the *n-th homotopy group* consists of the type

$$\pi_n \langle A, * \rangle :\equiv \| \Omega^n \langle A, * \rangle \|_0$$

with group operation coming from path concatenation in $\Omega^n \langle A, * \rangle$.

We remark that from now on we shall only ever be concerned with the first homotopy group of a pointed connected 1-type. In that case, the set-truncation is superfluous and can be dropped.

5.3 Higher Groups

Types in HoTT may be viewed as ∞-groupoids by taking elements as objects, paths as morphisms and higher paths as higher morphisms. Hence, we can see pointed connected types as higher groups with carrier the loop space at the base point. The group operation is given by path composition, refl provides the neutral element and the inverse operation is path inversion. Higher paths witness unit, identity and cancellation laws. These higher paths are themselves subject to further coherence conditions. The main reference is [BDR18].

The same ideas hold true in Cubical Type Theory: an ∞-group is a pointed connected type BG, the *delooping* or *classifying space* of the *carrier* $\Omega\langle BG, *\rangle$.

Ordinary set-level groups can be recovered by requiring that BG be a 1-type, because in that case the group axioms are not subject to any non-trivial higher paths.

Double loop spaces are better behaved than single loop spaces. This is known as the *Eckman-Hilton argument* [Uni13, Theorem 2.1.6]. This fact motivates the definition of higher groups which can be delooped multiple times.

Lemma 5.9. *For any k and n there is a URG structure on the type*

$$(A : \mathrm{Type}) \times A \times \mathrm{isConn}_k A \times \mathrm{isTrunc}_n A$$

pointed, k-connected and n-truncated types.

Proof. Apply Corollary 3.21 twice and take the total space. □

Definition 5.10. For integers $n \geq 0$ and $k \geq 1$ we define the type

$$\langle n, k\rangle\,\mathrm{Grp} :\equiv (B^k G : \mathrm{Type}) \times B^k G \times \mathrm{isConn}_{k-1}\, B^k G \times \mathrm{isTrunc}_{n+k}\, B^k G$$

of *k-tuply groupal n-groupoids* or *(n, k)-groups*.

Set-level groups can be recovered as $(0, 1)$-groups. Definition 5.10 justifies calling the URG structure from Lemma 5.9 $\mathfrak{S}_{\langle n,k\rangle\,\mathrm{Grp}}$. We see that pointed equivalences characterize the identity type of (n, k)-groups.

A homomorphism of higher groups is a pointed map. We exhibit homomorphisms of higher groups as displayed over pairs of higher groups, giving another example of a DURG structure on a type family which is not 1-truncated.

Proposition 5.11. *For any n and k, homomorphisms of (n,k)-groups can be displayed over pairs of (n,k)-groups, i.e., there is a DURG structure on the type family*

$$\lambda\,\langle B^kG, B^kH\rangle \mapsto B^kG \to_* B^kH$$

over $\mathfrak{S}_{\langle n,k\rangle\,\mathrm{Grp}} \times_{\mathfrak{S}} \mathfrak{S}_{\langle n,k\rangle\,\mathrm{Grp}}$.

Proof. The setup for the displayed relation is analogous to that of ordinary group homomorphisms. In the diagram

$$\begin{tikzcd}
B^kG \arrow[r, "f", "*"' near end] \arrow[d, "p"', "\wr", "*"' near end] & B^kH \arrow[d, "q"', "\wr", "*"' near end] \\
B^kG' \arrow[r, "f'", "*"' near end] & B^kH'
\end{tikzcd}$$

of (n,k)-groups B^kG, B^kG', B^kH, B^kH', *pointed* maps f and f', and *pointed* equivalences p and q we define $f \cong_{\langle p,q\rangle} f'$ to be the type of *pointed* homotopies filling that square. It follows from the left and right unit laws for pointed function composition together with pointed function extensionality that this relation satisfies the axioms for a DURG structure. □

The total space construction for DURG structures implies that the identity type $f = f'$ of any two pointed maps $f\,f' : B^kG \to_* B^kH$ is equivalent to the type of homotopies of the underlying maps of f and f'.

Corollary 5.12. *For any* $B^kG\,B^kH : \langle n,k\rangle\,\mathrm{Grp}$*, the type of homomorphisms*

$$B^kG \to_* B^kH$$

is an n-type. Furthermore, $\langle n,k\rangle\,\mathrm{Grp}$ *is* $(n+1)$*-truncated.*

Proof. The first claim is a consequence of Theorem 5.6 and Proposition 5.11. Since pointed equivalences are a subtype of pointed maps, the second claim follows from the first. □

5.4 Eilenberg-MacLane Spaces

We define the first Eilenberg-MacLane space of a group as a higher inductive type and show that this construction is a right inverse to the first homotopy group. The main reference is [LF14]. We put the focus on the cubical aspects of the argument.

Definition 5.13. The *first Eilenberg-MacLane space* of a group G is the higher inductive type $\mathscr{E}_1\, G$, abbreviated to $\mathscr{E}$, with constuctors

$$\begin{aligned}
\mathrm{base}_{\mathscr{E}} &: \mathscr{E};\\
\mathrm{loop}_{\mathscr{E}} &: G \to \mathrm{base}_{\mathscr{E}} = \mathrm{base}_{\mathscr{E}};\\
\mathrm{comp}_{\mathscr{E}} &: (g\, h : G) \to \mathrm{PathP}\,(\lambda\, i\; \mapsto \mathrm{base}_{\mathscr{E}} = \mathrm{loop}_{\mathscr{E}} h\, i)\,(\mathrm{loop}_{\mathscr{E}} g)\,(\mathrm{loop}_{\mathscr{E}}(gh));\\
\mathrm{squash}_{\mathscr{E}} &: \mathrm{isTrunc}_1\, \mathscr{E}\,.
\end{aligned}$$

The $\mathrm{comp}_{\mathscr{E}}$ constructor fills the square

$$\begin{tikzcd}
a \arrow[r, "\mathrm{loop}_{\mathscr{E}}(gh)"] \arrow[d, equal] & c \\
a \arrow[r, "\mathrm{loop}_{\mathscr{E}} g"'] & b \arrow[u, "\mathrm{loop}_{\mathscr{E}} h"']
\end{tikzcd} .$$

This is for technical purposes sometimes more convenient than the equivalent condition of requiring paths between $\mathrm{loop}_{\mathscr{E}}(gh)$ and $(\mathrm{loop}_{\mathscr{E}} g) \cdot (\mathrm{loop}_{\mathscr{E}} h)$.

Lemma 5.14. *Let $g\, h : G$. Then there is a term*

$$\mathrm{comp}'_{\mathscr{E}} : \mathrm{loop}_{\mathscr{E}}(gh) = (\mathrm{loop}_{\mathscr{E}} g) \cdot (\mathrm{loop}_{\mathscr{E}} h).$$

Proof. Any two definitions of double composition on the same open box are equal. The double concatenation $\mathrm{refl} \cdot\cdot\, \mathrm{loop}_{\mathscr{E}} g \cdot\cdot\, \mathrm{loop}_{\mathscr{E}} h$ with filler obtained from the homogeneous composition operation, and $\mathrm{loop}_{\mathscr{E}}(gh)$ with filler $\mathrm{comp}_{\mathscr{E}}\, g\, h$, are two double compositions. □

The next lemma prepares the elimination principle of $\mathscr{E}$ (Theorem 5.16). It is an immediate consequence of identity induction (J).

Lemma 5.15. *Let $B : \mathscr{E} \to \mathrm{Type}^{\leq -1}$ be a propositional family over $\mathscr{E}$. Then, for any $p : x = y$, $b_x : B\,x$ and $b_y : B\,y$, there is a term*

$$\mathrm{elimEq} : \mathrm{PathP}\,(\lambda\, i \;\mapsto B\,(pi))\; b_x\; b_y.$$

In general, the elimination rule for a higher inductive type states that to define a function out of it, one has to give a point for every point constructor, a path for each path constructor and a higher path for each higher path constructor.

Theorem 5.16. *Let $B : \mathscr{E} \to \mathrm{Type}^{\leq 1}$ be a family of groupoids over $\mathscr{E}$, and $* : B\,\mathrm{base}_{\mathscr{E}}$. If there is a map*

$$\mathrm{toLoop} : (g : G) \to \mathrm{PathP}\,(\lambda\, i \;\mapsto B\,(\mathrm{loop}_{\mathscr{E}} g\; i))\; *\; *$$

such that the dependent square

$$\begin{tikzcd} * \arrow[r, "\mathrm{toLoop}(gh)"] & * \\ * \arrow[u, equal] \arrow[r, "\mathrm{toLoop}\,g"'] & * \arrow[u, "\mathrm{toLoop}\,h"'] \end{tikzcd} \tag{5.6}$$

has a filler toIsComp *for all $g\,h : G$, then there is a function $f : (x : \mathscr{E}) \to_* B\,x$.*

Proof. We define f by induction on $\mathscr{E}$. Take $f\,\mathrm{base}_{\mathscr{E}} :\equiv *_B$. The action of f on $\mathrm{loop}_{\mathscr{E}} g$ should be the same as toLoop. The obvious choice is

$$f\,(\mathrm{loop}_{\mathscr{E}}\, g\, i) :\equiv \mathrm{toLoop}\; g\, i,$$

where $g : G$ and $i : \mathbf{I}$. This is different from HoTT where dependent paths are only expressed in terms of transport.

Similarly, a proof of the homomorphism property – a dependent path over $\mathrm{comp}_{\mathscr{E}}$ – can be defined directly:

$$f\,(\mathrm{comp}_{\mathscr{E}}\, g\, h\, i\, j) :\equiv \mathrm{toIsComp}\; g\, h\, i\, j.$$

The fourth case, a path over $\mathrm{squash}_{\mathscr{E}}$, asks for a proof that B is a 1-type over $\mathscr{E}$. This follows immediately from $B\,x$ being a 1-type for all x. □

We remark that if B is constant over $\mathscr{E}$, Theorem 5.16 becomes the *recursion principle* of $\mathscr{E}$. In that case the square (5.6) is non-dependent and its fillers are equivalent to the type $\mathrm{toLoop}(gh) = \mathrm{toLoop}\, g \cdot \mathrm{toLoop}\, h$.

We further note that if B is a family of sets, toIsComp can always be constructed, being a path between identity proofs in a set. Similarly, if B is a propositional family, the argument toLoop becomes superfluous.

Using this we can prove that $\mathscr{E}$ is connected. Simply eliminate into the proposition $\mathrm{isConn}_0\, \mathscr{E}$.

Lemma 5.17. *The function* $\mathrm{loop}_{\mathscr{E}}$ *respects the neutral element, and inverses in G, i.e.,*

$$\mathrm{loop}_{\mathscr{E}} 1 = \mathrm{refl}$$

and

$$(g : G) \to \mathrm{loop}_{\mathscr{E}} g^{-1} = (\mathrm{loop}_{\mathscr{E}} g)^{-1}.$$

Proof. Both identities use the same idea as the standard proof showing that group homomorphisms preserve the unit element and inverses. The difference here is that the domain is a priori not a group, but merely an HIT, the paths of which are of course subject to the needed groupoid laws. We sketch the proof of the first identity. It is easy to see that

$$\mathrm{loop}_{\mathscr{E}} 1 = \mathrm{loop}_{\mathscr{E}} 1 \cdot \mathrm{refl} = \mathrm{loop}_{\mathscr{E}} 1 \cdot \mathrm{loop}_{\mathscr{E}} 1 \cdot (\mathrm{loop}_{\mathscr{E}} 1)^{-1}.$$

By $\mathrm{comp}'_{\mathscr{E}}$, we have that $\mathrm{loop}_{\mathscr{E}} 1 \cdot \mathrm{loop}_{\mathscr{E}} 1 = \mathrm{loop}_{\mathscr{E}}(1 \cdot 1) = \mathrm{loop}_{\mathscr{E}} 1$. Cancellation of paths finishes the proof of the first identity. □

Being a pointed connected 1-type we see that $\mathscr{E}$ is a $(0,1)$-group. The recursion principle of $\mathscr{E}$ can be more conveniently rephrased as follows. Any homomorphism of groups

$$G \to_{\mathrm{Grp}} \pi_1 \langle B, *_B \rangle$$

induces a homomorphism of $(0,1)$-groups

$$\mathscr{E} \to_* \langle B, *_B \rangle.$$

Theorem 5.18. *The group $\pi_1 \mathscr{E}$ is isomorphic to G.*

Proof. We construct a pseudo-isomorphism of the underlying types which respects the group operation using the encode–decode method (cf. [LS13]).

We have a constructor $\mathrm{loop}_{\mathscr{E}} : G \to \Omega\,\mathscr{E}$. Lemma 5.14 states that $\mathrm{loop}_{\mathscr{E}}$ respects the group operation. The goal is to build a type family $\mathrm{Codes} : \mathscr{E} \to \mathrm{Type}^{\leq 0}$ such that $(\mathrm{Codes}\,\mathrm{base}_{\mathscr{E}}) = G$, because the function

$$\begin{aligned} &\mathrm{encode} : (x : \mathscr{E}) \to (\mathrm{base}_{\mathscr{E}} = x) \to \mathrm{Codes}\,x \\ &\mathrm{encode}_x\, p \;:\equiv\; \mathrm{subst}_{\mathrm{Codes}}\; p\, 1 \end{aligned}$$

at $x \equiv \mathrm{base}_{\mathscr{E}}$ gives rise to a potential inverse to $\mathrm{loop}_{\mathscr{E}}$.

The family Codes is defined by $\mathscr{E}$-recursion into the groupoid $\mathrm{Type}^{\leq 0}$ with basepoint $\langle G, \mathrm{set}_G\rangle$. The recursion principle asks for a map

$$\varphi : G \to (\langle G, \mathrm{set}_G\rangle = \langle G, \mathrm{set}_G\rangle)$$

such that

$$\begin{tikzcd} \langle G, \mathrm{set}_G\rangle \arrow[r, "\varphi(gh)"] \arrow[d, equal] & \langle G, \mathrm{set}_G\rangle \\ \langle G, \mathrm{set}_G\rangle \arrow[r, "\varphi\, g"'] & \langle G, \mathrm{set}_G\rangle \arrow[u, "\varphi\, h"'] \end{tikzcd}$$

can be filled for all $g\,h : G$. Since being truncated is a proposition and by univalence, φ can be obtained from

$$\begin{aligned} \varphi' : G &\to (G \to G) \\ g &\mapsto \lambda\, h \mapsto hg, \end{aligned}$$

combined with any proof that φ' is a fiberwise equivalence (of carrier types). Let $g\,h : G$. It is sufficient fill the simpler square:

$$\begin{tikzcd} G \arrow[r, "\varphi'(gh)"] \arrow[d, equal] & G \\ G \arrow[r, "\varphi'\, g"'] & G \arrow[u, "\varphi'\, h"'] \end{tikzcd} \tag{5.7}$$

By Lemma 2.4, the fillers of (5.7) are identical to the paths of type

$$\mathrm{refl} \cdot\!\cdot\; \varphi'\, g \cdot\!\cdot\; \varphi'\, h = \varphi'(gh).$$

Such a path can be constructed using that $\mathrm{idToEquiv}^{-1}$ respects path composition and that φ' sends a composition in G to a composition of equivalences. This completes the definition of Codes.

In total, we compute

$$\mathrm{encode}_{\mathrm{base}_{\mathscr{E}}}(\mathrm{loop}_{\mathscr{E}}\, g) \equiv \mathrm{subst}_{\mathrm{Codes}}\,(\mathrm{loop}_{\mathscr{E}}\, g)\, 1 = g,$$

so $\mathrm{encode}_{\mathrm{base}_{\mathscr{E}}}$ indeed defines a left inverse to $\mathrm{loop}_{\mathscr{E}}$.

We show that it is also a right inverse. The function $\mathrm{loop}_{\mathscr{E}}$ can be generalized to

$$\mathrm{decode} : (x : \mathscr{E}) \to \mathrm{Codes}\, x \to \mathrm{base}_{\mathscr{E}} = x$$

using the elimination principle of $\mathscr{E}$. This requires a base point in the fiber of that family over $\mathrm{base}_{\mathscr{E}}$. The obvious choice is

$$\mathrm{decode}_{\mathrm{base}_{\mathscr{E}}} :\equiv \mathrm{loop}_{\mathscr{E}}.$$

By Proposition 2.24, $\mathrm{Codes}\, x \to \mathrm{base}_{\mathscr{E}} = x$ is a set. It follows that we need to supply a proof that for any $g : G$ there is a path of type

$$\mathrm{PathP}\,(\lambda\, i \,\mapsto \mathrm{Codes}(\mathrm{loop}_{\mathscr{E}}\, g\, i) \to \mathrm{base}_{\mathscr{E}} = \mathrm{loop}_{\mathscr{E}}\, g\, i)\, \mathrm{loop}_{\mathscr{E}}\, \mathrm{loop}_{\mathscr{E}}$$

Such a path can be obtained from applying Lemma 2.15 to the equivalence $\lambda\, h \,\mapsto hg : G \simeq G$ and $\mathrm{comp}_{\mathscr{E}}$. This completes the definition of decode.

A simple path induction with Lemma 5.17 in the base case shows that encode is a right inverse of decode. In particular, $\mathrm{encode}_{\mathrm{base}_{\mathscr{E}}}$ is a right inverse of $\mathrm{decode}_{\mathrm{base}_{\mathscr{E}}}$. □

5.5 Delooping Groups

The goal of this section is to prove that there is an equivalence

$$\begin{tikzcd} \langle 0,1\rangle\,\mathrm{Grp} \arrow[r, shift left, "\pi_1"] & \mathrm{Grp} \arrow[l, shift left, "\mathscr{E}"] \end{tikzcd}.$$

To do so we construct a relational isomorphism between the underlying graphs of $\mathfrak{S}_{\mathrm{Grp}}$ and $\mathfrak{S}_{\langle 0,1\rangle\,\mathrm{Grp}}$ using parts of the adjunction

$$(\mathscr{E}_1\, H \to_{*} BG) \simeq (H \to_{\mathrm{Grp}} \pi_1\, BG).$$

for $H : \mathrm{Grp}$ and $BG : \langle 0,1\rangle\,\mathrm{Grp}$.

Proposition 5.19. *There is a map*

$$\varphi : (\pi_1(\mathscr{E}_1 H) \to_{\mathrm{Grp}} \pi_1 BG) \to (\mathscr{E}_1 H \to_* BG)$$

which restricts to isomorphisms in the sense that if $f : \pi_1(\mathscr{E}_1 H) \simeq_{\mathrm{Grp}} \pi_1 BG$, then φf is a pointed equivalence.

Proof. Let $f : \pi_1(\mathscr{E}_1 H) \to_{\mathrm{Grp}} \pi_1 BG$. From Theorem 5.18 we have a $g : H \to_{\mathrm{Grp}} \pi_1(\mathscr{E}_1 H)$. Put

$$h :\equiv f \circ_{\mathrm{Grp}} g : H \to_{\mathrm{Grp}} \pi_1 BG.$$

The type BG is a 1-type, so by $\mathscr{E}_1 H$-recursion we have a map $h' : \mathscr{E}_1 H \to BG$. Pointedness of h' is trivial.

Being a map between pointed connected types, h' is surjective. To see this, let $z : BG$. We need to show that

$$\|(x : \mathscr{E}_1 H) \times (h' x = z)\|_{-1}.$$

By definition, this means that

$$\|(x : \mathscr{E}_1 H) \times \|h' x = z\|_{-1}\|_{-1}.$$

We choose $\mathrm{base}_{\mathscr{E}}$ for the first component and show that $\|h' \mathrm{base}_{\mathscr{E}} = z\|_{-1}$ is contractible. This is equivalent to the statement that $h' \mathrm{base}_{\mathscr{E}} = z$ is (-1)-connected, but this follows from the connectedness of BG.

Assume now that f is an isomorphism to begin with. If we can show that h' is an embedding, then it is also an equivalence (cf. [Uni13, Theorem 4.6.3]).

We claim that it is sufficient to check that $\mathrm{ap}_{h'} : (x = y) \to (h' x = h' y)$ is an equivalence at $x \equiv y \equiv \mathrm{base}_{\mathscr{E}}$. This follows from two applications of the elimination principle for pointed connected types, because the map basepoint is (-1)-connected and being an embedding is a proposition.

We are now in the situation

$$\begin{tikzcd} H \arrow[r, "e"] & \pi_1(\mathscr{E}_1 H) \arrow[r, shift left, "f"] \arrow[r, shift right, "\mathrm{ap}_{h'}"'] & \pi_1 BG \end{tikzcd}$$

where e is the equivalence constructed in Theorem 5.18. By construction we have that the functions $f \circ e$ and $\mathrm{ap}_{h'}$ are trivially homotopic. The 2-out-of-3 property of equivalences (cf. [Uni13, Theorem 4.7.1]) implies that $\mathrm{ap}_{h'}$ is an equivalence. □

Theorem 5.20. *The functors $\mathscr{E}_1$ and π_1 define an equivalence between $\langle 0,1\rangle\,\mathrm{Grp}$ and Grp.*

Proof. The left inverse property is given by Theorem 5.18. Conversely, let $BG : \langle 0,1\rangle\,\mathrm{Grp}$. Theorem 5.18 also yields an isomorphism f between the groups $\pi_1(\mathscr{E}_1(\pi_1 BG))$ and $\pi_1 BG$. We can apply the previous proposition to $H :\equiv \pi_1 BG$ and f to see that $\mathscr{E}_1(\pi_1 BG) \simeq_* BG$.

We have thus constructed a relational isomorphism between $\mathfrak{S}_{\langle 0,1\rangle\,\mathrm{Grp}}$ and $\mathfrak{S}_{\mathrm{Grp}}$. This induces an equivalence on the underlying types. □

6 Formalization

This chapter is about the formalization of the results of the previous chapters in Cubical Agda. General references for univalent foundations in Agda are [Esc20] and [Ang+20].

6.1 The Code

My results are developed on a fork of the official Cubical Agda library.[1] They are publicly available on GitHub.[2] My code is about 7700 lines on top of the Cubical Agda library. For convenience,[3] all results are imported to a single file.[4] On the day of submission[5] that file has been tested with the latest Agda version[6]. It contains a list of all results of this paper and links to their implementation.

6.2 Performance

The file Cubical/Papers/HigherGroupsViaDURG.agda takes about 30 seconds to type-check. The project uses most of the tools Agda offers the performance of type checking.

The abstract keyword can be used to hide the implementation details of irrelevant terms such as proofs of propositions.

[1] `https://github.com/Agda/cubical`

[2] `https://github.com/Schippmunk/cubical`

[3] The Emacs and Atom Agda modes provide functionality to jump to the definition of a name currently in scope.

[4] Cubical/Papers/HigherGroupsViaDURG.agda

[5] Commit ID: b3609c07620847db571ac900b6fa8bfc0f7733d4

[6] 2.6.2—8ed0362

Another way of preventing a definition from expanding when reduced is the use of copatterns. When elements of records are defined using the record keyword, Agda still computes a normal form, which is in many cases quite large. In the case of DURG structures this prevents some of the operations from type checking within multiple hours. With copatterns, Agda merely has to check equality on the arguments.

6.3 Conventions

We highlight some notable code conventions of the library. Sometimes these deviate from the standard literature.

- In the library, truncation levels are shifted by 2 to start at 0. So, 0-types are contactible, 1-types propositions, 2-types sets, etc. This is because an implementation of the natural numbers is built into Agda, and that allows for use of actual numbers to be used as terms of $\mathbb{N}$, rather than n-fold applications of $S_{\mathbb{N}}$ to $0_{\mathbb{N}}$.
- Furthermore, in the library, a proposition is *defined* to be a type satisfying $(x\ y : X) \to x = y$, rather than the equivalent statement that all identity types are contractible. This is advantageous, because the chosen definition is used more often. However, proving something for all truncation levels now requires two case distinctions instead of one.
- There is a convention in the cubical library for the group operation to be written additively and we adopted this convention. In Agda, overloading of symbols is limited and $-_0\, x$ looks better than $\mathrm{inv}_0\, x$ or even x^{-1_0}. Similarly, the multiplication sign $\cdot$ cannot be dropped, unlike in informal mathematics. A dot could also be confused with the path concatenation operation.
- When formalizing composite structures in Cubical Agda one has to decide whether to use Σ- or record-types. The former are more flexible when constructing paths, but also slower. The synthesis is that important types such as Grp are incarnated as both record- and Σ-types.

7 Conclusion

In this chapter we shall dwell on what we learned about the framework provided by DURGs and present some potential future work.

7.1 Discussion

When we began the project the intention was only to internalize the equivalence of strict 2-groups and crossed modules. Constructing the maps in both directions was manageable, but when trying to prove that these maps were mutual inverses, we ran into two problems: the time it took to type-check was unacceptable and constructing paths between the transported structures was very difficult.

We realized a more systematic approach was necessary. The lack of category theory developed in the library at the time inspired the definition of displayed URGs. These also had performance issues which could, however, easily be resolved by the use of copatterns. All things considered, DURGs were very successful in providing a systematic proof of the equivalence and characterization of the identity types of our multi-component structures.

That being said, note that there is no guarantee we found the 'most cubical' proofs in this work. For example, there might be a proof of Proposition 5.4 that only uses groupoid laws like a generalization of Lemma 2.6 and does not rely on univalence – which would probably be avoided in favour of direct cubical arguments to qualify as a 'proof from the book' [AZ98].

In the same spirit, many proofs from the last two sections of Chapter 4 about identities in groups were very tedious to prove by hand. A ring solver could have vastly reduced the time it took to formalize the details.

In what follows we collect some general aspects and comparisons related to DURGs.

DURGs in General

- For any graph $\cong : A \to A \to \text{Type}$ there is an equivalence

  $$((\rho : (a : A) \to a \cong a) \times \text{isUnivalent} \cong \rho) \simeq ((a\, a' : A) \to (a = a') \simeq (a \cong a')).$$

  Hence, one could drop the reflexivity field and speak of (displayed) univalent graphs. It is not clear wheter this shortens the code, since URGs are – except for very simple cases – created by showing that the relational singleton is contractible. If nothing else, it at least shortens the name of the concept.

- When the process of adding structure to a type using DURGs is repeated, one has to make the decision of 'how to layer the cake'. To keep the univalence proofs simple, it makes sense to implement a layering as fine as possible. This practice also promotes code reuse and modularity.

  It is worthy of remark that we did not always live up to this standard ourselves. The bottom level of our towers of structure in Chapter 4 is the type of groups – the SIP of which had been obtained from the standard notion of structure (see below). Any library aiming to formalize a lot of algebraic structures should avoid this dependence on the theory of the standard notion.

- Development in Cubical Agda is sometimes slowed down due to the lack of code conventions. The abstraction and uniformity DURGs offer simplify the issue of finding a good convention.

- Reflexive graphs form an $(\infty, 1)$-topos with displayed reflexive graphs corresponding exactly to morphisms of reflexive graphs. DURGs form a sub-$(\infty, 1)$-topos thereof. This enables the use of topos theory to produce theorems about DURGs. For example, the construction in Corollary 3.21 corresponds to the categorical pullback.

- Bentzen [Ben19] prepares the programme of naive cubical reasoning and suggests to vastly extend it. If DURGs can be established across the different type theories, the need for such a programme is limited to what one needs to prove enough meta-properties of DURGs.

DURGs Versus Displayed Categories

- The main advantage of DURGs is that they are not limited to 1-truncated type families. While it should be possible to construct displayed higher categories, their internalization in type theory is most likely cumbersome to work with.
- Another advantage is that DURGs are easier to set up. Their definition and basic operations merely require some basic properties of univalence, while quite some category theory needs to be developed in the given proof assistant in order to make good use of displayed categories.
- Every univalent category induces a URG on its type of objects, so if one is only interested in the invertible morphisms, URGs might be the preferred framework. However, displayed categories likely serve as a better foundation for work with non-invertible morphisms and general categorical constructions.

DURGs Versus the Standard Notion of Structure

- In the introduction we hinted at other notions of structure. We briefly describe the 'standard notion' due to Escardó [Esc20]. Consider a type family $S : \mathrm{Type} \to \mathrm{Type}$. The type of S-structures is $(A : \mathrm{Type}) \times S\,A$. Such a pair $\langle A, s\rangle$ means that the type A is equipped with an S-structure, witnessed by s. The procedure of building a structure is similar, but the tower of Σ-types associates the other way. For instance, the type of types with an associative binary operation is presented as

  $$(A : \mathrm{Type}) \times ((+ : A \to A \to A) \times ((a\,b\,c : A) \to (a+b)+c = a+(b+c))),$$

  while a tower of DURGs produces a URG structure on

  $$(\langle A, +\rangle : (A : \mathrm{Type}) \times A \to A \to A) \times ((a\,b\,c : A) \to (a+b)+c = a+(b+c)).$$

- It is immediate that DURGs need more projections to obtain the carrier type – standard structures always need only one. Conversely, the forgetful functions mapping a structure to a weaker one have a shorter definition in the framework of DURGs.
- Every standard structure on a type is equivalently a DURG over the URG of equivalences on the universe, so DURGs are at least as general as the standard notion. In its current form, the standard notion does not properly capture structures with more than one carrier type or hierarchies shaped like diamonds [SUM20].

While more operations could be added to the standard notion in order to support more complex structures, DURGs do not have these issues in the first place. Hence, they might be a better choice.

7.2 Future Work

DURGs show a lot of potential in uniformly performing and understanding constructions in type theory. Thus, it is worthwile to explore more of their properties.

DURGs in General

- There are many common constructions, such as homotopy pullbacks, which should be revisited in the context of DURGs. These are not limited to mathematical objects, as the example of lists together with their different implementations shows (cf. [Ang+20]). Along the way, more operations making the process of formalizing more convenient will arise – for example, a symmetric analogue to Theorem 3.25 or a proof that Theorem 3.19 defines an equivalence.

  An important item on the list of constructions to be revisited in this context are categories. Tools to transfer results between categories and URGs are desirable for any library wishing to support both. Possible approaches include constructing a URG structure on the type of categories or defining a category structure on the type of URGs.

- One could construct a URG structure on the type of URG and even DURG structures. Doing so would allow the framework of DURGs to do its own meta-theory.

  Related to this is that it should be made precise to what extent DURGs form a model of type theory within itself.

- Agda's reflection mechanism can likely be used to simplify and automate parts of the process of constructing DURG structures. Since record-types are more performant, it would be advantageous to be able to automatically identify them with their corresponding nested Σ-type. Algebraic hierarchies naturally carry a directed acyclic graph structure. Thus, reflection can probably be used to generalize operations on them.

- It might be profitable to define an alternate Σ-type naturally associating to the left, because the deeply nested Σ-types sometimes become confusing. Even better would be a type theory in which composite Σ-types are judgementally equal to their differently associated versions.
- Another starting point for finding more theorems about DURGs is higher topos theory, as suggested in the discussion above.

Higher Groups

This thesis shows how the development of higher groups in a cubical setting can be progressed using DURGs.

The components of our implementation of the theory of higher groups in Cubical Agda are limited to the ingredients to the proof that groups are $(0, 1)$-groups. Thus, it makes sense to formalize more foundations like higher Eilenberg-MacLane spaces and group actions of higher groups before tackling open problems.

Either way, there are many open problems in the topic of higher groups; see [BDR18, p.24]. One possible next goal is to construct the map that turns a strict 2-group into a $(1, 1)$-group as a higher inductive type.

Bibliography

[Acz12] Peter Aczel. *Homotopy Type Theory and the Structure Identity Principle*. Feb. 2012. url: `https://www.newton.ac.uk/files/seminar/20120207160016301-153011.pdf`.

[AHH18] Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. "Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities". In: *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*. Ed. by Dan Ghica and Achim Jung. Vol. 119. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 6:1–6:17. isbn: 978-3-95977-088-0. doi: `10.4230/LIPIcs.CSL.2018.6`.

[AL19] Benedikt Ahrens and Peter Lefanu Lumsdaine. "Displayed Categories". In: *Logical Methods in Computer Science* 15.1, 20 (2019). doi: `10.23638/LMCS-15(1:20)2019`.

[Ang+20] Carlo Angiuli, Evan Cavallo, Anders Mörtberg, and Max Zeuner. *Internalizing Representation Independence with Univalence*. 2020. arXiv: `2009.05547 [cs.PL]`.

[Awo13] Steve Awodey. "Structuralism, Invariance, and Univalence†". In: *Philosophia Mathematica* 22.1 (Oct. 2013), pp. 1–11. issn: 0031-8019. doi: `10.1093/philmat/nkt030`.

[AZ98] Martin Aigner and Günter M. Ziegler. *Proofs from the Book*. 1998.

[BDR18] Ulrik Buchholtz, Floris van Doorn, and Egbert Rijke. "Higher Groups in Homotopy Type Theory". In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '18. Oxford, United Kingdom: Association for Computing Machinery, 2018, pp. 205–214. isbn: 9781450355834. doi: `10.1145/3209108.3209150`.

[Ben19] Bruno Bentzen. *Naive cubical type theory*. 2019. url: http://philsci-archive.pitt.edu/17148/.

[Buc19] Ulrik Buchholtz. "Higher Structures in Homotopy Type Theory". In: *Reflections on the Foundations of Mathematics: Univalent Foundations, Set Theory and General Thoughts*. Ed. by Stefania Centrone, Deborah Kant, and Deniz Sarikaya. Cham: Springer International Publishing, 2019, pp. 151–172. isbn: 978-3-030-15655-8. doi: 10.1007/978-3-030-15655-8_7.

[CD13] Thierry Coquand and Nils Anders Danielsson. "Isomorphism is equality". In: *Indagationes Mathematicae* 24.4 (2013). In memory of N.G. (Dick) de Bruijn (1918–2012), pp. 1105–1120. issn: 0019-3577. doi: https://doi.org/10.1016/j.indag.2013.09.002.

[CHM18] Thierry Coquand, Simon Huber, and Anders Mörtberg. "On Higher Inductive Types in Cubical Type Theory". In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '18. Oxford, United Kingdom: Association for Computing Machinery, 2018, pp. 255–264. isbn: 9781450355834. doi: 10.1145/3209108.3209197.

[Coh+18] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. "Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom". In: *21st International Conference on Types for Proofs and Programs (TYPES 2015)*. Ed. by Tarmo Uustalu. Vol. 69. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 5:1–5:34. isbn: 978-3-95977-030-9. doi: 10.4230/LIPIcs.TYPES.2015.5.

[DF04] David Steven Dummit and Richard M Foote. *Abstract algebra*. Vol. 3. Wiley Hoboken, 2004.

[Esc20] Martin Escardo. *Introduction to Univalent Foundations of Mathematics with Agda*. https://www.cs.bham.ac.uk/~mhe/HoTT-UF-in-Agda-Lecture-Notes/HoTT-UF-Agda.html. 2020.

[LF14] Daniel R. Licata and Eric Finster. "Eilenberg-MacLane Spaces in Homotopy Type Theory". In: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. CSL-LICS '14. Vienna, Austria: Association for Computing Machinery, 2014. isbn: 9781450328869. doi: 10.1145/2603088.2603153.

[LS13] Daniel R. Licata and Michael Shulman. *Calculating the Fundamental Group of the Circle in Homotopy Type Theory*. 2013. arXiv: 1301.3443 [math.LO].

[MM10] Sandra Mantovani and Giuseppe Metere. "Internal crossed modules and Peiffer condition". In: *Theory and Applications of Categories [electronic only]* 23 (Jan. 2010).

[Ort19] Richard Ian Orton. "Cubical Models of Homotopy Type Theory - An Internal Approach". Thesis. University of Cambridge, 2019. doi: 10.17863/CAM.36690.

[Por08] Sven-S. Porst. *Strict 2-Groups are Crossed Modules*. 2008. arXiv: 0812.1464 [math.CT].

[Rau15] Jakob von Raumer. "Formalization of Non-Abelian Topology for Homotopy Type Theory". MA thesis. May 2015.

[Rij18] Egbert Rijke. *Introduction to Homotopy Type Theory*. https://github.com/EgbertRijke/HoTT-Intro/blob/master/pdfs/2018-hott-intro-course.pdf. 2018.

[Rij19] Egbert Rijke. "Classifying Types". In: *arXiv e-prints* (June 2019). eprint: 1906.09435 (math.LO).

[SUM20] Daniel Selsam, Sebastian Ullrich, and Leonardo de Moura. *Tabled Typeclass Resolution*. 2020. arXiv: 2001.04301 [cs.PL].

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Johannes Philipp Manuel Schipp von Branitz, die vorliegende Master-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Darmstadt, October 22, 2020

J. Schipp von Branitz

Thesis Statement
pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Johannes Philipp Manuel Schipp von Branitz, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.